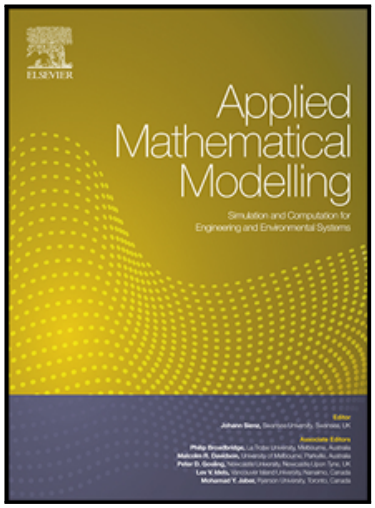
Journal Pre-proof



PORA: A Physarum-Inspired Obstacle-Avoiding Routing Algorithm for Integrated Circuit Design

Wenzhong Guo, Xing Huang

PII: S0307-904X(19)30615-8
DOI: https://doi.org/10.1016/j.apm.2019.10.027
Reference: APM 13088

To appear in: *Applied Mathematical Modelling*

Received date: 29 January 2019
Revised date: 3 October 2019
Accepted date: 8 October 2019

Please cite this article as: Wenzhong Guo, Xing Huang, PORA: A Physarum-Inspired Obstacle-Avoiding Routing Algorithm for Integrated Circuit Design, *Applied Mathematical Modelling* (2019), doi: https://doi.org/10.1016/j.apm.2019.10.027

Highlights

- Modelling and simulation of the biological behaviors of Physarum polycephalum.
- Proposing a Physarum-inspired obstacle-avoiding rectilinear Steiner tree construction algorithm.
- High-efficiency heuristics are proposed to enhance the performance of the Physarum computing model.
- The proposed obstacle-avoiding routing algorithm leads to shorter wirelength compared to several existing methods.

Appl. Math. Model. manuscript No.
(will be inserted by the editor)

PORA: A Physarum-Inspired Obstacle-Avoiding Routing Algorithm for Integrated Circuit Design

Wenzhong Guo[1,2] · **Xing Huang**[2,*]

Received: Jan 2019 / Accepted: MONTH YEAR

Abstract The plasmodium of Physarum polycephalum, a large, amoeboid cell, has attracted much attention recently due to its intelligent behaviors in pathfinding, danger avoidance, and network construction. Inspired by the biological behaviors of this primitive organism, in this study, we explore the optimization capability of Physarum polycephalum systematically and present the first Physarum-inspired obstacle-avoiding routing algorithm for the physical design of integrated circuits. We simulate the foraging behaviors of Physarum polycephalum using a novel nutrition absorption/consumption mathematical model, thereby presenting an efficient routing tool called *Physarum router*. With the proposed routing approach, for a given set of pin vertices and a given set of on-chip functional modules, a rectilinear Steiner minimal tree connecting all the pin vertices while avoiding the blockage of functional modules can be constructed automatically. Furthermore, several heuristics including a divide-and-conquer strategy, a non-pin leaf node pruning strategy, a dynamic parameter strategy, etc., are integrated into the proposed algorithm to fundamentally improve the performance of the Physarum router. Simulation results on multiple benchmarks confirm that the proposed algorithm leads to shorter wirelength compared with several state-of-the-art methods.

Keywords Physarum polycephalum · Mathematical modeling · Rectilinear Steiner minimal tree · Obstacle-avoiding routing · Integrated circuits

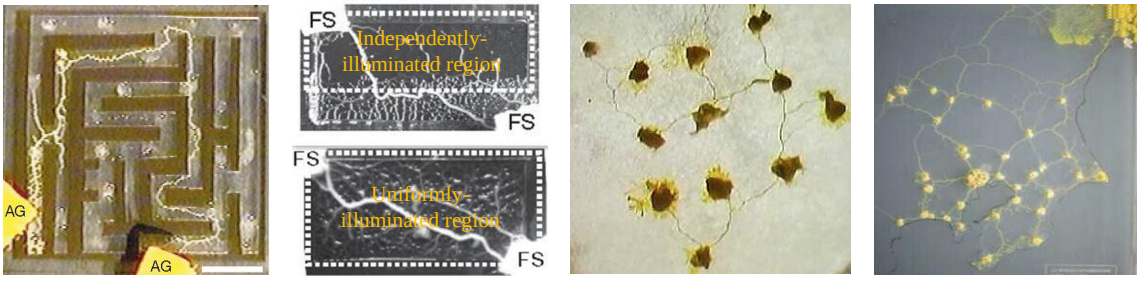
1 Introduction

Wire routing, also referred to simply as routing, plays an important role in the design automation of large-scale integrated circuits [1]. The construction of an interconnection system which seeks to connect a set of pin vertices on a silicon chip while ensuring the minimum total wirelength, is essential for each signal net [2].

Ever since the Hanan grid was proposed in 1966 [3], the rectilinear Steiner minimal tree (RSMT) problem has been studied extensively due to its practical implications [4]. RSMT is now being applied in many areas of electronic design automation. Particularly in the routing design of integrated circuits, it is usually used to construct an initial net topology connecting many pin vertices on the chip [5, 6]. Most prior work on circuit routing, however, assumes an obstacle-free routing plane. As the density of integrated circuits keeps increasing, more and more reusable components, such as macro blocks, IP cores, and pre-routed nets, are integrated into a single chip. These components cannot be run through during the routing process. As a result, the obstacle-avoiding RSMT (OARSMT) construction problem has become much more significant than ever before [7]. Furthermore, the RSMT construction

This work was supported in part by the National Basic Research Program of China under Grant 2011CB808000 and the National Natural Science Foundation of China under Grant 11501114 and 61672159.

✉ Xing Huang
E-mail: xinghuang@mx.nthu.edu.tw
1. College of Mathematics and Computer Sciences, Fuzhou University, Fuzhou, Fujian, China, 350116.
2. Key Laboratory of Network Computing and Intelligent Information Processing (Fujian Province), China, 350116.



(a) (b) (c) (d)

Fig. 1: Biological behaviors of Physarum. (a) Maze solving [18], (b) shortest-path construction with danger avoidance [19], (c) spanning tree construction [20], and (d) network construction using a simulated Tokyo rail network [21].

problem even without considering obstacles has been proved to be NP-complete [8], the presence of obstacles will further increase the complexity of routing design.

Over the past few years, a number of methodologies have been proposed to solve the OARSMT construction problem [9–13]. For example, in [9] an obstacle-avoiding routing algorithm called λ-OAT is proposed to construct a Steiner tree in the λ-geometry plane. In particular, an OARSMT can be generated efficiently when the value of λ is set to 2. In [10] an efficient four-step routing algorithm is proposed to construct an obstacle-avoiding rectilinear Steiner tree (OARST) for a given set of pins and obstacles. To reduce the total wirelength of the generated Steiner tree, an edge-based global refinement technique as well as a local refinement technique called "segment translation" are implemented as a postprocessing step, thus increasing the wirelength of shared routing paths. In [11] a fast four-step heuristic method is proposed to construct an obstacle-avoiding Steiner tree in both rectilinear and octilinear architectures [14]. The algorithm first constructs an obstacle-free Euclidean minimum spanning tree (MST) that connects the given pin vertices. Then an obstacle-avoiding Steiner tree is generated by transforming the edges in the MST into either rectilinear or octilinear paths based on two pre-computed lookup tables. Moreover, to avoid the blockage of obstacles, an efficient obstacle-avoiding strategy is proposed to introduce some additional vertices from the boundary of obstacles as intermediate nodes. Finally, the resulting obstacle-avoiding Steiner tree is refined to further reduce the total wirelength, thereby generating an obstacle-avoiding Steiner minimal tree. The aforementioned heuristics, however, suffer from the drawback of an unbalance between solving efficiency and routing quality, leading to either a low-quality solution or a relatively long computation time. Also, several exact routing algorithms are proposed to generate an optimal OARSMT [12, 13]. These methods, unfortunately, are not practical in industrial manufacturing due to their exponential time complexities.

On the other hand, learning the intelligent behaviors of bio-systems has inspired scientists to create a number of powerful computational methodologies over the past several decades [15–17]. In particular, the *Physarum polycephalum* (hereinafter referred to as Physarum) [18], an amoeboid single-cell organism, has been attracting high research interest in recent years due to its excellent capability in path finding and network construction. For example, in [18] Physarum forms a shortest path connecting two food sources distributed at two different exits on a labyrinth-shaped agar surface (Fig. 1(a)). In [19] the length of the path formed by Physarum in the region illuminated independently is reduced instinctively (Fig. 1(b)), showing a powerful danger-avoidance capability. In [20] Physarum forms a spanning tree connecting multiple food sources (Fig. 1(c)), which is very close to the one generated by the classical MST algorithms in terms of the number of edges and the total length. In [21] 36 food sources that represents the city locations of Tokyo area are used to verify the network construction capability of Physarum. Correspondingly, the resulting network is very similar to the real railway system in Tokyo in terms of cost, efficiency, and fault tolerance (Fig. 1(d)). Furthermore, in [22] an adaptive dynamic equation is proposed to simulate the biological mechanisms of Physarum, such as body shrinkage and signal transmission, leading to the first mathematical model of "Physarum computing" that is practical for complex engineering and

Table 1: List of abbreviations frequently used in this paper

Abbreviation	Description
RSMT	Rectilinear Steiner minimal tree
OARST	Obstacle-avoiding rectilinear Steiner tree
OARSMT	Obstacle-avoiding rectilinear Steiner minimal tree
SFCT	Steiner full connected tree
MST	Minimum spanning tree
PSO	Particle swarm optimization

optimization problems. Thereafter, a lot of researches based on Physarum computing have been carried out to address the routing problems in engineering and industry [23–28]. More importantly, Physarum computing is now being applied to various practical fields, including wireless sensor networks [29–32], transportation networks [33–35], network division [36, 37], supply chain network [38], Steiner tree problem [39, 40], etc.

The intelligent behaviors discussed above inspire us to further explore the optimization potential of Physarum, so that complex graph optimization problems can be solved in a computationally efficient manner. In addition, in view of the high complexity of the routing design of integrated circuits, it is indispensable to find a new method that can construct an OARSMT in a systematic manner. Accordingly, in this paper, we present *PORA*, a Physarum-inspired obstacle-avoiding routing algorithm. Table 1 lists the abbreviations frequently used in the paper. Our contributions are summarized as follows:

(1) To the best of our knowledge, this is the first work that applies Physarum computing to the routing design of integrated circuits. On the one hand, we further extend the application fields of Physarum optimization and prove its strong network construction capability. On the other hand, we present a novel Physarum-inspired OARSMT construction algorithm from the perspective of circuit routing. This work provides a basis for future research.

(2) We explore the optimization capability of Physarum systematically and present a powerful routing tool called Physarum router. Moreover, several heuristic strategies are integrated into the Physarum router to fundamentally improve its performance. Our work makes up for the shortcomings of prior work in the following aspects: 1) Physarum router can ensure the connectivity of the generated network when multiple paths with the same cost are connected to a common node in a complex graph and 2) Physarum router can ensure an exact termination time of Physarum computing.

(3) We present an effective divide-and-conquer strategy to guide the execution of Physarum router, so that the efficiency of the whole algorithm can be improved significantly. This strategy is based on several key models/techniques, including a pre-constructed Steiner full connected tree (SFCT, defined in Section 3.1), an obstacle-avoiding routing graph (OARG, defined in Section 3.2), and a multi-perspective evaluation mechanism.

(4) We adopt a nutrition absorption/consumption model to simulate the biological behaviors of Physarum. Moreover, a dynamic parameter strategy is presented to strengthen the search capability of Physarum. PORA can generate a high-quality OARSMT within a reasonable runtime. It is valuable for both scientific research and industrial production.

The rest of this paper is organized as follows. Section 2 formulates the OARSMT problem and presents the basic model of Physarum computing. The proposed algorithm is described in detail in Section 3. Section 4 presents experimental results, and the conclusions are drawn in Section 5.

2 Problem Formulation and Methodology

2.1 OARSMT problem

As discussed previously, as the feature size of integrated circuits keeps shrinking, many reusable components such as macro blocks and IP cores can now be integrated into a chip before performing the routing design. These components should therefore be viewed as obstacles and should not be run through during the routing process. Accordingly, in the OARSMT problem, an obstacle can geometrically be defined as a rectangle of any size [10].

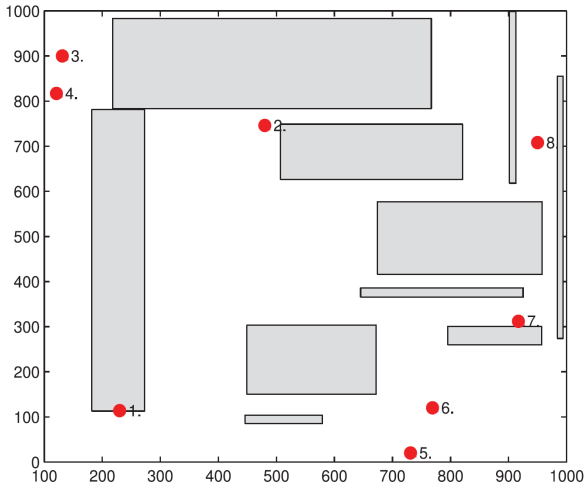


Fig. 2: The layout of a chip with 8 pin vertices and 10 obstacles [41].

As the layout shown in Fig. 2, the input of the OARSMT problem consists of a set of pin vertices $P = \{p_1, p_2, p_3 \dots p_m\}$ and a set of obstacles $O = \{o_1, o_2, o_3 \dots o_k\}$. Each pin $p_i \in P$ is associated with a coordinate position (x_i, y_i). Note that p_i cannot locate inside any obstacle, but it can be located on the boundary of an obstacle. An obstacle $o_j \in O$ is associated with two coordinates (x_{j1}, y_{j1}) and (x_{j2}, y_{j2}), which represent the left-lower point and the upper-right point of the obstacle, respectively. Any two obstacles cannot overlap with each other except at boundary points. Our goal is to construct an OARST connecting all the pin vertices. Inside such a tree, some additional vertices, i.e., Steiner points, may be introduced as internal nodes. Edges in the tree are either horizontal and vertical. Furthermore, an edge cannot intersect with any obstacle, but it can be line-touched on the boundary or point-touched at the corner of an obstacle. Accordingly, the OARSMT construction problem can be formulated as follows:

OARSMT problem: Given a set P of pin vertices and a set O of obstacles in the routing plane, construct a Steiner tree connecting all the given vertices in P, using only horizontal and vertical edges, such that no edge intersects with any obstacle in O and the total length of the tree is minimized.

2.2 The basic computation model of Physarum

The computation model used in PORA follows the biological mechanisms of Physarum [22]. An adaptive tubular network $\mathcal{G}$ is constructed to simulate the foraging behaviors of Physarum, where a vertex v_i in $\mathcal{G}$ represents a food source or an internal node and an edge $e_{i,j}$ in $\mathcal{G}$ represents a tubular path that can be used for protoplasmic-flow transportation between v_i and v_j. Moreover, each vertex and each path in the network are associated with a pressure value and a thickness value, respectively. Then the path-finding process of Physarum can be simulated by dynamically updating the tubular network according to the following rules:

- A tubular path that includes a non-food leaf node will be eliminated gradually, i.e., the thickness of the tubular path is decreased gradually during the path-finding process;
- A thick and short tubular path is more efficient for protoplasmic-flow transportation (hydrodynamic theory).

In addition, to simulate the body change of Physarum during foraging, a feedback mechanism between tube thickness and protoplasmic flow is formulated as follows:

- The thicker a tubular path is, the more flux of protoplasm flows through it;
- An increase in flux can further increase the thickness of a tubular path;
- The thickness of a tubular path is decreased in the potential dangerous regions.

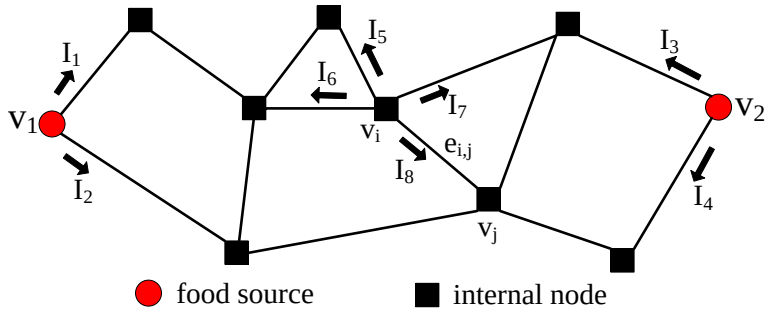


Fig. 3: The initial tubular network of a Physarum organism.

Fig. 3 shows a tubular network that represents the initial shape of a Physarum organism, where red circles and black squares represent food sources and internal nodes, respectively. Assuming that the pressure at a node v_i is pr_i and the fluid (i.e., protoplasm flow) along tubular paths is a Poiseuille flow, the flux through a path $e_{i,j}$ from v_i to v_j can be computed as

$$Q_{i,j} = \frac{\pi r_{i,j}^4}{8\tau} \cdot \frac{pr_i - pr_j}{l_{i,j}} = \frac{d_{i,j}}{l_{i,j}}(pr_i - pr_j) \tag{1}$$

where $l_{i,j}$ and $r_{i,j}$ are the length and radius of tubular path $e_{i,j}$, respectively. τ is the viscosity coefficient of fluid and $d_{i,j} = \pi r_{i,j}^4/8\tau$ is a measure of the conductivity of path $e_{i,j}$. Since the length of a tubular path is a constant, the evolution of the tubular network is mainly dominated by the conductivities of paths as well as the pressures at nodes.

At each iteration step of Physarum computing, a food source is selected as a source v_s to drive the protoplasm flow through the tubular network. Moreover, the other food source in the network is selected as a sink v_k to withdraw the protoplasm flow. Correspondingly, the following equations can be derived according to the law of Kirchhoff:

$$\sum_{e_{i,j}} Q_{i,j} - I_0 = 0, \ if \ \ v_i = v_s \tag{2}$$

$$\sum_{e_{i,j}} Q_{i,j} + I_0 = 0, \ if \ \ v_i = v_k \tag{3}$$

$$\sum_{e_{i,j}} Q_{i,j} = 0, \ if \ \ v_i \neq v_s \& v_k \tag{4}$$

where I_0 represents the total flux through the tubular network, i.e., the flux flowing from/into the source/sink node.

Taking the tubular network shown in Fig. 3 as an example. Assuming that v_1 and v_2 are selected as the source and the sink, respectively, we have $I_1 + I_2 = I_0$, $I_3 + I_4 = -I_0$, and $I_5 + I_6 + I_7 + I_8 = 0$. Accordingly, the Poisson equation of node pressures can be derived from (2)–(4):

$$\sum_{e_{i,j}} \frac{d_{i,j}}{l_{i,j}} (pr_i - pr_j) = \begin{cases} I_0, & \text{if } v_i = v_s \\ 0, & \text{if } v_i \neq v_s \& v_k \\ -I_0, & \text{if } v_i = v_k \end{cases} \tag{5}$$

By setting $pr_k = 0$ as the basic pressure level, the pressures at other nodes can be determined by solving the Poisson equation (5), and the flux through each tubular path can be computed by (1).

Furthermore, as discussed previously, besides the pressures at nodes, the self-adaptive behavior of the tubular network is also closely related to the thickness of tubular paths, which, correspondingly, can be described as the feedback mechanism between tube conductivities and the flux through the tubes:

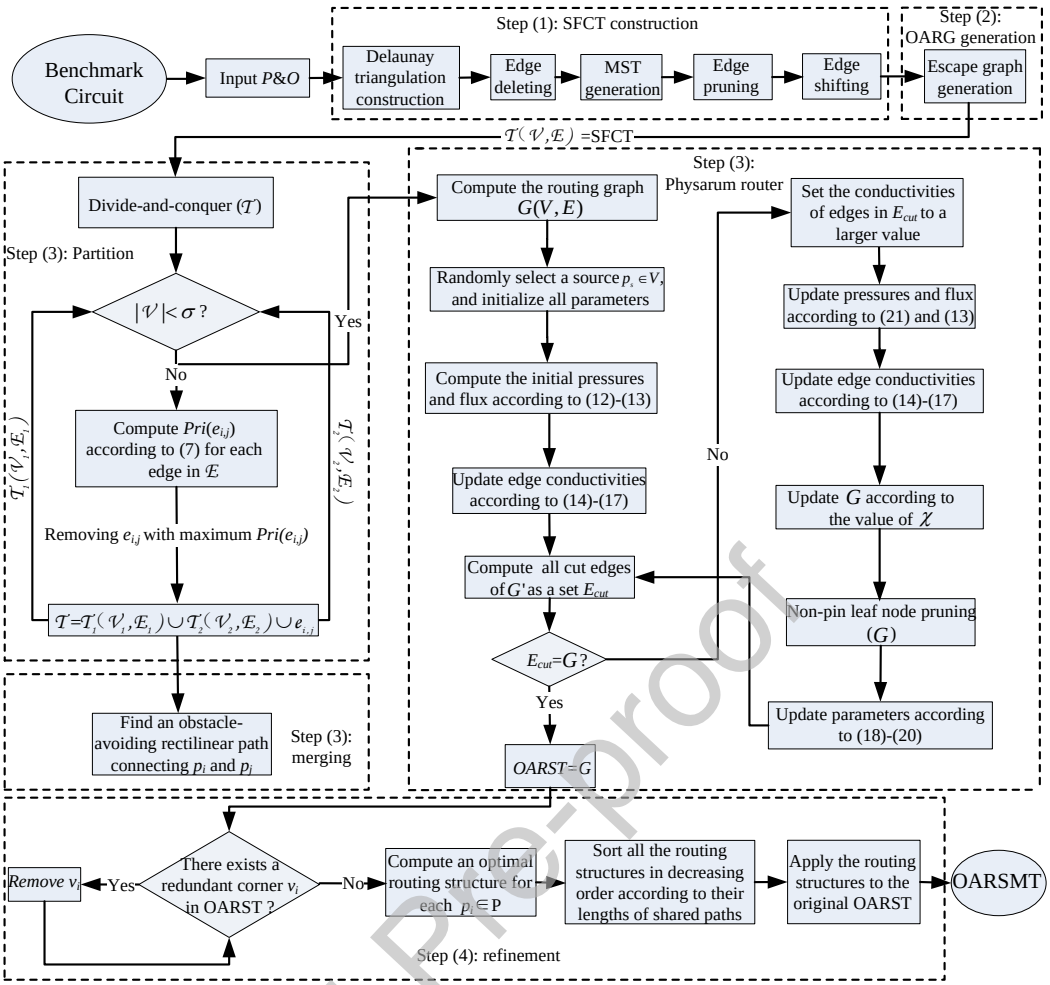


Fig. 4: Flow chart of PORA.

$$\frac{d}{dt}d_{i,j} = f\left(|Q_{i,j}|\right) - \delta d_{i,j} \tag{6}$$

where function $f(|Q_{i,j}|)$ describes the expansion of tubes in response to flux, which is usually formulated as a continuous function with the property of monotone increasing, while satisfying $f(0) = 0$. For example, function $f(\cdot)$ can be defined as $f(|Q_{i,j}|) = 2 \times |Q_{i,j}|$. The second term on the right side of (6) corresponds to the contraction of tubes. Parameter δ is a positive real number that indicates the decrease rate of tube conductivities in response to the dangers. In particular, tubes that without fluids flow through will be eliminated gradually in the network.

With the mechanisms above, tubular paths in the network will compete with each other for the limited amount of fluid, so that protoplasmic flow in those non-critical tubular paths disappears gradually. After a certain number of iterations, only a shortest path that connects the two food sources is remained in the tubular network.

3 Algorithm Design

Based on the discussion above, in this section, we present PORA, a Physarum-inspired obstacle-avoiding routing algorithm for OARSMT construction. Fig. 4 shows the flow chart of PORA, which consists of four major steps:

(1) **SFCT construction**. An SFCT connecting the given pin vertices and some corner vertices is constructed to guide the partition of routing plane.

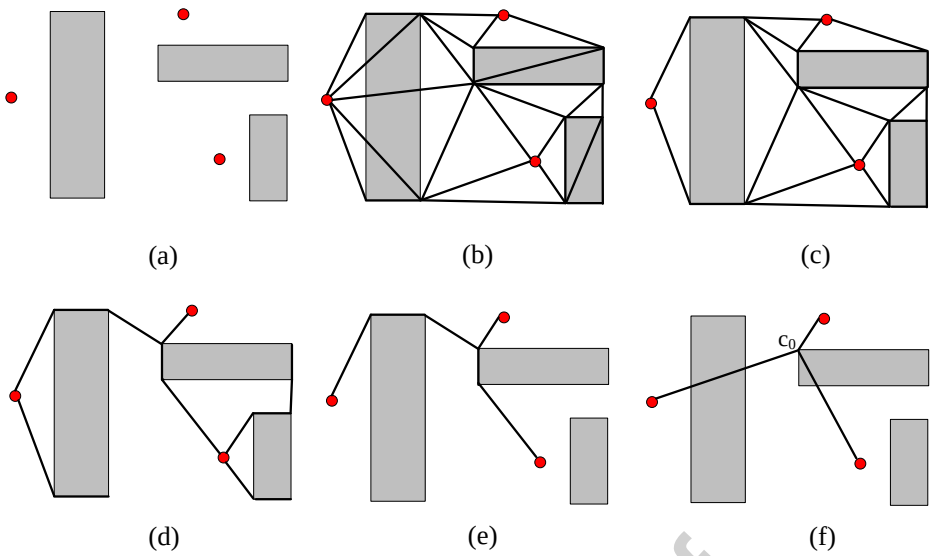


Fig. 5: Illustration of SFCT construction. (a) Input P&O, (b) Delaunay triangulation, (c) Delaunay triangulation after edge deleting, (d) MST, (e) MST after edge pruning, and (f) SFCT.

(2) **OARG generation**. An OARG connecting all the pin vertices and obstacles is constructed as the underlying graph.

(3) **OARST generation**. A Physarum router guided by an efficient divide-and-conquer strategy is presented to construct an OARST on the previously generated OARG.

(4) **Refinement**. Two refinement techniques are presented to further reduce the total wirelength of the constructed OARST.

3.1 SFCT Construction

Given a connected, weighted, and undirected graph, an MST is a subset of the edges that connects all the vertices together, without any cycles and with the minimum total edge weight [42]. Since MSTs are much easier to construct than Steiner trees, most previous studies on the routing design of integrated circuits usually construct an MST as the basic skeleton of the routing network first [6, 10, 11, 41], and then transform it to a Steiner tree. For example, In [10] a minimum terminal spanning tree is first constructed to connect all the pin vertices, and it is then transformed into an OARST by performing an edge-based refitment technique. In [41] a particle swarm optimization (PSO)-based method is proposed to construct an obstacle-avoiding Steiner minimal tree. In this algorithm, each individual in the particle population is initialized to an MST connecting the given pin vertices. The generated MSTs are then transformed to Steiner trees by introducing some additional points in the routing plane. Correspondingly, in the proposed method, we first construct an SFCT to connect the given pin vertices and some corner vertices, which can be defined as follows:

Definition 1 Given a set of pin vertices and a set of obstacles on the routing plane, a tree T is called a ***Steiner full connected tree*** if the followings are true [9]: 1) T connects all the pin vertices through some corner vertices, 2) all the leaf nodes in T are pin vertices, and 3) all the corner vertices in T are Steiner points.

As illustrated in Fig. 4, SFCT plays the following roles in PORA: 1) guiding the divide-and-conquer strategy to efficiently partition the routing plane and 2) determining the basic skeleton of the final OARSMT. Accordingly, in the proposed method, the construction of an SFCT consists of five steps, including Delaunay triangulation construction [43], edge deleting, MST generation, edge pruning, and edge shifting.

We take the layout shown in Fig. 5(a) as an example to illustrate the procedure above. Since a Delaunay triangulation has been proved to contain at least one MST for a set of

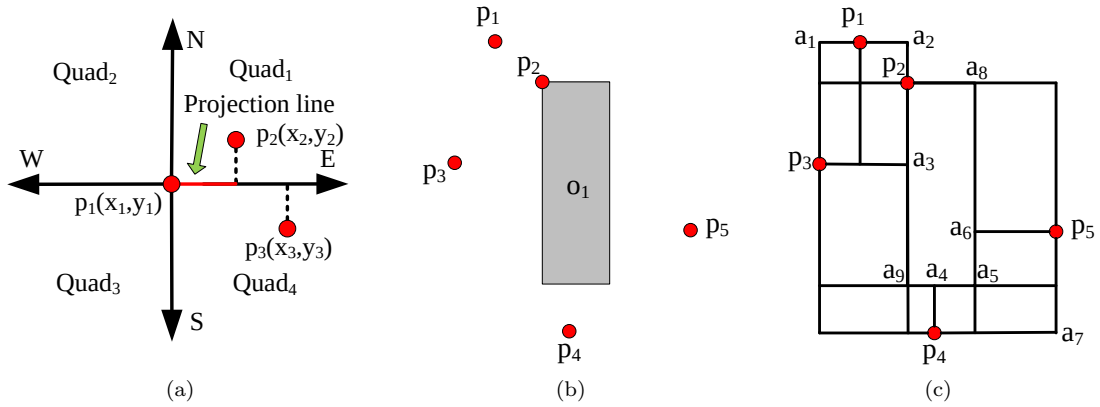


Fig. 6: Illustration of OARG construction. (a) Quadrant partition of a vertex, (b) a part of the layout of a circuit, and (c) the constructed OARG corresponding to (b).

points, and the candidate edges is confined in only linear space [9], we first construct a Delaunay triangulation connecting all the pin vertices and corner vertices (Fig. 5(b)) [43], and then delete the edges that run through obstacles (Fig. 5(c)). Next, the classical MST algorithms, e.g., Prim algorithm or Kruskal's algorithm, can be employed to generate an MST in linear time (Fig. 5(d)). Moreover, during the of edge pruning, all the non-pin leaf nodes as well as the edges connecting to them are removed from the tree (Fig. 5(e)). Finally, an SFCT can be generated by removing all the non-Steiner corner vertices (Fig. 5(e)). To ensure the connectivity of the tree, once a corner vertex is removed, the corresponding two adjacent nodes will be connected. Note that c_0 in Fig. 5(e) is not removed from the tree since it is a Steiner point.

3.2 OARG Generation

In this step, an OARG is constructed as the underlying graph of the proposed routing method, which can be defined as follows:

Definition 2 Given a set of pin vertices and a set of obstacles, an undirected graph connecting all the pin vertices and corner vertices is called an ***obstacle-avoiding routing graph*** if none of its edges intersects with obstacles [10].

In the proposed method, as shown in Fig. 6(a), we divide the routing plane into four quadrants with respect to each vertex, and adopt a line projection technique to generate an OARG called escape graph [44]. More specifically, we project line segments from all the pin vertices and corner vertices at four orthogonal directions, and each line is ended when encountering a component or the boundary of the routing plane. All the intersections of projection lines are points in the escape graph, and the lines among points are edges. The time complexity of escape graph construction is $O(n^2)$, where n is the total number of pin vertices and corner vertices. More importantly, it has been proved that escape graph contains at least one optimal OARSMT and the candidate edges for OARSMT construction is only $O(n^2)$.

On the other hand, to further improve the efficiency of OARG construction, two heuristic strategies are adopted to eliminate the redundant points and edges in the constructed escape graph [45]: 1) Line segments are projected only from pin vertices at the early stage, and then from corner vertices of obstacles that have blocked the previous projection lines and 2) the length of projection lines at each orthogonal direction is limited by the shortest distance between the source and the pin vertices in the two neighboring quadrants. For example, in Fig. 6(a), the projection line of source p_1 towards east is limited on the interval from x_1 to x_2 (see red line) since $x_2 - x_1 < x_3 - x_1$.

The strategies above bring the following benefits to the OARG construction: 1) the obstacles that have no effect on the optimality of OARG are ignored and 2) the points and edges

ALGORITHM 1: Divide-and-conquer ($T(\mathcal{V},\mathcal{E})$)

Input: An SFCT.
Output: An OARST.
Initialize the values of α, β, γ, σ, S_1, and S_2;
if $|\mathcal{V}| \leq \sigma$ **then**
  Physarum router($\mathcal{V}$);
  Return;
end
else
  for *each edge* $e_{i,j} \in \mathcal{E}$ **do**
    Compute the priority value of $e_{i,j}$ according to (7);
  end
  Generate $T_1(\mathcal{V}_1,\mathcal{E}_1)$ and $T_2(\mathcal{V}_2,\mathcal{E}_2)$ by breaking T at the edge with maximum priority value;
  Divide-and-conquer($T_1(\mathcal{V}_1,\mathcal{E}_1)$);
  Divide-and-conquer($T_2(\mathcal{V}_2,\mathcal{E}_2)$);
  Merge the OARSTs of T_1 and T_2 using the fast obstacle-avoiding strategy in [11];
end

that are useless for OARSMT construction are removed from the OARG directly. As an example, Fig. 6(b) and 6(c) shows a part of a circuit layout and the corresponding OARG, respectively.

3.3 OARST generation

After completing the construction of OARG, in this step, we formulate the mathematical model of Physarum computing based on its foraging behaviors, thereby presenting the aforementioned Physarum router, which can construct an OARST connecting the given pin vertices on the constructed OARG. Moreover, a divide-and-conquer strategy is proposed to efficiently partition the routing plane, so that the performance of Physarum router can be improved systematically.

3.3.1 Divide-and-conquer strategy

Algorithm 1 shows the pseudocode of the proposed divide-and-conquer strategy. Our algorithm divides all the pin vertices in P into a number of subsets by recursively removing edges from the previously generated SFCT, and constructs an OARST for each of them using the proposed Physarum router. These sub-OARSTs are then merged together to form a complete OARST connecting all the pin vertices.

In the beginning, two subtrees are generated after removing an edge from the SFCT. If the number of vertices in a subtree is less than a given threshold σ, an OARST conneting these vertices will be constructed directly. Otherwise, the resulting subtree will be further divided into smaller trees. To this end, we propose a multi-perspective evaluation mechanism to select edges from the SFCT, so that the efficiency of the dividing procedure can be improved systematically. Before discussing our method in more detail, we first give several concepts used in the proposed evaluation mechanism.

Definition 3 Given a set V of vertices, the ***rectilinear bounding box*** of V, denoted by $RBB(V)$, is the minimum rectangle that covers all the vertices in V [11]. $RBB(V)$ is associated with two coordinates $(R_l(V), R_b(V))$ and $(R_r(V), R_t(V))$, which are the positions of its left-lower point and right-top point, respectively.

Definition 4 Let $V = \{v_1, v_2 \dots v_h\}$ be a set of vertices, c_r be the barycentric coordinate of V, and $dis(v_i, c_r)$ be the Euclidean distance between v_i and c_r, we have the following definitions: 1) a vertex $v_i \in V$ is referred to as an ***off-group point*** if $dis(v_i, c_r) > \sum_{j=1}^{h} dis(v_j, c_r)/h$ and 2) the ***aggregation degree*** of V can be computed as $1 - h_o/h$, where h_o is the number of off-group points in V.

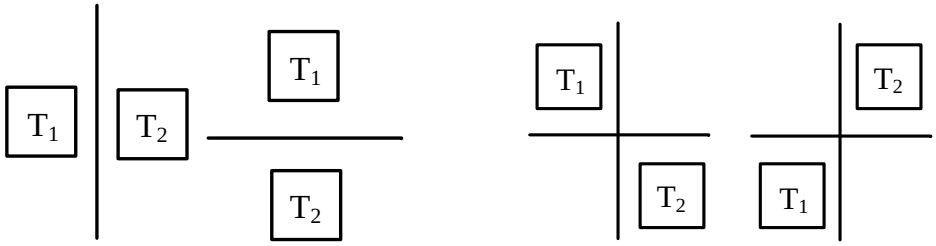


Fig. 7: Relative positions between T_1 and T_2 after removing an edge from T. (a) Left-right and bottom-up layouts and (b) diagonal layout.

With the concepts defined above, given a tree $T(\mathcal{V}, \mathcal{E})$ that needs to be divided into two subtrees $T_1(\mathcal{V}_1, \mathcal{E}_1)$ and $T_2(\mathcal{V}_2, \mathcal{E}_2)$ (Initially, T = SFCT), we compute a priority value for each edge in T and break the tree at the edge with maximum priority value. The priority value of an edge $e_{i,j} \in \mathcal{E}$ can be computed as

$$\mathrm{Pri}(e_{i,j}) = \frac{\alpha \cdot A_1(i,j) + \beta \cdot A_2(i,j) + \gamma \cdot A_3(i,j)}{A_4(i,j)} \tag{7}$$

where α, β, and γ are three weight coefficients and $A_1(i,j)$–$A_4(i,j)$ are the evaluation functions designed from different perspectives. These functions are formulated as follows.

• $A_1(i,j)$ is used to evaluate the relative positions between T_1 and T_2 when breaking T at edge $e_{i,j}$, including the following scenarios as illustrated in Fig. 7:

– Left-right layout: $R_r(\mathcal{V}_1) < R_l(\mathcal{V}_2)$;
– Bottom-up layout: $R_b(\mathcal{V}_1) > R_t(\mathcal{V}_2)$;
– Diagonal layout: $R_r(\mathcal{V}_1) < R_l(\mathcal{V}_2) \wedge R_b(\mathcal{V}_1) > R_t(\mathcal{V}_2)$ or $R_r(\mathcal{V}_1) < r_l(\mathcal{V}_2) \wedge R_t(\mathcal{V}_1) < R_b(\mathcal{V}_2)$.

The value of $A_1(i,j)$ is determined according to the following rules : 1) When T_1 and T_2 form a diagonal layout, an optimal RSMT connecting all the vertices in $\mathcal{V}$ can always be constructed by merging the optimal RSMTs of two subtrees [5]. In other words, breaking T at edge $e_{i,j}$ can still maintain the optimality of the solution, the result of $A_1(i,j)$ is therefore set to a relatively large value and 2) when T_1 and T_2 form a bottom-up or a left-right layout, implying that the optimal RSMT connecting the vertices in $\mathcal{V}$ may not be found if T is broken at edge $e_{i,j}$, the result of $A_1(i,j)$ is therefore set to a relatively small value. Accordingly, in the proposed method, $A_1(i,j)$ is computed as

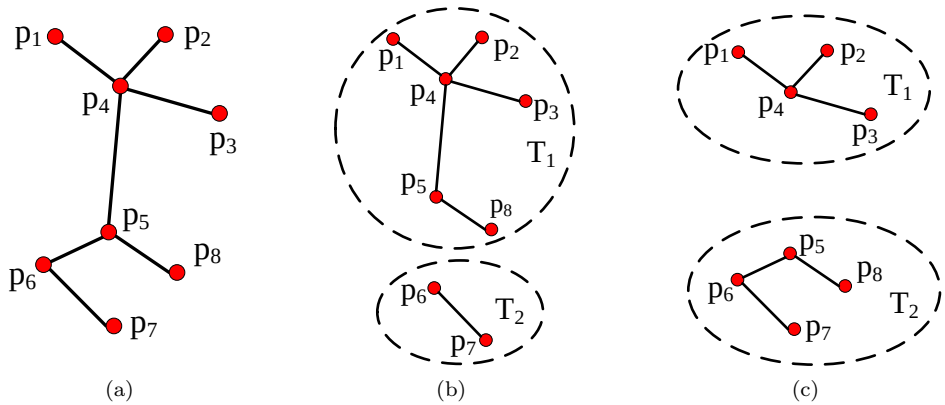
$$A_1(i,j) = \begin{cases} 10, & \text{if } T_1 \text{ and } T_2 \text{ form a dignoal layout} \\ 5, & \text{if } T_1 \text{ and } T_2 \text{ form a bottom-up/left-right layout} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

• $A_2(i,j)$ is used to evaluate the sizes of T_1 and T_2 when breaking T at edge $e_{i,j}$, i.e., the number of vertices contained in each subtree. Since a balanced division of the vertices in T is conducive to improving the performance of divide-and-conquer method, $A_2(i,j)$ is formulated as

$$A_2(i,j) = \frac{S_1}{||\mathcal{V}_1| - |\mathcal{V}_2|| + 1} \tag{9}$$

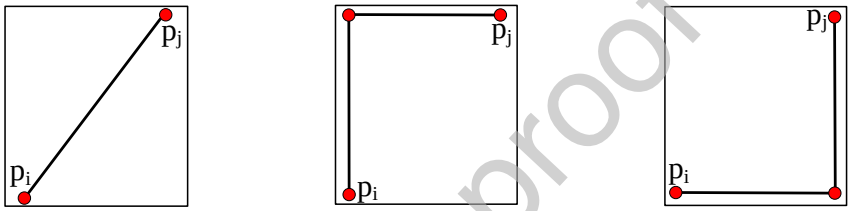
where S_1 is a constant and $|\mathcal{V}_1|$ and $|\mathcal{V}_2|$ are the numbers of vertices in T_1 and T_2, respectively.

• $A_3(i,j)$ is used to evaluate the aggregation degree of vertices in T_1 and T_2 when breaking T at edge $e_{i,j}$. When a set of vertices is widely scattered over the routing plane, i.e., the aggregation degree of these vertices is relatively low, zig-zag rectilinear paths have to be constructed between vertices due to the blockage of obstacles. Consequently, a large number of Steiner points have to be selected and added to the constructed OARST, thereby increasing the complexity of the problem. In the proposed method, our goal is find a division solution



(a) (b) (c)

Fig. 8: An example of tree breaking. (a) A tree T, (b) breaking T at edge $e_{5,6}$, and (c) braking T at $e_{4,5}$.



(a) (b) (c)

Fig. 9: L-paths between vertices. (a) An edge $e_{i,j}$ and (b)(c) two L-paths corresponding to $e_{i,j}$.

such that the aggregation degrees of vertices in the resulting subtrees can be improved as much as possible. For example, when dividing the tree shown in Fig. 8(a) into two subtrees, Fig. 8(b) and 8(c) gives two different dividing solutions by breaking the tree at edges $e_{4,5}$ and $e_{5,6}$, respectively. The result in Fig. 8(b), however, is less than satisfactory since p_5 and p_8 may become two off-group points, leading to a low aggregation degree of the vertices in the subtree T_1. In contrast, in Fig. 8(c), the aggregation degrees of the vertices in both subtrees are relatively high. Accordingly, $A_3(i,j)$ is formulated as

$$A_3(i,j) = \frac{S_2}{N_1+1} + \frac{S_2}{N_2+1} \tag{10}$$

where S_2 is a constant and N_1 and N_2 are the numbers of off-group points in T_1 and T_2, respectively.

- $A_4(i,j)$ is used to evaluate the complexity of the obstacle-avoiding rectilinear path between vertices p_i and p_j. As shown in Fig. 9, for an edge $e_{i,j}$, there are two types of L-paths connecting p_i and p_j. When constructing the rectilinear routing path between p_i and p_j, if both L-paths are blocked by obstacles, it is inevitable to find some intermediate nodes/Steiner points, which, consequently, increases the complexity of pathfinding. Accordingly, our algorithm tends to break a tree at an edge whose L-paths are blocked by obstacles, and then merge the resulting OARSTs using the fast obstacle-avoiding strategy in [11]. Based on the analysis above, $A_4(i,j)$ is computed as

$$A_4(i,j) = \begin{cases} 1, & \text{if both L-paths of } e_{i,j} \text{ are blocked by obstacles} \\ 1.5, & \text{otherwise} \end{cases} \tag{11}$$

3.3.2 Physarum router

As discussed previously, after dividing the given pin vertices into a number of subsets, Physarum router will be invoked to construct an OARST for the vertices in each subset (see

ALGORITHM 2: Physarum router ($\mathcal{V}_s$)

Input: A subset $\mathcal{V}_s$ of vertices in SFCT.
Output: An OARST connecting all the vertices in $\mathcal{V}_s$.
Compute the routing graph $G(V, E)$ of $\mathcal{V}_s$;
Randomly select a vertex $p_s \in \mathcal{V}_s$ as the source node;
Initialize $d_{i,j}(0)$ to a small value for the edges in E;
Initialize $pr_i(0)$ to 0 for the vertices in $\mathcal{V}_s - p_s$;
Initialize the values of parameters η, ξ, and $\chi(1)$;
Compute $pr_i(0)$ for each vertex $v_i \in V$ according to (12);
Compute $Q_{i,j}(0)$ for each edge $e_{i,j} \in E$ according to (13);
Compute $d_{i,j}(1)$ for each edge $e_{i,j} \in E$ according to (14)–(17);
Initialize $E_{cut} = \emptyset$ and $t = 1$;
while $E_{cut} \neq E'$ **do**
  Set the conductivities of the edges in E_{cut} to a large value;
  Update $pr_i(t)$ for each vertex $v_i \in V$ according to (21);
  Update $Q_{i,j}(t)$ for each edge $e_{i,j} \in E$ according to (12);
  Compute $d_{i,j}(t+1)$ for each edge $e_{i,j} \in E$ according to (14)–(17);
  Update G according to the value of $\chi(t)$;
  Perform the non-pin leaf node pruning for G;
  Update E_{cut} based on the current routing graph G;
  Update parameters η, ξ, and χ according to (18)–(20);
  t=t+1;
end

Algorithm 2). The construction of OARSTs is based on the previously generated OARG. For a subset $\mathcal{V}_s$ of vertices in SFCT, however, it is not necessary to search the entire OARG during the OARST construction. Physarum router only needs to consider a subgraph of OARG that contains at least one optimal OARST connecting the vertices in $\mathcal{V}_s$. We take the layout shown in Fig. 6(c) as an example. Assuming that we need to find an OARST connecting vertices p_1, p_2, and p_3. Then Physarum router only needs to search the region covered by the rectangle $a_1 - a_2 - a_3 - p_3$, i.e., the rectilinear bounding box formed by the three pin vertices. Note that if the rectilinear bounding box of vertices is completely blocked by obstacles, the regions that covered by these obstacles should also be considered. For example, in Fig. 6(c), since the rectilinear bounding box formed by p_3 and p_5 are blocked by obstacle o_1, the region covered by o_1 in OARG should be searched accordingly when constructing an OARST connecting the two vertices (in this case, the OARST is simplified to a shortest obstacle-avoiding rectilinear path between p_3 and p_5). Based on the analysis above, we have the following definition:

Definition 5 Given a subset $\mathcal{V}_s$ of vertices in SFCT, the ***routing graph*** of $\mathcal{V}_s$, denoted by $G(V, E)$, can be generated by the following steps: 1) G is initialized to the subgraph obtained by mapping $RBB(\mathcal{V}_s)$ onto OARG and 2) if $RBB(\mathcal{V}_s)$ is completely blocked by obstacles, G is expanded to include the regions covered by these obstacles in OARG.

With the routing graph $G(V, E)$ of a set of vertices $\mathcal{V}_s$, in the proposed Physarum router, we use $\mathrm{pr}_i(t)$ and $d_{i,j}(t)$ to represent the pressure at a vertex $v_i \in V$ and the conductivity of an edge $e_{i,j} \in E$ at the t-th iteration, respectively. Following the basic model of Physarum discussed in Section 2.2, we randomly select a pin vertex, denoted by p_s, from $\mathcal{V}_s$ as the source node, and let the remaining vertices in $\mathcal{V}_s$ be sink nodes. Moreover, the pressures at sink nodes are set to 0 as the basic pressure level and the conductivities of edges in E are set to a small value. Assuming that the flux driven by the source node is $(|\mathcal{V}_s| - 1) * I_0$, the fluid flows into each sink is I_0, and the length of each edge $e_{i,j} \in E$ is $len(i, j)$, where $|\mathcal{V}_s|$ is the number of vertices in $\mathcal{V}_s$, the Poisson equation corresponding to the region covered by G at the t-th iteration can be expressed as

$$\sum_{e_{i,j}} \frac{d_{i,j}(t)}{len(i,j)} \times (pr_i(t) - pr_j(t)) = \begin{cases} (|\mathcal{V}_s| - 1) \times I_0, & \text{if } v_i = p_s \\ -I_0, & \text{if } v_i \in \mathcal{V}_s - p_s \\ 0, & \text{if } v_i \in V - \mathcal{V}_s \end{cases} \tag{12}$$

By solving the equation system above, the initial pressure at each vertex $v_i \in V$ can be obtained. Correspondingly, the flux through each edge $e_{i,j} \in E$ at the t-th iteration, denoted by $Q_{i,j}(t)$, can be computed as

$$Q_{i,j}(t) = \frac{d_{i,j}(t)}{len(i,j)} \times (pr_i(t) - pr_j(t)) \tag{13}$$

In addition, we employ a nutrition absorption/consumption model to simulate the morphological changes of Physarum. The status of each edge $e_{i,j} \in E$ is updated according the following rules: 1) the fluid flows through $e_{i,j}$ provides nutrition for the edge, 2) $e_{i,j}$ performs fluid transportation by consuming the nutrition contained in the edge, and 3) the change of the nutrition contained in $e_{i,j}$ can further affects the conductivity of the edge. Accordingly, the nutrition provided by the flux through $e_{i,j}$ at the t-th iteration, denoted by $N^+_{i,j}(t)$, can be computed as

$$N^+_{i,j}(t) = \eta \times \frac{|Q_{i,j}(t)|}{|Q_{i,j}(t)| + 1} \tag{14}$$

where η represents the absorption factor of edges in E and $|Q_{i,j}(t)|$ is the absolute value of the flux through $e_{i,j}$ at the t-th iteration. Moreover, the nutrition consumed by edge $e_{i,j}$ at the t-th iteration, denoted by $N^-_{i,j}(t)$, is computed as

$$N^-_{i,j}(t) = \xi \times len(i,j) \times d_{i,j}(t) \tag{15}$$

where ξ is the consumption factor of edges. Correspondingly, the change of conductivity in response to the flux through edge $e_{i,j}$ at the t-th iteration can be computed as

$$\frac{d}{dt} d_{i,j}(t) = N^+_{i,j}(t) - N^-_{i,j}(t) \tag{16}$$

Finally, the conductivity of edge $e_{i,j}$ at the $t+1$-th iteration can be computed as

$$d_{i,j}(t+1) = d_{i,j}(t) + \frac{\mathrm{d}}{dt} d_{i,j}(t) \tag{17}$$

On the other hand, in the mathematical model of Physarum router above, the differences of edge conductivities are usually very small at the early stages of iteration, but the differences of fluxes through edges are relatively large. As a result, it is nutrition absorption (i.e., global search) that dominates the morphological evolution of Physarum. Accordingly, the searching efficiency of Physarum router can be strengthened by setting η in (14) to a relatively large value and ξ in (15) to a relatively small value. In contrary, the differences of edge conductivities gradually increases with the increasing of iterations, and it is nutrition consumption (i.e., local search) that dominates the morphological evolution of Physarum at the later stages of iteration. Correspondingly, a relatively large ξ and a small η will enhance the local search of Physarum router. Based on the analysis above, we propose a dynamic strategy to update parameters η and ξ, which is formulated as

$$\eta = \eta_u - \frac{\eta_u - \eta_l}{sp} * t \tag{18}$$

$$\xi = \xi_l + \frac{\xi_u - \xi_l}{sp} * t \tag{19}$$

where η_u (ξ_u) and η_l (ξ_l) are the upper bound and the lower bound of parameter η (ξ), respectively, sp is a predefined interval distance, and t represents the iteration times. Note that in the first iteration η and ξ are initialized to η_u and ξ_l, respectively.

After computing the conductivities of all the edges at the $t+1$-th iteration, the edges that satisfy the condition of $d_{i,j}(t+1) < \chi(t+1)$ are removed from G, where $\chi(t+1)$ is the threshold of edge conductivity at the $t+1$-th iteration. Then, a new Poisson equation can be formulated by randomly selecting another source node from $\mathcal{V}_s$, and through which pressures at vertices and fluxes through edges can be updated accordingly. Physarum router updates the routing graph iteratively until an OARST connecting the vertices in $\mathcal{V}_s$ is generated. Similarly, since the conductivities of edges gradually increases with the increasing of iterations, to accelerate the convergence of the proposed algorithm, parameter χ is updated dynamically as

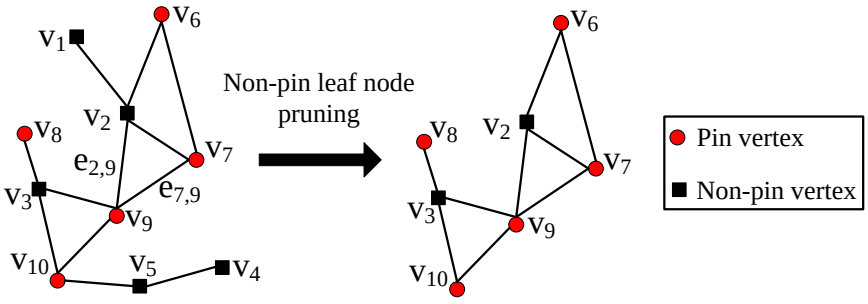


Fig. 10: An example of non-pin leaf node pruning.

$$\chi(t+1) = \chi(t) + \Delta c \tag{20}$$

where Δc is a user-defined constant.

3.3.3 Acceleration methods

In this subsection, two acceleration methods, including a non-pin leaf node pruning and a Poisson approximation method, are integrated into the proposed Physarum router, so that the searching efficiency of the algorithm can be further improved.

(1) *Non-pin leaf node pruning*: During the OARST construction, after removing the edges whose conductivities are smaller than the corresponding threshold at each iteration, some non-pin leaf nodes may be introduced to the routing graph, leading to some dead-end paths that cannot reach any pin vertex. If these redundant nodes as well as the edges connecting to them can be removed from the routing graph directly, the efficiency of Physarum router can be improved accordingly. To this end, we compute the degrees of vertices in the routing graph after each iteration, and then remove the non-pin vertices with degree one as well as the edges connecting to these vertices. Note that new non-pin leaf nodes may be generated after removing nodes from the graph. Accordingly, our algorithm performs the node-removing operation iteratively until all the remaining leaf nodes are pin vertices.

We take the graph shown in Fig. 10 as an example to illustrate the pruning method above. For the sake of simplicity, in this example, we directly use straight lines instead of rectilinear edges in the graph. It can be seen that nodes v_1, v_4 and v_5 as well as the edges connecting to them are removed from the graph. Note that v_5 is a new non-pin leaf node introduced by removing v_4 from the graph.

(2) *Poisson approximation*: On the other hand, simulation results in [31, 39] have shown that the variation of pressure at each vertex is continuous and relatively small between two successive iterations. Furthermore, Physarum router cares more about the overall trend of evolution and the structure of final OARST rather than the intermediate states of routing graph. Accordingly, we adopt an approximation scheme [31, 39] that substitute $pr_j(t) = pr_j(t-1)$ into (12) to compute the pressures at vertices as

$$\begin{cases} \sum\limits_{e_{i,j}} \frac{d_{i,j}(t) \times (pr_i(t) - pr_j(t-1))}{len(i,j)} = (|\mathcal{V}_s| - 1) \times I_0, & \text{if } v_i = p_s \\ pr_i(t) = 0, & \text{if } v_i \in \mathcal{V}_s - p_s \\ \sum\limits_{e_{i,j}} \frac{d_{i,j}(t) \times (pr_i(t) - pr_j(t-1))}{len(i,j)} = 0, & \text{if } v_i \in V - \mathcal{V}_s \end{cases} \tag{21}$$

In contrast to (12) which is of quadratic complexity, (21) can be solved in linear time, thereby improving the efficiency of Physarum router significantly.

3.3.4 Termination condition of Physarum router

Another issue that needs to be considered during the OARST construction is to determine a certain termination condition for Physarum router. This problem is very important for

the bio-inspired algorithms, since additional iterations not only reduce the efficiency of the algorithm, but also have no significant effect on solution quality. Most prior work, however, still uses a predefined maximum number of iteration times to stop the execution of the algorithm [6,41], which, consequently, may lead to a decrease in solution quality or an increase in runtime overhead.

To solve the problem above, so that the execution of Physarum router can be controlled in an accurate manner, we compute a set E_{cut} of cut edges in the routing graph after each iteration during the OARST construction. Physarum router can benefit from E_{cut} in two aspects:

- Since the objective of the proposed algorithm is to construct a routing tee, this implies that all the edges remained in the routing graph should be cut edges once an OARST is generated. Accordingly, the execution of Physarum router can be terminated once the condition of $E_{cut} = E_r$ is satisfied, where E_r is the set of edges remained in the routing graph after performing the edge-removing operation;
- Since any cut edge cannot be removed from the routing graph during the OARST construction, otherwise the routing graph will be divided into disconnected subgraphs. The conductivities of these cut edges can thus be set to a relatively large value, thereby accelerating the convergence of the algorithm.

Moreover, we observe from a series of simulations that an invalid solution may be generated when several edges with the same cost are connected to a common node in the routing graph. This problem has so far been ignored in most prior work. Let us revisit the routing graph described in Fig. 10. Assuming that edges $e_{2,9}$ and $e_{7,9}$ in the graph have the same length, the conductivities of the two edges tend to be the same value with the increasing of iterations. Consequently, the two edges may be removed from the graph simultaneously if their conductivities are less than the threshold, thereby resulting in a disconnected graph. To solve this problem, we stipulate that the degree of each vertex in the routing graph can be decreased by one at most in each iteration. In this way, when removing edges connecting to the same node and with same conductivity from the graph, at least one edge will be added to the set E_{cut}, thus ensuring the connectivity of the routing graph.

3.4 Refinement

Since the generated OARST may still include some suboptimal routing paths, in this step, two refinement techniques are implemented to further optimize the structure of OARST, so that the total wirelength can be further reduced.

3.4.1 Redundant bends elimination

Since the constructed OARG usually contains multiple rectilinear routing paths between each pair of pin vertices, this property potentially provides Physarum router with more routing choices. On the other hand, however, some redundant bends may be introduced to the OARST at the same time, because Physarum router selects routing paths between vertices in a random manner. Accordingly, we present an edge-merging technique to eliminate these redundant points. For each non-pin vertex v_i with degree 2 in the constructed OARST, assuming that v_j and v_k are the adjacent nodes of v_i, if there exists an L-path between v_j and v_k that does not intersect with any obstacle (see Fig. 9, there are two feasible L-paths between each pair of vertices), and the corresponding wirelength is not longer than that of the routing path $v_j \rightarrow v_i \rightarrow v_k$, we remove v_i from the OARST and connect v_j to v_k directly. The procedure above are performed repeatedly until all the bends in the OARST are non-redundant.

3.4.2 Steiner-point relocation

Moreover, we present another technique called Steiner-point relocation to transform those suboptimal structures in the OARST into the optimal ones. For example, Fig. 11(a) shows

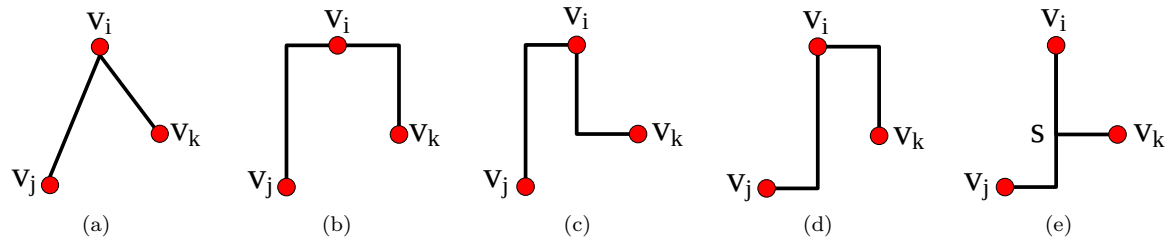


Fig. 11: Illustration of the Steiner-point relocation. (a) A vertex v_i with degree 2, and (b)-(e) four routing-path combinations of v_i.

Table 2: The setting of parameters in PORA.

Parameter						
Value						
α	β	γ	S_1	S_2	I_0	η_u
0.3	0.3	0.2	10.0	5.0	1.0	0.02
η_l	ξ_u	ξ_l	Δc	$\chi(1)$	$d_{i,j}(0)$	sp
0.0001	0.0003	0.00001	0.0004	0.0008	0.5	200

a vertex v_i with degree 2 in an obstacle-free plane, and Fig. 11(b)-Fig. 11(e) shows all the feasible routing-path combinations of v_i (as illustrated in Fig. 9, there are two feasible L-paths between each pair of vertices). Obviously, the path combination in Fig. 11(e) is the optimal structure since the introduction of Steiner point s increases the wirelength of shared paths. In the proposed algorithm, for each pin vertex $p_i \in P$, assuming that the degree of p_i is w, we enumerate all the 2^w routing-path combinations, and select the obstacle-avoiding structure with the shortest wirelength as the routing result of p_i. Then, we sort all the selected routing-path combinations in decreasing order according to their lengths of shared paths, and apply these structures to the original OARST sequentially. Note that the routing path between two vertices will not be changed once it has been determined during the refinement process, even if a different routing path is suggested by another combination later.

4 Simulation results

The proposed algorithm PORA was implemented in C language and tested on a PC with 2.3 GHz CPU and 8GB memory. A total of 25 benchmark circuits were used to validate the effectiveness of PORA, in which rst01-rst10 are synthetic benchmarks for the RSMT problem, ind1-ind5 are industrial circuits from Synopsys for the OARSMT problem, and rc01-rc10 are benchmarks for the obstacle-avoiding problem. *Pin#* and *Obs#* in each of the following tables represent the numbers of pin vertices and obstacles in the benchmarks, respectively. $\Delta w\%$ is computed as $(others - PORA)/others \times 100\%$, which provides the relatively improvement of our algorithm over other methods. Moreover, the setting of parameters used in this paper is shown Table 2.

4.1 Validation of the proposed dynamic parameters and acceleration methods

In Section 3.3.2, several dynamic strategies are proposed to update the key parameters used in PORA, including η, ξ, and χ. Moreover, in Section 3.3.3 and 3.3.4, three acceleration methods are presented to improve the computation efficiency of Physarum router, including a non-pin leaf node pruning method, a Poisson approximation method, and a cut-edge-based acceleration method. All these strategies have been analyzed in detail in the previous sections. In this subsection we experimentally validate the effectiveness of these methods using the aforementioned benchmarks. Note that since the Poisson approximation method is not the main contribution of this paper, here we do not consider its impact on the performance of Physarum router.

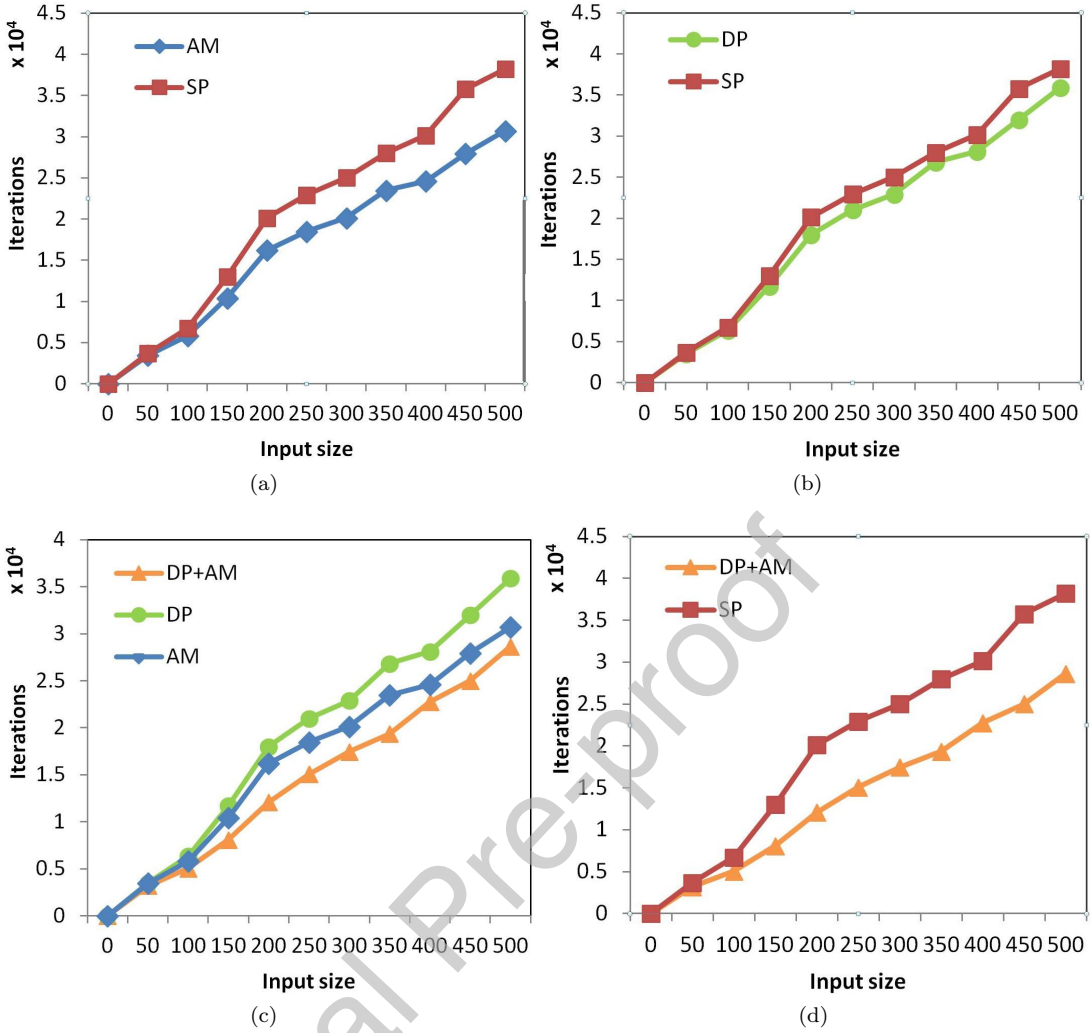


Fig. 12: Comparison results among different iteration strategies. (a) AM versus SP, (b) DP versus SP, (c) DP(+AM) versus AM, and (d) DP+AM versus SP.

Since PORA can be applied to both the obstacle-avoiding and obstacle-free routing problems, and the presence of obstacles can only increase the size of OARG, i.e., resulting in an OARG with more vertices and edges, we directly run PORA on the benchmarks without obstacles to study the convergence of Physarum router. The numbers of pin vertices in these circuits are distributed from 10 to 500. Four different iteration strategies are evaluated in our experiments:

- SP: The iteration strategy using only a set of static parameters;
- DP: The iteration strategy using only the dynamic parameters;
- AM: The iteration strategy using only the acceleration methods;
- DP+AM: The iteration strategy using both the dynamic parameters and acceleration methods.

Fig. 12 shows the comparison results with respect to the aforementioned iteration strategies, where the horizontal and vertical axis represent the numbers of pin vertices contained in the circuits and the numbers of iterations executed by PORA, respectively. The following conclusions can be drawn from Fig. 12: 1) both the acceleration methods and dynamic parameters can improve the execution efficiency of Physarum router (Fig. 12(a) and 12(b)). Moreover, the proposed acceleration methods are more effective than the dynamic parameters in speeding up the algorithm, mainly because the latter is used to strengthen the search capability of Physarum router, leading to an indirect effect on the convergence of the algorithm, 2) since the circuits with a large number of pin vertices usually generate more non-pin

Table 3: Comparison results of RSMT construction.

Benchmark	Pin#	Obs#	PORA	Wirelength		Δw%	
				[6]	[11]	[6]	[11]
rst01	10	0	590	604	590	2.32	0.00
rst02	10	0	1983	1952	1937	-1.59	-2.37
rst03	50	0	46224	46997	46533	1.64	0.66
rst04	50	0	52548	53660	54280	2.07	3.19
rst05	80	0	43551	44071	44762	1.18	2.71
rst06	80	0	72351	73246	72476	1.22	0.17
rst07	100	0	7645	7833	7723	2.40	1.01
rst08	100	0	7834	7889	7859	0.70	0.32
rst09	200	0	7889	7792	7765	-1.24	1.57
rst10	500	0	23421	24007	23820	2.44	1.68
Average						1.11	0.89

leaf nodes and cut edges after each iteration, the effects of the proposed acceleration methods are strengthened as the increase of input size (Fig. 12(a)), and 3) the acceleration methods and dynamic parameters are compatible, and a combination of them can further improve the performance of Physarum router (Fig. 12(c) and 12(d)).

4.2 Validation of RSMT construction

As discussed previously, an RSMT can be seen as a special case for an OARSMT, and PORA can generate an RSMT for a set of pin vertices without requiring any modification. Note that the constructed OARG will degenerate into a Hanan grid [3] if an input circuit contains only pin vertices.

Accordingly, in this subsection we first compare PORA with another bio-inspired algorithm, i.e., a PSO-based RSMT construction method [6]. Table 3 shows the comparison results. It can be seen that PORA achieves a -1.59%–2.44% wirelength reduction across all the test cases, with an average reduction rate of 1.11%. We analyze this result for two reasons: 1) Since [6] constructs an MST as the basic skeleton of the RSMT, which restricts the final routing structure to relatively small solution space, thereby leading to some Steiner points with unsatisfactory locations. As a result, the routing results of [6] are worse than that of PORA on most of the test cases and 2) in the circuits rst02 and rst09, although some pin vertices are scattered widely over the routing plane, [6] can still find some shortest paths connecting these vertices due to the guidance of the pre-constructed MST. Moreover, the introduction of the genetic operators, e.g., crossover and mutation operations, further enhances the pathfinding capability of PSO. Consequently, the results of PORA are slightly worse with respect to these two test cases. On the other hand, [11] presented an efficient heuristic for obstacle-avoiding routing design, which can also generate an RSMT connecting a set of pin vertices without considering the blockage of obstacles. However, since the method in [11] first constructs an MST connecting the pin vertices, and then transforms the edges in the MST into rectilinear paths directly, the locations of candidate Steiner points are restricted to very small space, leading to a number of independent routing paths. Compared with [11], it can be seen from Table 3 that PORA achieves a -2.37%–3.19% wirelength reduction, with an average reduction rate of 0.89%.

Rows 2–6 in Table 4 list the CPU times of PORA, [6], and [11] with respect to RSMT construction, respectively. To ensure a fair comparison, we required the binaries from the authors of [6] and [11], and ran both algorithms on our PC. We can see that PORA is sufficiently efficient and runs faster than [6] across all the benchmarks. Moreover, the running speed of the heuristic in [11] is very fast since it adopts a "one-pass" design manner, but the quality of the resulting RSMTs [11] is lower than ours on most of the test cases.

4.3 Verification of OARSMT construction

In this subsection, we compare PORA with several state-of-the-art OARSMT algorithms [9–11, 41]. The comparison results are reported in Table 5. [9] proposed an obstacle-avoiding

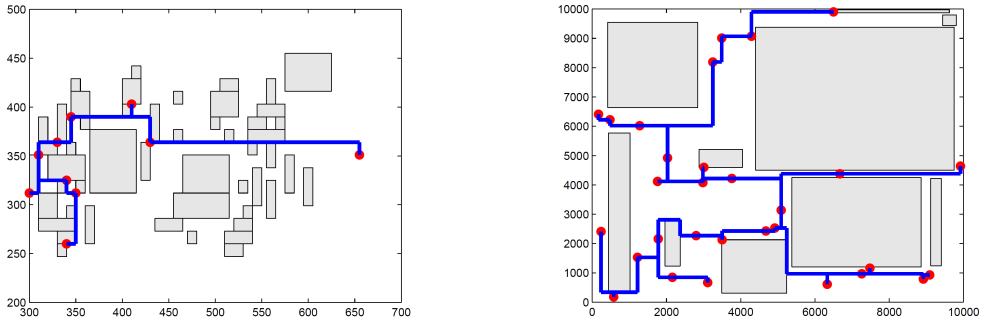
Table 4: Comparison on CUP time.

CPU time (s)				
RSMT construction (PORA/ [6]/ [11])				
rst01	rst02	rst03	rst04	rst05
0.02/0.14/0.00	0.03/0.10/0.00	0.35/0.82/0.00	0.34/0.74/0.00	1.69/2.51/0.00
rst06	rst07	rst08	rst09	rst10
1.77/2.69/0.00	2.00/3.77/0.00	3.05/4.15/0.00	4.95/8.32/0.00	9.74/14.75/0.00
OARSMT construction (PORA/ [11]/ [41])				
ind1	ind2	ind3	ind4	ind5
0.57/0.00/0.02	0.66/0.00/0.02	0.50/0.00/0.02	1.32/0.00/0.02	1.54/0.00/0.03
rc01	rc02	rc03	rc04	rc05
0.64/0.00/0.01	0.25/0.00/0.02	1.12/0.00/0.07	2.40/0.00/0.13	3.68/0.00/0.19
rc06	rc07	rc08	rc09	rc10
4.55/0.00/0.31	8.17/0.00/1.26	9.34/0.00/2.07	11.20/0.00/2.43	20.45/0.00/3.80

Table 5: Comparison results of OARSMT construction.

Benchmark	*Pin#*	*Obs#*	PORA	Wirelength				Δw%			
				[9]	[10]	[11]	[41]	[9]	[10]	[11]	[41]
ind1	10	32	622	-	639	618	609	-	2.66	-0.65	-2.13
ind2	10	43	9500	-	10000	9800	9691	-	5.00	3.06	1.97
ind3	10	59	600	-	623	613	613	-	3.69	2.12	2.12
ind4	25	79	1109	-	1126	1146	1118	-	1.51	3.23	0.81
ind5	33	71	1345	-	1379	1412	1365	-	2.47	4.75	1.47
rc01	10	10	26334	30410	27540	27630	27015	13.40	4.38	4.69	2.52
rc02	30	10	42462	45640	41930	43290	43882	6.96	-1.27	1.92	3.24
rc03	50	10	54722	58570	54180	56940	54737	6.57	-1.00	3.90	0.03
rc04	70	10	60925	63340	59050	61990	60800	3.81	-3.18	1.72	-0.21
rc05	100	10	75146	83150	75630	75685	75685	9.63	0.64	0.71	0.71
rc06	100	500	84030	149725	86381	84662	85808	43.9	2.72	0.75	2.07
rc07	200	500	113056	181470	117093	113598	113672	37.7	3.45	0.48	0.54
rc08	200	800	118277	202741	122306	119177	122057	41.7	3.29	0.76	3.10
rc09	200	1000	117722	214850	119308	117074	117993	45.2	1.33	-0.55	0.23
rc10	500	100	167781	198010	167978	167219	169443	15.3	0.12	-0.34	0.98
Average								22.41	1.72	1.77	1.16

algorithm for the λ-geometry routing plane, which can construct an OARSMT when λ is set to 2. It can be seen from Table 5 that PORA outperforms [9] across all the benchmark circuits with an average wirelength reduction of 22.41%. Obviously, the performance of PORA is much better than that of [9]. This is mainly because the method in [9] introduces plenty of redundant Steiner points into the constructed OARSMT, and most of the routing paths in the OARSMT are independent with other. In other words, routing paths generated by [9] are rarely shared with each other, thereby resulting in impractical routing results. Compared with the method in [10], which is an spanning-graph-based obstacle-avoiding routing algorithm, PORA achieves a -3.18%-5.00% wirelength reduction in the benchmarks, with an average reduction of 1.72%. [11] presented an effective four-step obstacle-avoiding algorithm, which can construct an obstacle-avoiding Steiner minimal tree in both rectilinear and octilinear architectures [14]. It can be seen from Table 5 that PORA achieves a -0.65%–4.75% wirelength reduction across all the benchmarks, with an average reduction of 1.77%. We analyze this result for two reasons: 1) since the method in [11] only selects Steiner points and corner vertices from the boundary of obstacles, the corresponding obstacle-avoiding paths are restricted to a limited routing region, leading to some routing paths with relatively long wirelength. In contrast, in the proposed method, the intermediate nodes in the generated Steiner tree are selected by exploring the design space of the whole routing plane, leading to Steiner points with satisfactory positions and 2) in [11] the structure of the final obstacle-avoiding Steiner minimal tree is mostly determined by the previously constructed MST. In contrast, in the proposed method, the high-level structure of the final OARSMT is determined by an SFCT. Moreover, the detailed routing among vertices is determined by the proposed Physarum router. In other words, PORA provides more flexibility for routing-path construction, leading to routing results with shorter wirelength. Furthermore, we compare PORA with the PSO-based hybrid routing method in [41]. Considering [41] allows diagonal



(a) (b)

Fig. 13: Routing results of two industrial circuits.

wires to be implemented on the routing plane, to ensure a fair comparison, we replace all the non-rectilinear paths generated by [41] by obstacle-avoiding rectilinear paths. It can be seen that PORA achieves a -2.13%-3.24 wirelength reduction, with an average reduction rate 1.16% and this, further proves the strong optimization capability of the proposed Physarum router.

Rows 7–13 in Table 4 list the CPU times of PORA, [11], and [41] with respect to OARSMT construction, respectively. The CPU times of [9] and [10] are not provided in the table since we cannot get the binaries/source codes of the two algorithms. As we can see from Table 4, [11] still achieves the fastest running speed among the three algorithms. Moreover, both [41] and PORA can construct an OARSMT for each benchmark circuit within reasonable runtime.

In addition, routing diagrams of two industrial circuits generated by PORA are shown in Fig. 13. It can be seen that routing paths connecting each pair of pin vertices bypass all the obstacles and the wirelength of the generated OARSMT is minimized at the same time.

4.4 Discussion on Physarum-inspired Steiner tree construction

As mentioned previously, [39] also presented an SMT construction method based on the biological behaviors of Physarum. Accordingly, in this part, we discuss the differences between PORA and the method in [39] in details, thus summarizing the major advantages of the proposed method. The details are analyzed as follows:

- *Problem model*: The method in [39] is proposed to solve the traditional SMT problem, i.e., Steiner tree construction for a set of vertices in Euclidean and rectilinear metrics. In contrast, the problem formulated in this paper is to construct a rectilinear Steiner tree connecting a set of pin vertices, while avoiding the blockage of a set of obstacles, i.e., the OARSMT construction problem. Since the traditional SMT problem even without considering obstacles has been proved to be NP-complete [8], the presence of obstacles will further increase the complexity of the problem. In other words, PORA has stronger robustness and more powerful optimization capability than the method in [39].
- *Problem size*: The size of the problem considered in [39] is far smaller than that in our problem. More precisely, in the proposed method, except for an effective Physarum computing model, several heuristics including a divide-and-conquer strategy, a non-pin leaf node pruning strategy, etc., are integrated into the proposed algorithm to fundamentally improve the performance of Physarum computing. In addition, two refinement techniques are proposed to further reduce the wirelength of the obstacle-avoiding Steiner tree generated by Physarum router in an efficient manner. All the aforementioned characteristics enable PORA to be applied to large-scale optimization problems. For example, in [39] the number of vertices contained in each test case is less than 20. In contrast, in our experiments, a total of 25 benchmark circuits were used to test the performance of PORA. Among them, the largest circuit includes 200 pin vertices and 1000 obstacles. In other words, the number of vertices that need to be considered by PORA reaches up to 4200 (each obstacle includes 4 vertices in our problem model).

5 CONCLUSION

In this paper, we have studied the obstacle-avoiding routing problem for the physical design of integrated circuits, and presented an efficient methodology called PORA to solve this problem systematically. For a given set of pin vertices and a given set of obstacles, by simulating the biological behaviors of Physarum polycephalum mathematically and thereby exploring the design space of the entire routing plane, PORA can construct a Steiner tree connecting all the pin vertices with minimized total wirelength, while avoiding the blockage of obstacles. Moreover, several heuristics and dynamic parameter strategies are integrated into the proposed Physarum model to improve the performance of PORA in a global manner. Multiple sets of benchmarks from both industry and academia are used to validate the effectiveness of the proposed method. Experimental results have confirmed that PORA leads to better results compared with several existing heuristics (a 1.16%–22.41% wirelength reduction on average). This work applies the "Physarum computing" to the physical design of integrated circuits for the first time and provides a basis for future research. Future work will consider presenting a more efficient mathematical model to simulate the biological behaviors of Physarum.

References

1. P. Yang, H. Yang, W. Qiu W et al. Optimal approach on net routing for VLSI physical design based on Tabu-ant colonies modeling, Appl. Soft Comput. 21 (2014) 376–381.
2. C. Chu, Y. C. Wong, Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design. in: Proceedings of the international symposium on Physical design, 2005, pp. 28–35.
3. M. Hanan, On Steiner's problem with rectilinear distance, SIAM J. Appl. Math. 14 (2) (1996) 255–265.
4. M. Brazil, M. Zachariasen, Steiner trees for fixed orientation metrics, J. Global Optim. 43 (1) (2009) 141.
5. C. Chu, Y. C. Wong, FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (1) (2008) 70–83.
6. G. G. Liu, G. L. Chen, W. Z. Guo, Z. Chen, DPSO-based rectilinear Steiner minimal tree construction considering bend reduction, in: Proceedings of International Conference on Natural Computation, 2011, pp. 1161–1165.
7. C. H. Liu, S. Y. Yuan, S. Y. Kuo, S. C. Wang, High-performance obstacle-avoiding rectilinear steiner tree construction, ACM Trans. Des. Autom. Electron. Syst. 14 (3) (2009) 613–622.
8. M. R. Garey, D. S. Johnson, The rectilinear Steiner tree problem is NP-complete, SIAM J. Appl. Math. 32 (4) (1977) 826–834.
9. T. T. Jing, Z. Feng, Y. Hu, et al., λ-OAT: λ-geometry obstacle-avoiding tree construction with $O(nlogn)$ complexity, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 26 (11) (2007) 2073–2079.
10. J. Y. Long, H. Zhou, S. O. Memik, EBOARST: an efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 27 (12) (2008) 2169–2182.
11. X. Huang, W. Z. Guo, G. G. Liu, C. L. Chen, FH-OAOS: a fast four-step heuristic for obstacle-avoiding octilinear Steiner tree construction, ACM Trans. Des. Autom. Electron. Syst. 21 (3) (2016) 1–30.
12. L. Li, Z. C. Qian, E. F. Y. Young, Generation of optimal obstacle-avoiding rectilinear Steiner minimum tree, in: Proceedings of International Conference on Computer-Aided Design-Digest of Technical Papers, 2009, pp. 21–25.
13. T. Huang, E. F. Y. Young, ObSteiner: an exact algorithm for the construction of rectilinear Steiner minimum trees in the presence of complex rectilinear obstacles, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 31 (6) (2013) 882–893.
14. C. K. Koh and P. H. Madden, Manhattan or non-Manhattan? A study of alternative VLSI routing architectures, in: Proceedings of the ACM Great Lakes Symposium on VLSI, 2000, pp. 47–52.
15. J. Luo, Q. Liu, Y. Yang et al., An artificial bee colony algorithm for multi-objective optimisation, Appli. Soft Comput., 50 (2017) 235–251.
16. J. Kennedy, Particle swarm optimization, in: Proceedings of International Conference on Neural Networks, 1955, pp. 1942–1948.

17. Y. C. Chuang, C. T. Chen, C. Hwang, A simple and efficient real-coded genetic algorithm for constrained optimization, Appl. Soft Comput. 38 (2016) 87–105.
18. T. Nakagaki, H. Yamada, A Toth, Intelligence: maze-solving by an amoeboid organism, Nature, 407 (6803) (2000) 470–470.
19. T. Nakagaki, M. Iima, T. Ueda, et al., Minimum-risk path finding by an adaptive amoebal network, Phys. Rev. Lett. 99 (6) (2007) 068104:1–4.
20. A. Adamatzky, Physarum machines: encapsulating reaction-diffusion to compute spanning tree, Naturwissenschaften, 94 (12) (2007) 975–980.
21. A. Tero, S. Takagi, T. Saigusa, et al., Rules for biologically inspired adaptive network design, Science, 327 (5964) (2010) 439–442.
22. T. Atsushi, K. Ryo, N. Toshiyuki, A mathematical model for adaptive transport network in path finding by true slime mold, J. Theor. Biol. 244 (4) (2007) 553–564.
23. A. Adamatzky, Routing Physarum with repellents, Euro. Phys. Jour. E, 31 (4) (2010) 403–410.
24. S. Tsuda, J. Jones, A. Adamatzky et al., Routing Physarum with electrical flow/current, Inter. Jour. Nanotech. Molec. Comput. (IJNMC), 3 (2) (2011) 56–70.
25. A. Adamatzky, Steering plasmodium with light: Dynamical programming of Physarum machine, arXiv preprint arXiv:0908.0850, (2009).
26. V. Evangelidis, J. Jones, N. Dourvas et al., Physarum machines imitating a Roman road network: the 3D approach, Scient. Repor., 7 (1) (2017) 1–14.
27. J. G. H. Whiting, R. Mayne, N. Moody et al., Practical circuits with physarum wires, Biome. Engin. Lett., 6 (2) (2016) 57–65.
28. D. Schenz, Y. Shima, S. Kuroda et al., A mathematical model for adaptive vein formation during exploratory migration of Physarum polycephalum: routing while scouting, Jour. Phys. D: Appl. Phys., 50 (43) (2017) 434001:1–14.
29. C. Gao, C. Yan, A. Adamatzky, Y. Deng, A bio-inspired algorithm for route selection in wireless sensor networks, IEEE Commu. Lett. 18 (11) (2014) 2019–2022.
30. M. C. Zhang, C. Q. Xu, J. F. Guan, et al., A novel Physarum-inspired routing protocol for wireless sensor networks, J. Distri. Sens. Netw. 9 (6) (2013) 761–764.
31. Y. N. Song, L. Liu, H. D. Ma, A. V. Vasilakos, A biology-based algorithm to minimal exposure problem of wireless sensor networks, IEEE Trans. Netw. Serv. Manag. 11 (3) (2014) 417–430.
32. K. Li, C. E. Torres, K. Thomas et al., Slime mold inspired routing protocols for wireless sensor networks, Swarm Intelligence, 5 (3-4) (2011) 183–223.
33. X. G. Zhang, S. Mahadevan, A bio-inspired approach to traffic network equilibrium assignment problem, IEEE Trans. Cybern. 48 (4) (2018) 1304–1315.
34. M. A. I. Tsompanas, G. C. Sirakoulis, A. I. Adamatzky, Evolving transport networks with cellular automata models inspired by slime mould, IEEE Trans. Cybern. 45 (9) (2015) 1887–1899.
35. H. Yang, M. Richard, Y. Deng, A bio-inspired network design method for intelligent transportation, Inter. Jour. Unconventional Comput. (2019) to appear.
36. H. Yang, Y. Deng, J. Jones. Network division method based on cellular growth and physarum-inspired network adaptation, Inter. Jour. Unconventional Comput. 13 (6) (2018) 477–491.
37. H. Yang, Q. Wan, Y. Deng. A bio-inspired optimal network division method, Physica A. 527, 121259 (2019).
38. X. Zhang, F. T. S. Chan, A. Adamatzky, et al. An intelligent physarum solver for supply chain network design under profit maximization and oligopolistic competition, Inter. Jour. Produc. Reser. 55 (1) (2017) 244–263.
39. L. Liu, Y. N. Song, H. Y. Zhang, et al., Physarum optimization: a biology-inspired algorithm for the Steiner tree problem in networks, IEEE Trans. on Comput. 64 (3) (2015) 818–831.
40. M. Caleffi, I. P. Akyildi, L. Paura, On the solution of the Steiner tree NP-hard problem via Physarum bionetwork, IEEE/ACM Trans. Netw. 23 (4) (2015) 1092–1106.
41. X. Huang, G. G. Liu, W. Z. Guo, et al., Obstacle-avoiding algorithm in X-architecture based on discrete particle swarm optimization for VLSI design, ACM Trans. Des. Autom. Electron. Syst. 20 (2) (2015) 1–28.
42. S. Pettie, V. Ramachandran, An optimal minimum spanning tree algorithm, Jour. of ACM. 49 (1) (2002) 16–34.
43. D. T. Lee, B. J. Schachter, Two algorithms for constructing a Delaunay triangulation, Inter. Jour. of Comput. Infor. Sci. 9 (3) (1980) 219–242.
44. J. L. Ganley, J. P. Cohoon, Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles, in: Proceedings of IEEE International Symposium on Circuits and Systems, 1994, pp. 113–116.
45. W. K. Chow, L. Li, E. F. Y. Young, C. W. Sham, Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach, Integ., the VLSI J. 47 (1) (2014) 105–114.