第 3 章

绕障直角结构 Steiner 最小树算法

3.1 引言

随着集成电路制造工艺的发展，特征尺寸已经进入纳米级，芯片的晶体管数目达到数十亿级别，集成电路的规模越来越大，系统更加复杂，约束也越来越苛刻，为线网布线带来更多挑战。

特征尺寸进入纳米级后，器件的尺寸逐渐变小，互连线的线宽逐渐变细、密度逐渐变大。互联线的长度飞速增加，器件变小的速度也超过互联线变细的速度，另外，互联线的延迟要比门的延迟大得多，在线网总延迟中所占比例较高。

现代 VLSI 设计中，布线区域内存在大量布线障碍，如预布线的线网、宏单元以及知识产权保护模块(Intellectual Property Block)。根据障碍所占据的布线金属层，障碍阻断了所有布线金属层，则线网必须绕过所有的障碍区域，这种问题称为绕障直角 Steiner 最小树(Obstacle-Avoiding Rectilinear Steiner Minimum Tree，OARSMT)问题。OARSMT 问题在互连引脚时需要避开障碍，是比直角 Steiner 最小树(RSMT)更难的问题，也是 RSMT 的扩展。以下是 OARSMT 的相关定义。

定义 3.1　矩形障碍　二维矩形布线区域内的一个矩形，矩形障碍不能互相叠加，可以拥有共同的顶点或者边界线。

定义 3.2　引脚　二维矩形布线区域内的一个顶点，任何引脚都不能位于障碍的内部，可以位于障碍的边界或者拐点上。

定义 3.3　绕障直角 Steiner 最小树问题　在二维的矩形布线区域内，有一组引脚 $P=\{p_1,p_2,\cdots,p_l\}$ 和一组矩形障碍 $O=\{o_1,o_2,\cdots,o_k\}$，用水平线或者垂直线将所有引脚都互连起来，且不会穿过任何障碍内部，使得所用的总线长度最短。

RSMT 问题是 OARSMT 问题中 $k=0$ 的特例。

集成度的增高使得互连线面积的不断增加，达到芯片所有面积的 30%～40%。为了降低芯片大小，布线金属层(metal layer)的数量也在不断增加。目前最大布线层数已达到 13 层，预计 2028 年会达到 17 层。这不仅增加了多层之间布

线问题，即通孔问题，还涉及多层间障碍对布线的影响。随着芯片复杂度的增长，布线工具必须限制布线长度和通孔数目，这些对芯片的性能、动态功率消耗和成品率影响很大。随着集成电路设计工艺的不断发展，允许绕线的布线层数随之逐渐增多，大幅度减少了互连线宽度和互连线间距，从而提高了集成电路的性能和密度。所以多层布线随之产生，成为了许多学者的研究热点。

接下来，本章将从单层绕障直角结构 Steiner 最小树和多层绕障直角结构 Steiner 最小树两大类问题分别介绍相关的算法构建。

3.2 基于候选 Steiner 点的 GSTP 启发式算法框架

3.2.1 引言

图中 Steiner 树问题(Steiner Tree Problem in Graphs，GSTP)是经典的组合优化问题，是计算机科学和运筹学的基本问题之一，多种工程问题都可以建模为 GSTP 来求解。在大规模集成电路设计的物理设计领域，每个线网要采用金属线将多个引脚互连起来，需要找到一种布线方法，使得所需的总时延(金属线总长)最小；在物流运输领域，货物要使用交通工具运送到各个转运站，需要找到一种货物分发路径使得运输费用最低；在城市管网设计中，管网要将多个管口(如排污口)连通，需要设计一种管网使得管道建设费用最低。因此，研究 GSTP 问题具有重要的实践意义。

1972 年，Karp 已经证明 GSTP 是 NP-难问题。为了求解 GSTP 问题，学者们提出了许多求解方法，如确定性算法、近似算法、启发式算法、局部搜索策略、计算智能算法以及约简技术。近似算法可以保证在多项式时间内找到一个可行解，使其和最优解的权重之比低于某个常数(即近似比)，文献[22]中已将近似比从文献[21]提出的 1.55 降低到了 1.39。但近似算法在设计过程中往往更注重缩小近似比而不是提高求解效率。因此，在求解大规模实际问题时，近似算法所耗费的运算时间远远超过了启发式算法。经典启发式算法运算速度快，其近似比都为 2，具有较好的求解质量，所以在工程领域得到了广泛的应用。Aragao 等人在文献[22]中对经典启发式算法进行了优化、扩展和测试对比，验证了经典启发式算法的运算复杂度并不完全和实际运行时间一致。另一种著名的启发式算法是 Rayward-Smith 提出的 ADH(Average Distance Heuristic)算法，该算法基于顶点遍历，求解质量明显优于经典启发式算法，同时消耗了较长的运算时间。局部搜索策略和计算智能算法是在已有可行解的基础上，根据邻居的定义在解空间内搜索更好的可行解，这种不可预知的迭代过程会消耗大量的运行时间，此外，它们的求解质量对初始解的选择较为敏感。约简技术是一个预处理过程，通过对求解问题的等价转换来降低问题规模，从而节约各类构造算法的运行时间。因其修改了求解问题，故在本节中没有考虑。

在工程领域中，所求 GSTP 问题的规模大，对算法调用频繁、实时性要求强，对求解质量要求高。因此，在较短时间构造出较高质量的 Steiner 树具有重要的实际意义。经典启发式算法的求解质量尚有较大的提升空间，而其他算法对运算时间的消耗较大。本节以经典启发式算法为基础来延续算法的高效率，并引入相应改进策略，以提高算法求解质量。本节提出基于候选 Steiner 点的通用启发式算法框架，记为 SPCF。该算法框架中，分别使用最短路径簇和泰森图推测两种类型候选 Steiner 点的位置，使用经典启发式算法互连候选 Steiner 点和端点来构造初始解，并引入绕行路径消除策略和基于候选 Steiner 点的改善过程。为了权衡运行时间和求解质量，算法框架中每个步骤都设计了多种可选策略，可以根据实际需求进行合理组合。

本节在 SteinLib 的测试实例上，测试了该算法框架中各种策略的效率和效果。该算法框架大大提高了 SPH 算法和 DNH 算法的求解质量；与启发式算法 ADH 算法和 KBMPH(Key node Based Minimum cost Path Heuristic)算法相比，本算法框架使用较少的运行时间获得了更优的求解结果。与近似比分别为 1.55 和 1.39 的两种最新近似算法 LCA(Loss_Contracting Algorithm)算法、LPIRR(LP-based Iterative Randomized Rounding) 算法相比，本算法框架在求解质量和运行时间方面均具有更好的性能。DW(Dreyfus-Wagner)算法是基于动态规划的一种实用的确定性算法，在本节的测试环境下，当端点数目超过 20 时，则 DW 算法很难在短时间内求得解。

本节结构如下：3.2.2 节介绍基于候选 Steiner 点的启发式算法框架；3.2.3 节将提出的算法与已有算法进行仿真测试和性能对比；3.2.4 节为小结。

3.2.2　SPCF 算法框架

1. 最短路径和候选 Steiner 点

在经典启发式算法中，最短路径是其构造 Steiner 树的基本贪心策略。GSTP 问题中，合适的候选 Steiner 点可以引导 Steiner 树构造算法靠近最优解的 Steiner 点，从而提高算法性能。不同的策略得到的候选 Steiner 点也不尽相同，因此后续构造算法得到的可行解质量也不同。

2. SPCF 算法框架主要构成

SPCF 算法框架主要由 4 部分构成：

(1) 标记候选 Steiner 点 SPC_{I}。

(2) 互连候选 Steiner 点和端点的 Steiner 树构造方法。

(3) 消除 Steiner 树中的绕行路径。

(4) 基于候选 Steiner 点 SPC_{II} 优化可行解。

选择合适的候选 Steiner 点，有益于得到质量较高的解。本节引入了两种类型的候选 Steiner 点 SPC_{I} 和 SPC_{II}。在构造 Steiner 树时，应尽可能经过这些候选 Steiner 点。

3. 标记候选 Steiner 点 SPC_I

为了便于描述，首先将两点之间的最短路径定义扩展到两顶点集合之间的最短路径簇。在图 $G=(V,E,\omega)$ 中，假设存在两个顶点集合 $A,B\subseteq V$，有以下定义。

定义 3.4　顶点集合之间的距离　$D(A,B)=\inf\{D(u,v)\mid u\in A,v\in B\}$ 称为顶点集合 A 和 B 之间的距离。

定义 3.5　连通点、最短路径和最短路径簇　如果两个顶点 $a\in A,b\in B$ 且 $D(a,b)=D(A,B)$，则对应的最短路径 $\mathrm{Path}(a,b)$ 称为顶点集合 A 和 B 之间的一条最短路径，顶点 a 和 b 称为连通点，A 和 B 之间所有最短路径构成的路径集合称为 A 和 B 之间的最短路径簇，记为 $\mathrm{SPB}(A,B)$。

文献[25]中提出了一种基于候选 Steiner 点构造 Steiner 树的通用框架（记为 RS 算法框架）：将现有启发式算法所得 Steiner 树的 Steiner 点视为候选 Steiner 点，然后使用最小端点生成树算法来连通端点和候选 Steiner 点得到 Steiner 树。RS 算法框架改进了启发式 Steiner 树构造算法的求解质量。

受 RS 算法框架启发，改进 Steiner 树质量有贡献的候选 Steiner 点很有可能位于某个最短路径上。在经典启发式算法执行中，当存在多个可选最短路径时，选择不同的最短路径可能会产生不同的重叠边，重叠边越多的 Steiner 树权重越小，但是哪个最短路径是更好的选择却很难抉择。因而，在互连所有端点过程，本节采用最短路径簇来替代最短路径以避免困难抉择，并将产生的所有非端点连通点标记成候选 Steiner 点 SPC_I。这种标记候选 Steiner 点的方法称为最短径簇扩展算法，记为 SPCH 算法。SPCH 算法是一个迭代过程，包含 4 个阶段：初始化阶段、扩展阶段、回溯阶段、更新阶段，后 3 个阶段将不断迭代直到所有端点被连通。SPCH 算法的输入为带权图 $G=(V,E,\omega)$ 和端点集合 $T\subseteq V$，输出为 SPC_I 候选 Steiner 点集合。

根据 SPCH 算法互连策略的不同可以分为单源点最短路径簇扩展算法（SS_SPCH）、多源点最短路径簇扩展算法（MS_SPCH）和多源点最短路径簇并行扩展算法（MSP_SPCH）。

1）单源点最短路径簇扩展算法

在 SS_SPCH 算法中，从一个端点出发，使用最短路径簇依次扩展到其他端点。每次迭代过程采用最短路径簇将一个最近的端点连接到已连接顶点集合（记为 CV），直到所有端点都包含在已连通顶点集合中。具体迭代过程如下。

初始化阶段：任意选择一个端点 t'，初始化为已连通顶点集合 $\mathrm{CV}=\{t'\}$。

扩展阶段：扩展阶段寻找端点 t，使得 $t\in T\setminus(\mathrm{CV}\cap T)$ 并且 $D(t,\mathrm{CV})=\inf\{D(v,\mathrm{CV})\mid v\in T\setminus(\mathrm{CV}\cap T)\}$。执行 Dijkstra 算法，以 CV 中所有顶点为源，直到找到顶点 t，暂停 Dijkstra 算法进入回溯阶段。

回溯阶段：收集 CV 和 $\{t\}$ 之间的最短路径簇 $\mathrm{SPB}(\mathrm{CV},\{t\})$。这是个递归过程：根据每个顶点到 CV 的距离和每条边的权重，可以得到当前点（初始化为 t）在

最短路径集合中的前驱节点(即从 CV 经过最短路到达 t 时,可能经过的上一个顶点),再将所有前驱节点作为当前点,若当前点是 CV 中的顶点时,该递归分支停止且该当前点就是一个连通点。

更新阶段:将 SPB(CV,$\{u\}$)上所有的顶点都加入到 CV 中。

性质 3.1　SS_SPCH 算法复杂度为 $O(|T||E|\lg|V|)$。

证明:每次迭代包含一次 Dijkstra 算法,其算法复杂度为 $O(|E|\lg|V|)$。回溯阶段和更新阶段都通过遍历 Dijkstra 算法扩展过的顶点实现,同时该顶点距离 CV 的距离修改为 0,算法会遍历每个节点,有且仅有一次,算法复杂度为 $O(|V|)$。为了连通所有端点,SS_SPCH 算法需要执行 $|T|-1$ 次迭代。因此,算法复杂度为 $O(|T||E|\lg|V|)$。

在实现时,除了首次迭代,每次 Dijkstra 算法不需要从头开始执行,只需要将新加入的已连通顶点作为源点(即距离源点距离为 0)继续执行算法即可。算法 3.1 为 SS_SPCH 算法伪代码。

算法 3.1: SS_SPCH($G(V,E,\omega)$, T)

```
输入:  G(V,E,ω)                          //带权图
       T                                 //端点集合
输出: SPC1                               //候选 Steiner 点集合
1   Begin
2      SPCl = ∅;
3      SPB = ∅;                          //最短路径簇中的顶点
4      VHeap = ∅;                        //顶点二进制堆
5      for each 顶点 u∈V
6         u.dist = ∞;
7      任意选择一个端点 t;
8      CV = {t};                         //初始化已连通顶点集合
9      t.dist = 0;
10     VHeap.push.back(t);
11     Repeat
12        u = VHeap.pop_heap();          //弹出具有最小 dist 值的顶点
13        if (u∈T)
14           TraceBack(u);
15           CV = SPB ∪ CV;
16           SPB = ∅;
17           continue;
18        for each 与 u 关联的边 e(u,v)
19           if v.dist > u.dist + ω(e)
20              v.dist = u.dist + ω(e);
21              VHeap.push_back(v);
22     Until T⊆CV;
23  return SPC1;
24  Function TraceBack(u)                //递归过程
25     if u∈CV
```

```
26          if(u ∉ T)
27              SPC1 = SPC1 ⋃ {u};
28          return;
29      for each 与 u 关联的边 e(u,v)
30          if v.dist = u.dist - ω(e)
31              TraceBack(v);
32      u.dist = 0;
33      SPB = SPB ⋃ {u};
34      VHeap.push_back(u);
35  return;
```

2）多源点最短路径簇扩展算法

在 MS_SPCH 算法中，使用最短路径簇依次将最近的两个已连通节点集合连接起来，直到剩下一个已连接节点集合。具体迭代过程如下。

初始化阶段：每个节点初始化成一个已连接节点的节点集，$CV_i=\{t_i\}$，$i=1,2,\cdots,|T|$。

扩展阶段：采用泰森图构造算法，找到最靠近的一对已连通顶点集合 CV_i 和 CV_j。以每个已连通顶点集合分别作为一个泰森种子开始构造泰森图，记录遍历过程中得到的桥边，每轮扩展中 Dijkstra 算法遍历顶点范围 Range 为当前找到桥边的最小跨度；每轮扩展结束时就有两个邻接泰森单元之间的所有主桥边 $MBs(CV_i,CV_j)$ 被找到，相应的两个泰森种子 CV_i 和 CV_j 最靠近，暂停构造泰森图进入回溯阶段。

回溯阶段：收集 CV_i 和 CV_j 之间的最短路径簇 $SPB(CV_i,CV_j)$。这个过程与 SS_SPCH 算法的回溯阶段类似，所不同的是将 $MBs(CV_i,CV_j)$ 中每个主桥边的两个关联点都作为当前点开始递归。

更新阶段：构造一个新已连接节点的节点集 $CV_{new}=SPB(CV_i,CV_j)\cup CV_i\cup CV_j$ 代替 CV_i 和 CV_j，作为新的泰森种子。

性质 3.2 MS_SPCH 算法复杂度为 $O(|T||E|\lg|V|)$。

证明：扩展阶段包括一次泰森图构建策略，以及选取最小跨度的一组主桥边集合。泰森图构建的时间复杂度为 $O(|E|\lg|V|)$，主桥边的边数为 $O(|E|)$，选择最小跨度的一组主桥边集合的时间复杂度为 $O(|E|)$。在回溯阶段和更新阶段，与性质 3.1 类似，算法复杂度为 $O(|V|)$。MS_SPCH 算法需要执行 $|T|-1$ 次迭代。因此 MS_SPCH 算法复杂度为 $O(|T||E|\lg|V|)$。

在实现时，除了首次迭代，每次泰森图构造算法都不需要从头开始，只需要将 CV_{new} 作为一个泰森种子继续执行泰森图构造算法即可。与 SS_SPCH 算法相比，避免了对起始点的敏感。在扩展阶段，为了确保找到跨度最小的主桥边需要扩大扩展区域，并从所有的桥边中找出跨度最小的桥边。

3）多源点最短路径簇并行扩展算法

在 MSP_SPCH 算法中，每次迭代中采用最短路径簇将若干对距离最小的已

连接节点集合连接起来，直到剩下一个已连通顶点集合。具体迭代过程如下。

初始化阶段、回溯阶段和更新阶段均与 MS_SPCH 算法类似。所不同的是在初始化阶段和更新阶段将所有泰森种子(已连通顶点集合)设置为未标记状态，每轮回溯阶段和更新阶段要处理多个最短路径簇。

扩展阶段：通过泰森图构造算法，寻找最接近的数对已连通顶点集合。本阶段包含一个更小的迭代过程，该迭代过程类似于一次 MS_SPCH 算法的扩展阶段，找到一对最靠近未被标记的泰森种子，并记录这些泰森种子。若一个泰森种子被标记了，则其所在的泰森单元暂停扩展，在本轮扩展阶段中弃用与之关联的桥边。当所有泰森种子都被标记了时，进入回溯阶段。

性质 3.3　MSP_SPCH 算法复杂度为 $O(|T||E|\lg|V|)$。

证明：扩展阶段包括一次泰森图的构建策略，耗时 $O(|E|\lg|V|)$。选择最小跨度主桥边集合的次数和耗时与 MS_SPCH 算法相同。回溯阶段和更新阶段，与性质 3.1 类似，算法复杂度为 $O(|V|)$。MSP_SPCH 算法需要执行不超过 $|T|-1$ 次迭代。因此 MSP_SPCH 算法复杂度为 $O(|T||E|\lg|V|)$。

在实现时，除了首次迭代，每次泰森图构造算法都不需要从头开始，只需要将每个新构成的已连通顶点集合作为一个种子继续执行算法即可。与 MS_SPCH 算法相比，所需的迭代次数更少，但需要将扩展阶段被弃用的桥边代入到下一轮迭代中。

4. Steiner 树的构造

本节构造 Steiner 树来连通端点和候选 Steiner 点。DNH 的算法复杂度小于 SPH，SPH 算法结果要优于 DNH 算法。文献[22]的算法 SPH 所需的运行时间并不明显多于 DNH 算法，SPH 所得解的质量普遍优于 DNH，本节后续的测试也验证了这一结果。本节中除特殊说明以外，均采用 SPH 算法作为 Steiner 树的构造方法(记作 CA)。

5. 绕行路径的消除

在 Steiner 树的构造算法中引入的候选 Steiner 点，不能保证一定有利于缩小 Steiner 树总权重。候选 Steiner 点可能导致 Steiner 树中出现绕行路径，为了消除这种绕行路径，本节使用两种可选策略，分别记为 EDP_RS 和 EDP_KPE。EDP_RS 策略重复调用 RS 算法框架，并构造 Steiner 树，直到求解质量不再被改进。但是对于不同的 GSTP 问题，重复构造 Steiner 树的次数无法预测。实际测试中重复的次数往往较小，本节设置将重复次数设置为常数 5。因此，EDP_RS 策略的算法复杂度与构造算法相同(使用 SPH 时为 $O(|T||E|\lg|V|)$)。

6. 基于 $SPC_{Ⅱ}$ 优化可行解

SPCH 算法和 RS 算法框架都是采用基于最短路径贪心策略来标记候选 Steiner 点的。但并非所有 GSTP 问题最优解的 Steiner 点都可以使用这种贪心策略得到。对于一些特殊 GSTP 问题(称为困难 GSTP 问题)，在基于最短路径贪心策略的扩展时，不能经过该问题最优解的一些 Steiner 点(称为困难节点)，因此会

影响后续的构造算法求解质量。例如，定义 3.6 中给出的完全轮图就是一个典型的困难 GSTP 问题。

定义 3.6　完全轮图　在 GSTP 问题中，有 $l(l>2)$ 个端点、1 个非端点顶点（称为中心点），图 G 是一个完全图，任何两个端点相连的边称为内部边，其边长为 $2\times\beta$，其他边称为辐条边，辐条边长为 $\beta+\tau$，其中 $\beta\gg\tau>0$，这种 GSTP 问题称为完全轮图。

性质 3.4　若 GSTP 问题中包含完全轮图或部分完全轮图，以端点为种子构造泰森图，则中心节点是交界点。

证明：由完全轮图和部分完全轮图的定义可知，在以端点为种子构造的泰森图中，中心节点的邻居都是端点，且邻居个数超过 3，因此中心节点是交界点。

为了提高困难 GSTP 问题的求解质量，基于性质 3.4，本节算法提出 3 种可选的改善策略：IS-Ⅰ、IS-Ⅱ、IS-Ⅲ。

首先，以每个节点视为一个种子，并构造出相应的泰森图。

改善策略 IS-Ⅰ：以泰森图的交界点为 $\mathrm{SPC}_{\mathrm{Ⅱ}}$，然后对每个 $\mathrm{SPC}_{\mathrm{Ⅱ}}$ 执行插入关键点局部搜索，使用构造算法连接该 $\mathrm{SPC}_{\mathrm{Ⅱ}}$ 和当前可行解的关键节点，若得到的解比当前解更好，则以该解代替当前解。

性质 3.5　改善策略 IS-Ⅰ算法复杂度为 $O(|V||T||E|\lg|V|)$。

证明：泰森图构造时间复杂度为 $O(|E|\lg|V|)$，泰森图交界点的个数为 $O(|V|)$，在改善策略 IS-Ⅰ需要执行 $O(|V|)$ 次 SPH 算法构造 Steiner 树，每次耗时 $O(|T||E|\lg|V|)$。因此，改善策略 IS-Ⅰ算法复杂度为 $O(|V||T||E|\lg|V|)$。

改善策略 IS-Ⅰ通过消耗较长的算法运行时间，获得了较好的求解结果。为了提高算法效率，IS-Ⅱ策略通过减少 $\mathrm{SPC}_{\mathrm{Ⅱ}}$ 候选 Steiner 点的个数来减少构造算法的执行次数。

改善策略 IS-Ⅱ先使用构建算法连接每一个交界点和端点，将得到的 Steiner 树 T_{new} 的 Steiner 点看作 $\mathrm{SPC}_{\mathrm{Ⅱ}}$，并执行与 IS-Ⅰ策略相同的插入关键节点局部搜索。

性质 3.6　改善策略 IS-Ⅱ算法复杂度为 $O(|T|^2|E|\lg|V|)$。

证明：在改善策略 IS-Ⅱ需要执行 $O(|T|)$ 次构造函数，其他过程与改善策略 IS-Ⅰ相同。因此，改善策略 IS-Ⅱ算法复杂度为 $O(|T|^2|E|\lg|V|)$。

为了进一步提高改善策略的效率，避免多次执行构造算法，IS-Ⅲ策略中在当前可行解和 T_{new} 之间执行净化搜索策略，只需执行两次构造算法，因此有性质 3.7。

性质 3.7　改善策略 IS-Ⅲ算法复杂度为 $O(|T||E|\lg|V|)$。

7. SPCF 算法框架

SPCF 算法框架共包含了 3 个阶段，伪代码如算法 3.2 所示。表 3.1 中列出了各个阶段不同可选策略的时间复杂度。为了权衡考虑求解质量和运行时间，可以选择不同的策略组合，构成所需的算法。表 3.2 中列出了基于 SPCF 算法框架的 9 组算法，每组算法可以使用不同 $\mathrm{SPC}_{\mathrm{I}}$ 标记策略。前 6 组算法是通过增加策略或

者替换策略来改进求解质量，后 3 组算法则侧重于减少运行时间。

```
算法 3.2: SPCF (G(V,E,ω),T)
输入:  G(V,E,ω)                                  //带权连通图
       T                                          //端点集合
输出: Tree                                        //可行解
1   Begin
2      SPC_I = SPCH(G(V,E,ω),T);                  //SPC_I 候选 Steiner 点标记过程;
3      pre_Tree = CA(G(V,E,ω),T,SPC_I );          //基于 SPC_I 构造初始解 pre_Tree;
4      Tree = IS(G(V,E,ω),T,pre_Tree);            //基于 SPC_II 改善初始解质量;
5   return Tree;
```

表 3.1　各阶段可选策略及时间复杂度

类　型	策　略	时间复杂度
SPCH	MSP_SPCH	$O(\|T\|\|E\|\lg\|V\|)$
	MS_SPCH	$O(\|T\|\|E\|\lg\|V\|)$
	SS_SPCH	$O(\|T\|\|E\|\lg\|V\|)$
CA	DNH	$O(\|E\|\lg\|V\|)$
	SPH	$O(\|T\|\|E\|\lg\|V\|)$
EDP	EDP_RS	$O(\|T\|\|E\|\lg\|V\|)$
	EDP_KPE	$O(\|E\|\lg\|V\|)$
IS	IS-Ⅰ#	$O(\|V\|\|T\|\|E\|\lg\|V\|)$
	IS-Ⅰ	$O(\|V\|\|T\|\|E\|\lg\|V\|)$
	IS-Ⅱ	$O(\|T\|^2\|E\|\lg\|V\|)$
	IS-Ⅲ	$O(\|T\|\|E\|\lg\|V\|)$

IS-Ⅰ# 表示在改进策略中的构造算法也执行了绕行路径消除策略

表 3.2　基于 SPCF 算法框架的 9 组算法

SPCF 算法框架		
算　法	组　成	时间复杂度
*_SPCF-Ⅰ	*_SPCH+DNH	$O(\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅱ	*_SPCH+SPH	$O(\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅲ	*_SPCH+SPH+EDP_RS	$O(\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅳ	*_SPCH+SPH+EDP_KPE	$O(\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅴ	*_SPCH+SPH+EDP_KPE+IS-Ⅰ#	$O(\|V\|\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅵ	*_SPCH+SPH+EDP_RS+IS-Ⅰ#	$O(\|V\|\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅶ	*_SPCH+SPH+EDP_RS+IS-Ⅰ	$O(\|V\|\|T\|\|E\|\lg\|V\|)$
*_SPCF-Ⅷ	*_SPCH+SPH+EDP_RS+IS-Ⅱ	$O(\|T\|^2\|E\|\lg\|V\|)$
*_SPCF-Ⅸ	*_SPCH+SPH+EDP_RS+IS-Ⅲ	$O(\|T\|\|E\|\lg\|V\|)$

* 表示使用 MSP、MS 和 SS 3 种 SPC_I 标记算法；IS-Ⅰ# 表示在改善策略中的构造算法也执行了绕行路径消除策略。

3.2.3 测试与对比

1. 测试实例与测试安排

为了便于测试和对比，本节使用 GSTP 测试用例库 SteinLib 中的 389 个例子作为测试实例，称作 LittleCase 类。LittleCase 类的测试实例，端点数不超过 20，顶点数不超过 1000，基本信息见表 3.3。

测试安排：对 9 组 SPCF 算法进行测试，比较各阶段每种可选策略的性能；将 SPCF 算法与五种对比算法进行性能对比。每种算法对每个测试用例执行 20 次，并计算测试用例集合误差率和相对算法运行时间。

表 3.3 LittleCase 类的测试实例基本信息

数量		描述	$\|V\|$	$\|E\|$	$\|T\|$
LittleCase	389	$\|V\|\leqslant 1000$ 且 $\|T\|\leqslant 20$：euclidean(19 个)、fst(30 个)、hard(11 个)、incidence(225 个)、random(44 个)、vlsi(60 个)	6～1000	9～204 480	3～20

2. SPCF 算法框架中可选策略性能测试

在可选策略的性能测试过程中，在经典启发式算法的基础上，不断增加可选策略，以验证其有效性。如表 3.4 所示，两种经典启发式算法和 9 组 SPCF 算法的性能对比。由 DNH 行和 SPH 行可知，SPH 算法的求解质量明显优于 DNH 算法，且相对运行时间略低于 DNH 算法，验证了 SPH 算法在实践中的优越性。*_SPCH-Ⅰ算法与 DNH 算法对比，平均相对误差均从 16.87%降低到了 10%以下。*_SPCH-Ⅱ算法与 SPH 算法对比，平均相对误差也从 10.43%降低到了 8.7%左右。这说明 3 种 SPCH 策略均在优化求解质量方面具有明显的积极效果，且使用 SPH 算法作为构造算法所得的求解质量更好。因此，本节的构造算法默认采用 SPH 算法。

表 3.4 SPCF 算法框架中各种可选策略的性能对比

算　法	AVG	BEST	RT
DNH	16.87	—	1.75
SPH	10.43	7.1	1.73
MSP_SPCF-Ⅰ	9.47	—	16.88
MSP_SPCF-Ⅱ	8.76	7.37	15.84
MSP_SPCF-Ⅲ	8.43	7.04	17.16
MSP_SPCF-Ⅳ	6.83	6.83	20.51
MSP_SPCF-Ⅴ	**1.34**	**0.93**	**109.46**
MSP_SPCF-Ⅵ	1.59	0.85	72.72
MSP_SPCF-Ⅶ	2.20	1.36	51.99

续表

算　法	AVG	BEST	RT
MSP_SPCF-Ⅷ	2.98	1.82	29.39
MSP_SPCF-Ⅸ	3.94	2.33	27.62
MS_SPCF-Ⅰ	9.4	—	7.07
MS_SPCF-Ⅱ	8.67	7.28	6.54
MS_SPCF-Ⅲ	8.42	6.98	7.68
MS_SPCF-Ⅳ	7.72	6.76	9.98
MS_SPCF-Ⅴ	**1.33**	**0.87**	**88.84**
MS_SPCF-Ⅵ	1.59	0.83	56.57
MS_SPCF-Ⅶ	2.18	1.37	38.05
MS_SPCF-Ⅷ	2.96	1.79	17.01
MS_SPCF-Ⅸ	3.95	2.38	15.98
SS_SPCF-Ⅰ	9.18	6.64	4.02
SS_SPCF-Ⅱ	8.63	6.13	3.02
SS_SPCF-Ⅲ	8.4	6.07	4.02
SS_SPCF-Ⅳ	7.74	5.87	5.53
SS_SPCF-Ⅴ	**1.33**	**0.54**	**100.32**
SS_SPCF-Ⅵ	1.57	0.48	60.00
SS_SPCF-Ⅶ	2.24	0.85	36.97
SS_SPCF-Ⅷ	3.01	1.08	12.22
SS_SPCF-Ⅸ	3.96	1.96	11.64

AVG 表示测试实例集合平均误差率；BEST 表示测试实例集合最佳误差率；RT 表示测试实例集合相对平均运行时间。

3 种 SPCH 策略在相对运行时间方面 SS_SPCF-Ⅰ和 SS_SPCF-Ⅱ所耗时间相对较少，仅为 SPH 算法的 2～4 倍，MSP_SPCF-Ⅰ和 MSP_SPCF-Ⅲ所耗时间为 SPH 算法的 9～10 倍。主要原因在于后两者扩展范围的增大和对桥边的处理。*_SPCH-Ⅲ和 MS_SPCF-Ⅳ行表明，两种绕行路径消除策略可以有效改进求解质量。EDP_KPE 策略在求解质量方面要优于 EDP_RS 策略，实际消耗运行时间也较多。*_SPCH-Ⅶ～MSP_SPCF-Ⅸ 3 组算法表明，基于 $SPC_{Ⅱ}$ 的改善策略明显提高了求解质量，平均相对误差率从 8%左右降低到 3%左右。其中 IS-Ⅰ改善策略对求解质量改进最多，耗时最多。而 IS-Ⅲ改善策略则耗时最少，求解质量比另外两种改善策略略差。为进一步提高求解质量，*_SPCH-Ⅵ和 MS_SPCF-Ⅴ两组算法在 IS-Ⅰ改善策略的构造算法中执行相应绕行路径消除策略，使得这两组算法具有最好的求解质量和也消耗最长的运行时间。如表 3.4 中 BEST 列所示，在 SS_SPCH 算法和 SPH 算法中随机选择不同的起始点，所得的解质量相差较大。

因此，在不同应用需求中，可选择 SPCF 算法框架不同的策略组合，权衡求解质量和所耗运行时间。

3. 与其他算法对比

本节将 SS_SPCF-Ⅴ算法与两种启发式算法、两种近似算法和一种动态规划算法进行性能对比。ADH 算法是一种著名的启发式算法，具有比经典启发式算法好得多的求解质量。ADH 算法过程为：首先构造图 G 闭包完全图；将每个端点初始化为一棵子树，共 $|T|$ 棵子树；随后每次迭代过程根据平均距离函数从所有顶点中选择一个最佳的顶点来合并两棵子树，直到所有子树合并为一棵树。该算法在迭代过程中，遍历了所有可能的顶点，因此可以引导所求解经过或靠近困难节点，这也是 ADH 算法求解质量高的主要原因之一。ADH 的算法复杂度是 $O(|V|^3)$，近似比是 $2(1-1/|T|)$。余燕平等人在文献[32]提出的 KBMPH 算法是 SPH 算法的变体，可以改进 SPH 算法的求解质量。KBMPH 算法过程为：首先构造图 G 闭包完全图并保存任意两顶点间的最短路径，再根据每对端点间最短路径经过的非端点顶点和参数 K 确定一个加权顶点集合 F，所有有经过 F 中顶点的路径都需要采用参数 λ 修正，最后根据修正后的路径使用 SPH 算法构建 Steiner 树。该算法也属于最短路径的贪心算法，因此很难指导所求解经过或者靠近困难节点。参数 λ 和 K 的选择对求解结果的影响很大，不同的例子对参数的依赖也不同，本节综合多次的测试结果将参数设置为 $\lambda=0.9$，$K=|V|/4.5$。KBMPH 的算法复杂度是 $O(|V|^3)$，近似比是 $2(1-1/|T|)/\lambda$。Dreyfus 和 Wagner 提出的 DW 算法是基于动态规划的确定性算法，具有较强的实践性。DW 算法过程为：首先构造图 G 闭包完全图，然后根据递归式(3.1)求解，其中 $S(u,T')$ 表示顶点 u 连通端点集合 T' 的最小费用。DW 算法复杂度是如式(3.2)所示，当端点个数小于某个固定值时，该算法是多项式算法。Robins 等人在文献[21]提出的 LCA 算法是一种近似算法。LCA 算法过程为：首先构造图 G 闭包完全图，构造最小端点生成树作为初始可行解，构造所有的 k-满部件；在每次迭代中，根据收益比从所有 k-满部件中选择一个最佳的 k-满部件插入到可行解；直到所有 k-满部件都不能改善可行解为止。LCA 算法复杂度为 $O(|V|^3+|T|^k\times(|V|-|T|)^{k-2}+k\times|T|^{2k+1}\lg|T|)$，$k\to\infty$近似比为$(1+0.5\times\ln 3)\approx 1.55$，本节测试中 k 取 3。文献[22]提出的 LPIRR 算法是目前最新的近似算法。LPIRR 算法基于有向 k-满部件，每次迭代采用线性规划方法评估一个有向 k-满部件在构造 Steiner 树过程中所起的积极作用，选择一个最佳的有向 k-满部件用来参与构造一个 Steiner 树，最多需要迭代 $|T|$ 次。$k\to\infty$近似比为 $\ln 4\approx 1.39$，本节测试中 k 取 3 和 4。LPIRR 算法是按一定概率随机选择 k-满部件，每次运行结果可能不同，本节在重现 ADH、LCA、LPIRR 算法时，均使用了 RS 通用算法框架，以提高求解质量。采用 Floyd-Warshall 算法构造图 G 闭包完全图；采用 DW 算法构造所有的 k-满部件；使用 IBM CPLEX 软件包求解线性规划。

$$S(u,t')=\min_{v\in V}\left(D(u,v)+\min_{\substack{E\subseteq T'\\F=T'\backslash E}}(S(v,E),S(v,F))\right),$$

$$u\in V,\quad T'\subseteq T\backslash\{q\},\quad q\in T \tag{3.1}$$

$$O\left(\frac{|V|^3}{2}+|V|^2(2^{|T|-1}-|T|-1)+\frac{|V|(3^{|T|-1}+2^{|T|}+3)}{2}\right) \tag{3.2}$$

表 3.5 为 SS_SPCF-Ⅴ算法与其他 5 种算法的性能对比。ADH 算法运行时间是 SS_SPCF-Ⅴ算法的 13 倍，且求解质量略差，原因在于 ADH 算法每次迭代中都逐个检查每个顶点，以选中最佳的顶点来合并两棵子树，该算法有机会引导可行解经过或靠近困难节点。测试中，KBMPH 算法中选择的 F 顶点集合常常会集中在个别端点附近，而且很难引导所求解经过或者靠近最优解中的困难节点，所以对 Steiner 树的优化有限。KBMPH 算法的求解质量仅略优于 SPH 算法。KBMPH 算法在构造图 G 闭包完全图时，还需保存所以顶点对之间的最短路径，所以运行时间略长于 ADH 算法。虽然 LCA 算法和 LPIRR 算法的理论近似比都较小，但当参数 $k\to\infty$时其运行时间是无法承受的。即便只选择较小的 k 值，算法的运行时间也是 SS_SPCF-Ⅴ算法的几十倍乃至几百倍，求解质量也远不及 SS_SPCF-Ⅴ算法。由 LPIRR-3 算法和 LPIRR-4 算法的结果可以看出，当参数 k 增大时，运行时间增加，求解结果也明显变好。DW 算法在求解端点数较少的例子时，其运行时间较少，当端点数超过 20 时，其运行时间也是无法承受的。

表 3.5　SPCF 算法与其他算法性能对比

算　　法	AVG	BEST	RT
SS_SPCF-V	**1.33**	**0.54**	**100.32**
ADH	1.39	—	1310.65
LCA-3	5.26	—	2748.62
LPIRR-3	5.06	4	29 911.75
LPIRR-4	3.08	2.2	70 636.35
DW	0	—	67 200.44
KBMPH	10.33	7.01	1315.68

3.2.4　小结

本节提出一种求解 GSTP 问题的算法框架 SPCF，SPCF 包含了 $SPC_{Ⅰ}$候选 Steiner 点的标记、Steiner 树的构造、绕行路径的消除和基于 $SPC_{Ⅱ}$候选 Steiner 点的优化。每个阶段都设计了数种可选策略。这些策略对改善 GSTP 问题求解质量均起到了较为明显的积极作用，而且所耗运行时间也相对较少；与现有算法相比，具有求解质量高、运行时间短的特点，对 GSTP 问题的启发式算法研究的意义是显而易见的。

SPCF 算法框架可根据具体工程应用问题的需求进行合理组合和拓展，用于求解各种场合的 GSTP 问题或带约束的 GSTP 问题。

3.3 基于绒泡菌算法的绕障直角结构 Steiner 最小树算法

多头绒泡菌的疟原虫是一种大型变形虫细胞，由于其在寻路、避险和网络构建等方面的智能行为，近年来受到广泛关注。受这种原始生物的行为的启发，本研究探索了多头绒泡菌的优化能力，提出了第一个基于绒泡菌的集成电路物理设计绕障布线算法。本节使用一种新的营养消耗数学模型来模拟多头绒泡菌的觅食行为，从而提出了一种称为绒泡菌布线器的高效布线工具。利用所提出的布线方法，对于给定的一组引脚节点和给定的一组芯片上的功能模块，可以自动构建一个连接所有引脚节点同时避免功能模块拥塞的直角 Steiner 最小树。此外，所提出的算法结合了一些启发式算法，其中包括分治策略、一个非引脚叶节点修剪策略、一个动态参数策略等，以从根本上提高绒泡菌布线器的性能。

3.3.1 引言

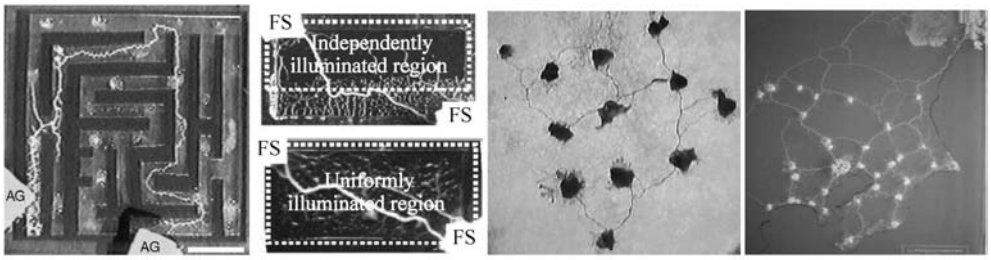
布线在大规模集成电路的设计中起着重要的作用。互联系统的构建对于每个信号网络都是至关重要的，该互联系统旨在连接硅片上的一组引脚节点，同时使得总导线长度最小。

自从 Hanan 网格在 1966 年被提出以来，直角 Steiner 最小树问题由于其实际意义而被广泛研究。RSMT 现在正被应用于电子设计自动化的许多领域。特别是在集成电路的布线设计中，它通常用于构建连接芯片上许多引脚节点的初始网络拓扑。然而，电路布线的大多数先前工作都假设了布线平面是无障碍的。随着集成电路密度的不断增加，越来越多的可重用组件，如宏块、IP 核和预布线网络，被集成到单个芯片中。这些组件不能在布线过程中运行。因此，绕障 RSMT 的构建问题变得尤为重要。此外，RSMT 的构建问题即使不考虑障碍也已被证明是 NP-难的问题，障碍的存在将进一步增加布线设计的复杂性。

在过去的几年里，已经有许多策略用于 OARSMT 的构建问题。例如，在文献[47]中，提出了一种称为 λ-OAT 的绕障布线算法，以在 λ 几何平面中构建 Steiner 树。特别地，当 λ 的值被设置为 2 时，算法可以有效地生成 OARSMT。文献[48]提出了一种有效的四步布线算法，为给定的一组引脚和障碍物构建出绕障直角 Steiner 树(Obstacle-Avoiding Rectilinear Steiner Tree，OARST)。为了减少生成的 Steiner 树的总线长，本节将基于边的全局优化技术以及称为“段转换”的局部优化技术，作为一个后处理步骤，从而增加了共享布线路径的线长。文献[49]提出了一种快速的四步启发式方法来构建直角和 X 结构中的绕障 Steiner 树。该算法首先构建一个连接给定引脚节点的无障碍欧几里得最小生成树。然后，基于两个预先计算的查找表，通过将 MST 中的边转换成直角路径或 X 结构路径来生成绕障 Steiner 树。此外，为了避

免障碍物的堵塞，提出了一种有效的绕障策略，从障碍物边界引入一些额外的节点作为中间节点。最后，对生成的绕障 Steiner 树进行优化，进一步减少总线长，从而生成绕障 Steiner 最小树。然而，上述启发式算法的缺点是求解效率和布线质量之间不平衡，可能会导致得到低质量的解，否则就需要较长的计算时间。此外，文献[50]和文献[51]还提出了几种精确的布线算法来生成最佳的绕障直角 Steiner 最小树(Obstacle-Avoiding Rectilinear Steiner Minimal Tree，OARSMT)。不幸的是，因为它们的时间复杂度为指数级别，导致这些方法在实际的工业制造中并不实用。

另一方面，在过去的几十年里，科学家通过学习生物系统的智能行为，受到启发，创造了许多强大的计算方法。特别地，作为变形虫单细胞生物的多头绒泡菌(以下称为绒泡菌)，由于其在路径查找和网络构建方面的优异能力，近年来吸引了高度的研究兴趣。例如，在文献[56]中，绒泡菌算法找到连接分布在迷宫的两个不同出口处的两个食物源的最短路径(见图 3.1(a))。在文献[57]中，绒泡菌算法在独立照明区域中形成的路径长度本能地减少(见图 3.1(b))，表现出强大的避险能力。在文献[58]中，绒泡菌算法形成连接多个食物来源的生成树(见图 3.1(c))，就边数和总长度而言，这与经典 MST 算法生成的结果非常接近。在文献[59]中，代表东京地区城市位置的 36 个食物源被用来验证绒泡菌算法的网络构建能力。相应地，最终的网络在成本、效率和容错性方面与东京的真实铁路系统非常接近(见图 3.1(d))。此外，在文献[60]中，提出了一个自适应动态方程来模拟绒泡菌的生物机制，如身体收缩和信号传输，从而产生了第一个用于“绒泡菌算法计算”的数学模型，该模型适用于复杂的工程问题。此后，许多基于绒泡菌算法计算的研究被用来解决工程和工业中的布线问题。更重要的是，绒泡菌算法计算现在正被应用到各种实际领域，包括无线传感器网络、运输网络、网络划分、供应链网络、Steiner 树问题等。



(a) 迷宫问题　(b) 最短路径构建　(c) 生成树构建　(d) 铁路网络构建

图 3.1　绒泡菌生物行为

受到上面讨论的智能行为的启发，进一步探索绒泡菌算法的优化潜力，以便通过高效的计算方式解决复杂的图形优化问题。此外，由于集成电路布线的高度复杂性，寻找一种系统的新的方式来构建 OARSMT 显得尤为重要。因此，本节提出了 PORA——一个由绒泡菌启发的绕障布线算法。表 3.6 列出了论文中经常使用的缩写。本节的贡献总结如下。

表 3.6 本节常用缩写列表

缩　写	描　述
RSMT	直角 Steiner 最小树
OARST	绕障直角 Steiner 树
OARSMT	绕障直角 Steiner 最小树
SFCT	Steiner 全连通树
MST	最小生成树
PSO	粒子群优化

(1) 本节是第一个将绒泡菌计算应用于集成电路布线设计的工作。一方面，进一步拓展了绒泡菌算法的应用领域，证明了其强大的网络构建能力；另一方面，从电路布线的角度提出了一种新颖的、受绒泡菌启发的 OARSMT 构建算法。这项工作为今后的研究提供了基础。

(2) 系统地探索了绒泡菌算法的优化能力，并提出了一个强大的布线工具，称为绒泡菌布线器。此外，结合了几个启发式策略到绒泡菌布线器中，以从根本上提高其性能。本节的工作在以下几方面弥补了前人工作的不足：当多条代价相同的路径连接到复杂图中的一个公共节点时，绒泡菌布线器可以保证生成网络的连通性。绒泡菌布线器可以保证绒泡菌计算的终止时间的准确性。

(3) 本节提出了一种有效的分治策略来指导绒泡菌布线器的执行，从而可以显著提高整个算法的效率。该策略基于几个关键模型/技术，包括预先构建的 Steiner 全连通树(SFCT，在 3.3.3 节中定义)、绕障布线图(OARG，在 3.3.3 节中定义)和多角度评估机制。

(4) 本节采用营养吸收/消耗模型来模拟绒泡菌的生物学行为。此外，提出了一种动态参数调整策略来加强绒泡菌算法的搜索能力。PORA 可以在合理的运行时间内生成高质量的 OARSMT，这对科学研究和工业生产都很有价值。

3.3.2 问题模型

1. OARSMT 问题

如前所述，随着集成电路的特征尺寸不断缩小，许多可重复使用的组件(如宏块和 IP 核)现在可以在布线设计之前集成到芯片中。因此，这些组件应被视为障碍，不能在布线过程中穿过。因此，在 OARSMT 问题中，障碍物可以在几何上定义为任意大小的矩形。

在如图 3.2 所示的布线图中，OARSMT 问题的输入由一组引脚节点 $P=\{p_1,p_2,\cdots,p_m\}$ 和一组障碍物 $O=\{o_1,o_2,\cdots,o_k\}$ 构成。每个引脚 $p_i\in P$ 有一个对应的坐标位置 (x_i,y_i)。

请注意，p_i 不能位于任何障碍物内部，但它可以位于障碍物的边界上。一个障碍物 $o_j\in O$ 有两个对应的坐标 (x_{j1},y_{j1}) 和 (x_{j2},y_{j2})，这两个坐标分别表示障碍物的左下点和右上点。任何两个障碍物不能相互重叠，除非在边界点。目标是

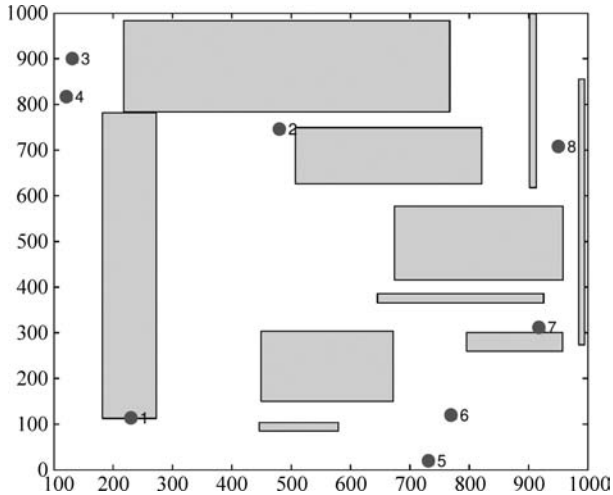


图 3.2　包含 8 个引脚和 10 个障碍物的芯片布线图

构建一个连接所有引脚节点的 OARST。在这样的树内，有一些额外的节点，即 Steiner 点，可以作为内部节点引入。树的边可以是水平的，也可以是垂直的。此外，树的边不能与任何障碍物有交集，但它可以相交于障碍物边界上或是障碍物的端点。因此，OARSMT 构建问题可表述如下。

在布线平面上给定一组引脚节点集合 P 和一组障碍物集合 O，构建一个 Steiner 树，从而连接 P 中所有给定的节点，只使用水平和垂直的边，且没有边与 O 中的任何障碍物相交，并使得树的总长度最小。

2. 绒泡菌计算的基本模型

在 PORA 使用的计算模型遵循绒泡菌的生物学机制。本节构建了一个自适应的管状网络 G 来模拟绒泡菌的觅食行为，其中 G 中的节点 v_i 代表食物源或内部节点，而 G 中的边 $e_{i,j}$ 代表可用于 v_i 和 v_j 之间原生质流运输的管状路径。此外，网络中的每个节点和每个路径分别与压力值和厚度值相关联。然后可以通过按照以下规则动态更新管状网络来模拟绒泡菌的寻路过程。

(1) 没有包含食物的叶节点的管状路径将逐渐被消除，即在寻路过程中，管状路径的厚度逐渐减小。

(2) 厚且短的管状路径对原生质流的输送更有效率(根据流体动力学理论)。

此外，为了模拟绒泡菌在觅食过程中的身体变化，管状路径的厚度和原生质流之间的反馈机制如下。

(1) 管状路径越厚，原生质流过的流量越大。

(2) 流量的增加会进一步增加管状路径的厚度。

(3) 在潜在的危险区域，管状路径的厚度将减小。

图 3.3 显示了一个管状网络，代表绒泡菌生物的初始形状，其中圆圈和方块分别代表食物源和内部节点。

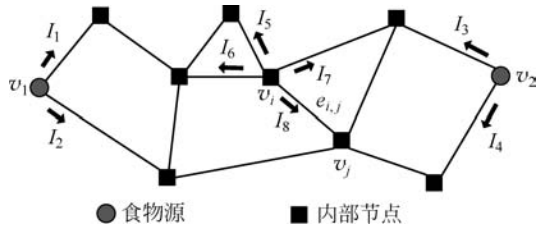


图 3.3 绒泡菌生物的初始管状网络

假设节点 v_i 处的压力是 pr_i，管状路径的原生质流是一种泊肃叶流，从 v_i 到 v_j 的路径 $e_{i,j}$ 的流量可以计算为

$$Q_{i,j}=\frac{\pi r_{i,j}^4}{8\tau}\cdot\frac{\mathrm{pr}_i-\mathrm{pr}_j}{l_{i,j}}=\frac{d_{i,j}}{l_{i,j}}(\mathrm{pr}_i-\mathrm{pr}_j) \tag{3.3}$$

其中，$l_{i,j}$ 和 $r_{i,j}$ 分别是管状路径 $e_{i,j}$ 的长度和半径。τ 是流体的黏度系数，$d_{i,j}=\pi r_{i,j}^4/8\tau$ 是路径 $e_{i,j}$ 的导电率的计算公式。由于管状路径的长度是一个常数，管状网络的进化主要由路径的导电率以及节点处的压力决定。

在绒泡菌计算的每个迭代步骤中，选择食物源作为源点 v_s，通过管状网络来发送原生质流。此外，网络中的另一个食物源作为汇点 v_k 来接收原生质流。相应地，可以根据基尔霍夫定律推导出以下方程：

$$\sum_{e_{i,j}}Q_{i,j}-I_0=0,\quad \text{若 } v_i=v_s \tag{3.4}$$

$$\sum_{e_{i,j}}Q_{i,j}+I_0=0,\quad \text{若 } v_i=v_k \tag{3.5}$$

$$\sum_{e_{i,j}}Q_{i,j}=0,\quad \text{若 } v_i\neq v_s \text{ 且 } v_i\neq v_k \tag{3.6}$$

其中，I_0 代表通过管状网络的总流量，即从源点流出，流入到汇点的流量。

以图 3.3 所示的管状网络为例。假设分别选择 v_1 和 v_2 作为源点和汇点，可以得到 $I_1+I_2=I_0$、$I_3+I_4=-I_0$、$I_5+I_6+I_7+I_8=0$。因此，节点压力的泊松方程以从式(3.4)～式(3.6)中导出。

通过设置 $\mathrm{pr}_k=0$ 来作为基本压力水平，可以通过求解式(3.7)来确定其他节点处的压力，并且可以通过式(3.3)来计算通过每个管状路径的流量。

$$\sum_{e_{i,j}}\frac{d_{i,j}}{l_{i,j}}(\mathrm{pr}_i-\mathrm{pr}_j)=\begin{cases}I_0, & \text{若 } v_i=v_s\\ 0, & \text{若 } v_i\neq v_s \text{ 且 } v_i\neq v_k\\ -I_0, & \text{若 } v_i=v_k\end{cases} \tag{3.7}$$

此外，如前所述，除了节点处的压力之外，管状网络的自适应行为也与管状路径的厚度密切相关，相应地，管状路径的厚度可以由管导电率和通过管的流量之间的反馈机制决定。

$$\frac{\mathrm{d}}{\mathrm{d}t}d_{i,j}=f(|Q_{i,j}|)-\delta d_{i,j} \tag{3.8}$$

其中，函数 $f(|Q_{i,j}|)$表示的是由流量引起的管道的扩张，通常被表述为具有单调递增性质的连续函数，同时满足 $f(0)=0$。比如函数 $f(\cdot)$可以定义为 $f(|Q_{i,j}|)=2\times|Q_{i,j}|$。式(3.8)右侧的第二项对应于管的收缩。参数 δ 是一个正实数，表示由危险引起的管道导电率下降率。

通过上述机制，网络中的管状路径将相互竞争有限的流量，使得那些非关键管状路径中的原生质流逐渐消失。经过一定次数的迭代后，管状网络中只剩下连接两个食物源的最短路径。

3.3.3　算法设计

本节提出了 PORA——受绒泡菌启发的用于 OARSMT 构建的绕障布线算法。图 3.4 显示了 PORA 的流程图。

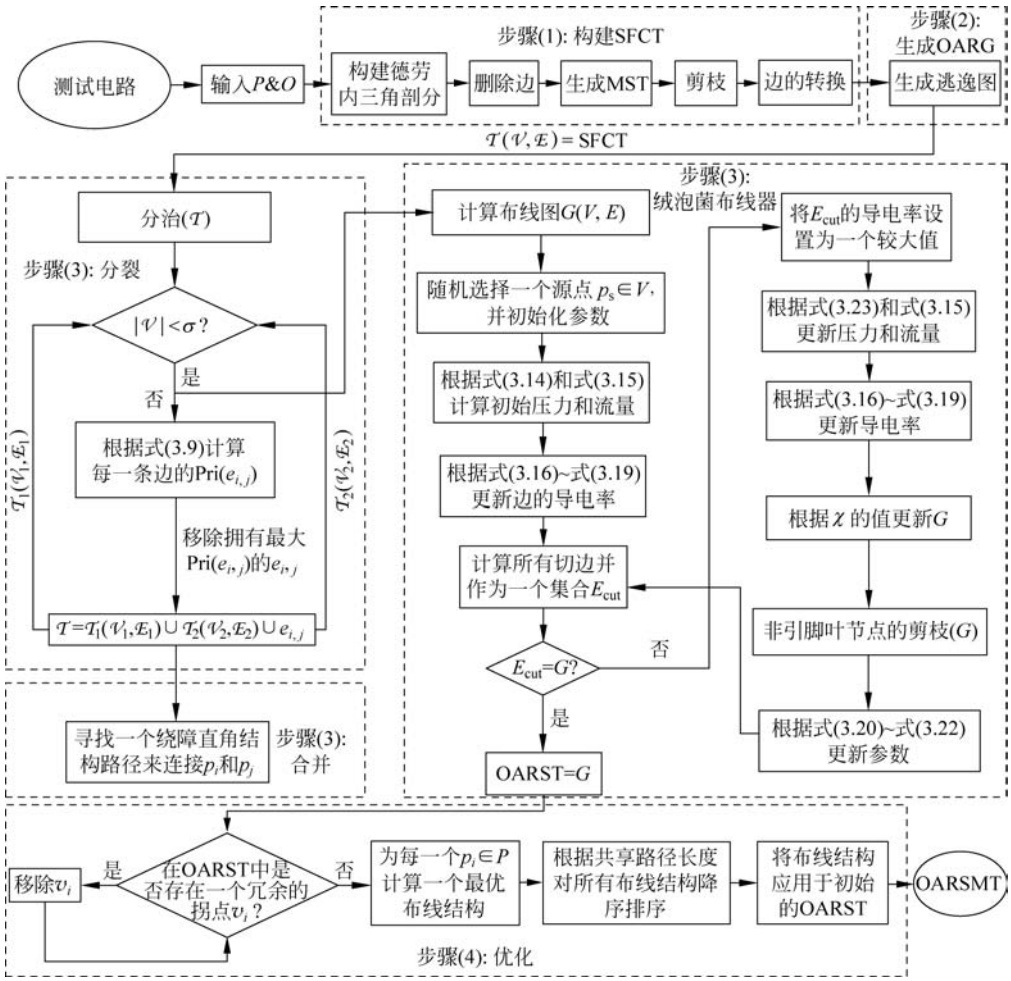


图 3.4　PORA 的流程图

该流程图包括 4 个主要步骤。

(1) 构建 SFCT。构建一个连接给定的引脚节点和拐点的 SFCT,进而对布线平面进行划分。

(2) 生成 OARG。OARG 连接所有引脚节点,同时构建障碍物,并将其作为基础图。

(3) 生成 OARST。提出了一种有效的分治策略指导下的绒泡菌布线器,用于在先前生成的 OARG 上构建 OARST。

(4) 优化。提出了两种改进技术,以进一步减少所构建的 OARST 的总线长。

1. 构建 SFCT

给定一个连通、加权的无向图,MST 是将所有节点连接在一起的边的子集,没有形成回路,并且具有最小的总边权重。由于构建 MST 比构建 Steiner 树容易得多,以往大多数关于集成电路布线设计的研究通常先构建一个 MST 作为布线网络的基本框架,然后将其转化为 Steiner 树。例如,在文献[48]中,首先构建一个最小终端生成树来连接所有的引脚节点,然后通过利用基于边的技术将其转换为 OARST。文献[79]提出了一种基于粒子群算法的策略来构建绕障最小 Steiner 树。在这个算法中,所有个体都被初始化为一个连接给定节点的 MST。然后,通过在布线平面中引入一些额外的点,将生成的最小生成树转换为 Steiner 树。相应地,在所提出的方法中,首先构建一个 SFCT 来连接给定的引脚节点和一些拐点,其可以定义如下。

定义 3.7 给定布线平面上的一组引脚节点和一组障碍物,如果下列条件成立,则树 T 被称为 Steiner 全连通树:

(1) T 通过一些拐点连接所有的引脚节点。

(2) T 中的所有叶节点都是引脚节点。

(3) T 中的所有拐点都是 Steiner 点。

如图 3.4 所示。SFCT 在 PORA 中有以下功能:

(1) 通过分而治之策略,有效地划分布线平面。

(2) 确定最终 OARSMT 的基本框架。

因此,在所提出的方法中,SFCT 的构建包括 5 个步骤,包括德劳内三角剖分(Delaunay Triangulation,DT)的构建、边的删除、生成 MST、剪枝和边的转换。

以图 3.5(a)所示的布局为例,说明上述过程。在一个点集中,由于一个德劳内三角剖分已被证明至少包含一个 MST,并且候选边只被受限于线性空间,首先构建一个连接所有引脚节点和拐点的德劳内三角剖分(见图 3.5(b)),然后删除穿过障碍物的边(见图 3.5(c))。接下来,采用经典的最小生成树算法,比如 Prim 算法或 Kruskal 算法,在线性时间内生成 MST(见图 3.5(d))。此外,在剪枝过程中,所有不是引脚的叶子节点以及连接到它们的边都从树中移除(见图 3.5(e))。最后,通过移除所有非 Steiner 拐点可以生成 SFCT(见图 3.5(e))。为了保证树的连通

性，一旦某个拐点被移除，相应的两个相邻节点将会连接。注意，因为图 3.5(f)中的 c_0 是 Steiner 点，所以它不会从树中移除。

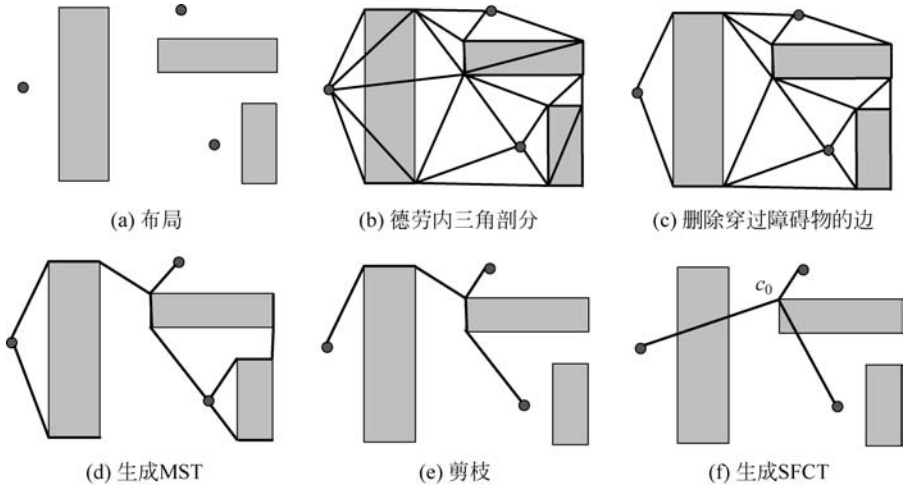


图 3.5　SFCT 的构建过程

2. 生成 OARG

在这一步中，将构建出来的 OARG 作为所提出的布线方法的基础图，其定义如下。

定义 3.8　给定一组引脚节点和一组障碍物，如果一个连接所有引脚节点和拐点的无向图的边都不与障碍物相交，则称该无向图为**绕障布线图**。

在提出的方法中，如图 3.6(a)中，将布线平面相对于每个引脚节点分成 4 个象限，并采用称为逃逸图的线投影技术生成一个 OARG。更具体地说，从 4 个正交方向上对所有引脚节点和拐角节点进行线段投影，当遇到组件或布线平面的边界时，投影线就中断。投影线的交点都是逃逸图中的点，点与点之间的线就是边。逃逸图构建消耗的时间为 $O(n^2)$，其中 n 是引脚节点和拐点的总数。更重要的是，已经证明了逃逸图至少包含一个最优的 OARSMT，并且得到 OARSMT 构建的候选边的时间复杂度只有 $O(n^2)$。

另一方面，为了进一步提高 OARG 构建的效率，采用了两种启发式策略来消除构建的逃逸图中的冗余点和冗余边：

(1) 线段在算法早期仅从引脚节点投影，然后再从阻挡了先前投影线的障碍物的拐点投影。

(2) 每个正交方向上的投影线长度，受到在两个相邻象限中的源点和引脚节点之间最短距离的限制。

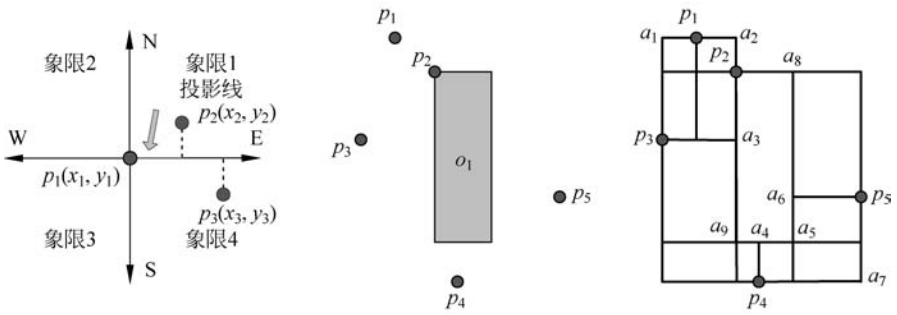
例如，在图 3.6(a)中，由于 $x_2-x_1<x_3-x_1$，所以源点 p_1 向东延伸的投影线被限制在了从 x_1 到 x_2 的区间内。

上述策略给 OARG 构建带来的优势如下：

(1) 忽略了对优化 OARG 没有影响的障碍。

(2) 从 OARG 中消除了对 OARSMT 构建没有作用的点和边。

例如，图 3.6(b)和图 3.6(c)分别显示了电路布局的一部分和对应的 OARG。



(a) 逃逸图的线投影　(b) 电路布局的一部分　(c) OARG

图 3.6　OARG 构建图

3. 生成 OARST

在完成 OARG 的构建之后，在这一步中，基于绒泡菌的觅食活动构建了绒泡菌计算的数学模型，从而给出了前面提到的绒泡菌布线器，它可以在生成的 OARG 上构建一个连接给定引脚节点的 OARST。此外，本节还提出了一种分治策略来有效地划分布线平面，从而系统地提高绒泡菌布线器的性能。

1) 分治策略

算法 3.3 给出了分治策略的伪代码。本节的算法通过不断地从先前生成的 SFCT 图中移除掉边，从而将 P 中的所有引脚节点划分为多个子集，并使用所提出的绒泡菌布线器为每个子集合构建 OARST。最后将这些 OARST 合并在一起，形成一个完整的连接所有引脚节点的 OARST。

首先，从 SFCT 中删除一条边，并生成两个子树。如果某个子树中的引脚节点数小于一个给定的阈值 σ，则直接构建一个连接这些引脚节点的 OARST；否则，生成的子树将被进一步划分为更小的树。最后，本节提出了一种多角度评估机制来选择 SFCT 的边，从而可以系统地提高分割过程的效率。

在更详细地讨论本节的方法之前，本节首先给出在所提出的评估机制中所使用的几个概念。

定义 3.9　给定一个节点集合 V，V 的**直角边界框**，用 RBB(V)表示，指覆盖 V 中所有引脚节点的最小矩形。RBB(V)用两个坐标(R_l(V)，R_b(V))和(R_r(V)，R_t(V))来表示，这两个坐标分别是矩形的左下角和右上角的位置。

算法 3.3　Divide - and - conquer($T(V,\varepsilon)$)
输入：一个 SFCT。

```
输出:一个 OARST。
初始化 α,β,γ,σ,S1 和 S2;
    if|V|≤σ then
      Physarum router(V);
      return;
    end
    else
        for 每一条边 e_{i,j}∈ε do
          根据式(3.9)计算 e_{i,j} 的优先值;
        end
        断开 T 中拥有最大优先值的边,并生成 T1(V1,ε1)和 T2(V2,ε2);
        Divide - and - conquer T1(V1,ε1);
        Divide - and - conquer T2(ε2,ε2);
        使用快速绕障策略合并 T1 和 T2 的 OARST;
    end
```

定义 3.10　$V=\{v_1,v_2,\cdots,v_h\}$是一个节点集合，c_r 是集合 V 的重心坐标，$\mathrm{dis}(v_i,c_r)$表示的是 v_i 和 c_r 之间的欧几里得距离，同时本节有以下定义：

(1) 如果某个节点 $v_i\in V$ 满足 $\mathrm{dis}(v_i,c_r)>\sum_{j=1}^{h}\mathrm{dis}(v_j,c_r)/h$，则该节点被称为**离群点**。

(2) 集合 V 的**聚合度**计算公式为 $1-h_0/h$，其中，h_0 表示的是集合 V 中离群点的个数。

基于上面的定义，给定需要被分成两个子树 $T_1(v_1,\varepsilon_1)$和 $T_2(v_2,\varepsilon_2)$的树 $T(v,\varepsilon)$，为 T 中的每个边计算优先值，并在具有最大优先值的边处断开树。边 $e_{i,j}\in\varepsilon$ 的优先值可以计算公式为

$$\mathrm{Pri}(e_{i,j})=\frac{\alpha\cdot A_1(i,j)+\beta\cdot A_2(i,j)+\gamma\cdot A_3(i,j)}{A_4(i,j)} \tag{3.9}$$

其中，α、β 和 γ 是 3 个权重系数，$A_1(i,j)\sim A_4(i,j)$是从不同角度设计的评估函数。这些函数解释如下。

$A_1(i,j)$用于评估 T 在边 $e_{i,j}$ 处断开时，T_1 和 T_2 之间的相对位置，包括图 3.7 中所示的情况。

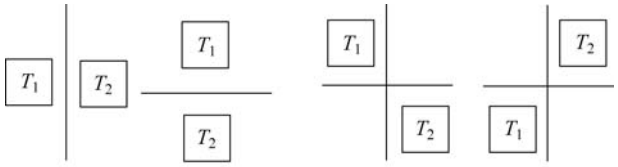
(1) 左右布局：$R_r(V_1)<R_l(V_2)$。

(2) 上下布局：$R_b(V_1)>R_t(V_2)$。

(3) 对角线布局：$R_r(V_1)<R_l(V_2)\wedge R_b(V_1)>R_t(V_2)$或$R_r(V_1)<R_l(V_2)\wedge R_t(V_1)<R_b(V_2)$。

$A_1(i,j)$的值会受以下规则影响：

(1) 当 T_1 和 T_2 形成对角布局时，通过合并两个子树的最优 RSMT 一定可以构建连接 V 中所有引脚节点的最优 RSMT。换句话说，T 在边 $e_{i,j}$ 处断开仍然可



(a) 左右布局与上下布局　　(b) 对角线布局

图 3.7　T_1 与 T_2 的相对位置

以保持解的最优性，$A_1(i,j)$因此被设置为相对较大的值。

(2) 当 T_1 和 T_2 形成上下或左右布局时，意味着如果 T 在边 $e_{i,j}$ 处断开，可能找不到连接 V 中引脚节点的最优 RSMT。$A_1(i,j)$因此被设置为相对较小的值。

故在所提出的方法中，$A_1(i,j)$计算如下。

$$A_1(i,j)=\begin{cases}10, & \text{若 } T_1 \text{ 和 } T_2 \text{ 形成一个对角线布局}\\ 5, & \text{若 } T_1 \text{ 和 } T_2 \text{ 形成上下布局 / 左右布局}\\ 0, & \text{否则}\end{cases} \tag{3.10}$$

$A_2(i,j)$用于计算 T 在边 $e_{i,j}$ 处断开时 T_1 和 T_2 的大小，即每个子树中包含的引脚节点个数。T 中引脚节点的平衡划分有助于提高分治策略的性能，$A_2(i,j)$计算公式如下：

$$A_2(i,j)=\frac{S_1}{||v_1|-|v_2|+1|} \tag{3.11}$$

其中，S_1 是常数，$|V_1|$和$|V_2|$分别是 T_1 和 T_2 的引脚节点数。

$A_3(i,j)$用于计算当 T 在边 $e_{i,j}$ 处断开时，T_1 和 T_2 中引脚节点的聚合度。当一组引脚节点零星分布在布线平面上时，这些引脚节点的聚集度相对较低，由于障碍物的阻挡，不得不在引脚节点之间构建折线路径。因此，必须利用大量的 Steiner 点并将其添加到构建的 OARST 中，从而增加了问题的复杂性。在提出的方法中，目标是找到一个子解，使得生成的子树中引脚节点的聚集度可以尽可能地提高。例如，将图 3.8(a)中的树分成两个子树，图 3.8(b)和图 3.8(c)分别通过在边 $e_{4,5}$ 和 $e_{5,6}$ 处断开树，得到两种不同的解。然而图 3.8(b)的结果不太令人满意，因为 p_5 和 p_8 能成为两个离群点，导致子树 T_1 中引脚节点的聚集度较低。相比之下，在图 3.8(c)中，两个子树中引脚节点的聚合度都比较高。因此，$A_3(i,j)$计算如下：

$$A_3(i,j)=\frac{S_2}{N_1+1}+\frac{S_2}{N_2+1} \tag{3.12}$$

其中，S_2 是常数，N_1 和 N_2 分别是 T_1 和 T_2 的离群点个数。

$A_4(i,j)$用于计算引脚节点 p_i 和 p_j 之间绕障路径的复杂性。如图 3.9 所示，对于边 $e_{i,j}$，有两种类型的连接 p_i 和 p_j 的 L 形路径。在构建 p_i 和 p_j 之间的

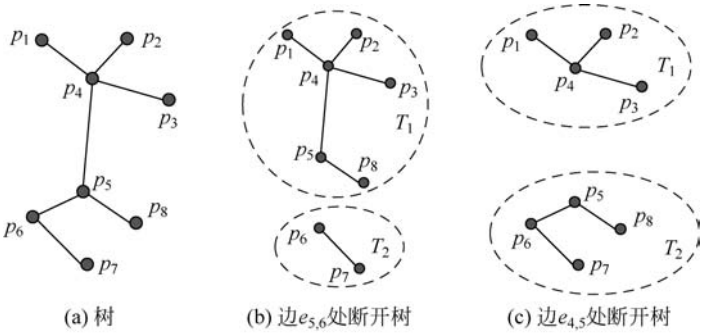


图 3.8　树分裂的样例

直角路径时，假设两条 L 形路径都被障碍物阻挡，则不可避免地需要一些中间节点/Steiner 点，从而增加了路径寻找的复杂性。因此，本节的算法倾向于在 L 形路径被障碍物阻挡的边上将树断开，然后使用快速绕障策略合并生成的 OARST。基于上述分析，$A_4(i,j)$计算如下：

$$A_4(i,j)=\begin{cases}1, & \text{若 } e_{i,j} \text{ 的两条 L 形路径均被障碍物阻挡}\\ 1.5, & \text{否则}\end{cases} \tag{3.13}$$

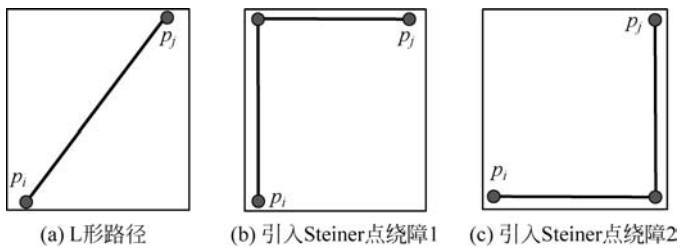


图 3.9　两点之间的 L 形路径

2）绒泡菌布线器

如上所述，在将给定的引脚节点划分成多个子集之后，将调用绒泡菌布线器为每个子集中的引脚节点构建一个 OARST(见算法 3.4)。

OARST 的构建基于之前生成的 OARG。然而，对于 SFCT 引脚节点的子集 V_s，在 OARST 构建期间没有必要搜索整个 OARG。绒泡菌布线器只需要考虑 OARG 的一个子图，满足其中至少包含一个连接 V_s 中引脚节点的最优 OARST 即可。以如图 3.6(c)所示的布局为例，假设需要找到一个连接引脚节点 p_1、p_2 和 p_3 的 OARST，那么绒泡菌布线器只需要搜索矩形 a_1-a_2-a_3-p_3 所覆盖的区域，即由 3 个引脚节点形成的直角边界框。需要注意的是，如果引脚节点的直角边界框完全被障碍物阻挡，那么也应该考虑这些被障碍物覆盖的区域。例如，在图 3.6(c)中，由于 p_3 和 p_5 形成的直角边界框被障碍物 o_1 阻挡，所以在构建连接两个引脚

节点的 OARST 时，应该相应地搜索 OARG 中被 o_1 覆盖的区域（在这种情况下，OARST 被简化为 p_3 和 p_5 之间的最短绕障直角路径）。基于以上分析，有以下定义。

定义 3.11 给定 SFCT 中引脚节点的子集 V_s，由 $G(V,E)$ 表示的 V_s 的布线图可以通过以下步骤生成：

(1) 将 G 初始化为通过将 RBB(V_s)映射到 OARG 上而获得的子图。

(2) 如果 RBB(V_s)完全被障碍物阻挡，则 G 将扩大到包括 OARG 中这些障碍物所覆盖的区域。

算法 3.4 Physarum router(V_s)
输入：一个 SFCT 中的顶点子集合 V_s。
输出：一个连接 V_s 中所有顶点的 OARST。

```
1   计算 V_s 的布线图 G(V,E);
2   随机选择一个顶点 p_s∈V_s 作为源点;
3   初始化 E 中的边的 d_{i,j}(0)值为一个较小值;
4   初始化 V_s - p_s 中的顶点的 pr_i(0)值为 0;
5   初始化参数 η,ξ 和 χ(1)的数值;
6   根据式(3.14)计算每一个顶点 v_i∈V 的 pr_i(0);
7   根据式(3.15)计算每一条边 e_{i,j}∈E 的 Q_{i,j}(0);
8   根据式(3.16)～式(3.19)计算每一条边 e_{i,j}∈E 的 d_{i,j}(1);
9   初始化 E_cut = ∅和 t = 1;
10  while E_cut ≠ E´do
11      设置 E_cut 中的边的导电率为一个较大值;
12      根据式(3.23)更新每一个顶点 v_i∈V 的 pr_i(t);
13      根据式(3.14)更新每一条边 e_{i,j}∈E 的 Q_{i,j}(t);
14      根据式(3.16)～式(3.19)计算每一条边 e_{i,j}∈E 的 d_{i,j}(t+1);
15      根据 χ(t)的值更新 G;
16      对 G 进行非引脚叶节点剪枝;
17      根据当前布线图 G 更新 E_cut;
18      根据式(3.20)～式(3.22)更新参数 η,ξ 和 χ;
19      t = t + 1;
20  end
```

在提出的绒泡菌布线器中，对于布线图 $G(V,E)$ 的一组引脚节点 V_s，使用 $\mathrm{pr}_i(t)$ 和 $d_{i,j}(t)$ 分别表示第 t 次迭代时，引脚节点 $v_i \in V$ 处的压力和边 $e_{i,j} \in E$ 的导电率。遵循 3.3.2 节中讨论的绒泡菌计算的基本模型，从 V_s 中随机选择一个用 p_s 表示的引脚节点作为源点，并让 V_s 中剩余的引脚节点作为汇点。此外，汇点处的压力被设置为 0 作为基本压力水平，且边的电导率被设置为一个较小值。假设源点的流量为 $(|V_s|-1)\times I_0$，流入每个汇点的流量为 I_0，每条边 $e_{i,j} \in E$ 的长度为 $\mathrm{len}(i,j)$，其中，$|V_s|$ 为 V_s 中的引脚节点的个数，在第 t 次迭代时 G 覆盖的区

域所对应的泊松方程如下。

$$\sum_{e_{i,j}} \frac{d_{i,j}(t)}{\mathrm{len}(i,j)} \times (\mathrm{pr}_i(t) - \mathrm{pr}_j(t)) = \begin{cases} (|V_s|-1) \times I_0, & \text{若 } v_i = p_s \\ -I_0, & \text{若 } v_i \in V_s - p_s \\ 0, & \text{若 } v_i \in V - V_s \end{cases} \tag{3.14}$$

通过计算上述方程组，能够获得各引脚节点 $v_i \in V$ 处的初始压力。相应地，在第 t 次迭代时通过每条边 $e_{i,j} \in E$ 的流量，由 $Q_{i,j}(t)$ 表示，可以计算为

$$Q_{i,j}(t) = \frac{d_{i,j}(t)}{\mathrm{len}(i,j)} \times (\mathrm{pr}_i(t) - \mathrm{pr}_j(t)) \tag{3.15}$$

此外，采用营养吸收/消耗模型来模拟绒泡菌的形态变化。每条边 $e_{i,j} \in E$ 的状态按照以下规则更新：

(1) 流体流经 $e_{i,j}$ 则为该边提供营养。

(2) 边 $e_{i,j}$ 通过消耗营养来进行流体输送。

(3) 边 $e_{i,j}$ 所包含的营养的变化会进一步影响边的导电率。

因此，在第 t 次迭代时由通过 $e_{i,j}$ 的流量所提供的营养，由 $N_{i,j}^{+}(t)$ 表示，可以计算为

$$N_{i,j}^{+}(t) = \eta \times \frac{|Q_{i,j}(t)|}{|Q_{i,j}(t)|+1} \tag{3.16}$$

其中，η 表示边的吸收因子，$|Q_{i,j}(t)|$ 是在第 t 次迭代时通过 $e_{i,j}$ 的流量的绝对值。此外，在第 t 次迭代中由边 $e_{i,j}$ 所消耗的营养，由 $N_{i,j}^{-}(t)$ 表示，计算如下：

$$N_{i,j}^{-}(t) = \xi \times \mathrm{len}(i,j) \times d_{i,j}(t) \tag{3.17}$$

其中，ξ 是边的消耗因子。相应地，在第 t 次迭代时通过边 $e_{i,j}$ 的流量对导电率产生的变化可以计算为

$$\frac{\mathrm{d}}{\mathrm{d}t} d_{i,j}(t) = N_{i,j}^{+}(t) - N_{i,j}^{-}(t) \tag{3.18}$$

最后，在第 $t+1$ 次迭代时，边 $e_{i,j}$ 的导电率可以计算为

$$d_{i,j}(t+1) = d_{i,j}(t) + \frac{\mathrm{d}}{\mathrm{d}t} d_{i,j}(t) \tag{3.19}$$

另一方面，在上述绒泡菌布线器的数学模型中，边缘导电率的差异在算法的早期通常很小，但是通过边的流量的差异相对较大。因此，营养吸收(即全局搜索)控制着绒泡菌的形态变化。因此，通过将式(3.16)中的 η 设置为相对较大的值并将式(3.17)中的 ξ 设置为相对较小的值，以此来增强绒泡菌布线器的搜索效率。相反，随着迭代次数的增加，边的导电率的差异逐渐增大，营养消耗(即局部搜索)在迭代的后期阶段控制着绒泡菌的形态变化。相应地，相对较大的 ξ 和较小的 η 会增强绒泡菌布线器的局部搜索能力。基于以上分析，提出了参数 η 和 ξ 的动态更新策略，其公式如下：

$$\eta = \eta_u - \frac{\eta_u - \eta_l}{sp} \times t \tag{3.20}$$

$$\xi = \xi_l + \frac{\xi_u - \xi_l}{sp} \times t \tag{3.21}$$

其中，$\eta_u(\xi_u)$和 $\eta_l(\xi_l)$分别是参数 η（ξ）的上界和下界，sp 是预先定义的区间距离，t 表示迭代次数。值得注意的是，在第一次迭代中，η 和 ξ 分别初始化为 η_u 和 ξ_l。

在计算第 $t+1$ 次迭代时所有边的导电率之后，从 G 中删去满足 $d_{i,j}(t+1) < \chi(t+1)$条件的边，其中 $\chi(t+1)$是第 $t+1$ 次迭代时的边导电率阈值。然后，可以通过从 V_s 中随机选择另一个源点来建立新的泊松方程，并且通过该方程可以相应地更新引脚节点处的压力和通过边的流量。绒泡菌布线器会不断更新布线图，直到生成一个连接 V_s 中引脚节点的 OARST。同样，由于迭代次数越多，边的导电率越大，为了加快算法的收敛速度，参数 χ 动态更新公式为

$$\chi(t+1) = \chi(t) + \Delta c \tag{3.22}$$

其中，Δc 是用户定义的常量。

3）加速策略

这里将两种加速方法结合到所提出的绒泡菌布线器中，其中包括非引脚叶节点剪枝和泊松近似法，从而可以进一步提高算法的搜索效率。

（1）非引脚叶节点剪枝：在 OARST 构建过程中，每次迭代删去导电率小于阈值的边后，可能会在布线图中加入一些非引脚叶节点，导致产生一些无法到达任何引脚节点的死路。如果这些冗余节点以及它们的边可以直接从布线图中删除，则绒泡菌布线器的效率可以相应地提高。为此，在每次迭代后计算布线图中引脚节点的度数，然后移除度数为 1 的非引脚节点以及连接到这些引脚节点的边。值得注意的是，从图中除去节点后，可能会生成新的非固定叶节点。因此，本节的算法不断地执行节点删除操作，直到所有剩余的叶节点都是引脚节点。

采用图 3.10 作为例子来说明上面的剪枝方法。可以看出，节点 v_1、v_4 和 v_5 以及连接它们的边都从图中删除了。请注意，v_5 是通过从图中删除 v_4 而得到的新的非引脚叶节点。

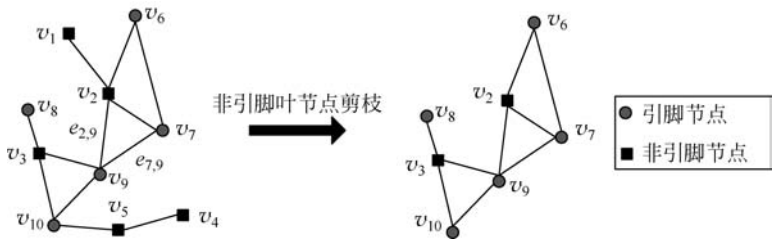


图 3.10　非引脚叶节点剪枝样例

(2) 泊松近似法：另一方面，文献[69]和文献[77]中的模拟结果表明，每个引脚节点的压力变化是连续的，并且在两次迭代之间变化相对较小。此外，绒泡菌布线器更注重整体的变化趋势和最终的 OARST 结构，而不是布线图的中间状态。因此，本节采用一种近似方案，将 $\mathrm{pr}_j(t)=\mathrm{pr}_j(t-1)$代入式(3.16)计算引脚节点压力。

$$\begin{cases}\displaystyle\sum_{e_{i,j}}\frac{d_{i,j}(t)\times(\mathrm{pr}_i(t)-\mathrm{pr}_j(t-1))}{\mathrm{len}(i,j)}=(|V_{\mathrm{s}}|-1)\times I_0, & \text{若 } v_i=p_{\mathrm{s}}\\ \mathrm{pr}_i(t)=0, & \text{若 } v_i\in V_{\mathrm{s}}-p_{\mathrm{s}}\\ \displaystyle\sum_{e_{i,j}}\frac{d_{i,j}(t)\times(\mathrm{pr}_i(t)-\mathrm{pr}_j(t-1))}{\mathrm{len}(i,j)}=0, & \text{若 } v_i\in V-V_{\mathrm{s}}\end{cases} \tag{3.23}$$

4) 绒泡菌布线器的终止条件

OARST 构建过程中需要考虑的另一个问题是：决定绒泡菌布线器的终止条件。这个问题对于生物启发算法来说非常重要，因为多余的迭代计算不仅降低了算法的效率，而且对解的质量没有显著的影响。然而，大多数以前的工作仍然使用预定义的最大迭代次数来控制算法的执行，因此，这可能导致解的质量降低或者运行时间的增加。

为了解决上述问题，以便能够更精确地控制绒泡菌布线器的执行，算法在 OARST 构建期间的每次迭代之后，计算布线图中的一组切边 E_{cut}。E_{cut} 对绒泡菌布线器有两方面的好处。

(1) 由于所提出的算法的目标是构建一个布线图，意味着一旦生成一个 OARST，布线图中剩余的所有边都应该是切边。相应地，一旦满足 $E_{\mathrm{cut}}=E_{\mathrm{r}}$ 的条件，就可以终止绒泡菌布线器的执行，其中，E_{r} 是在执行完边的删除操作之后，布线图中剩余的边的集合。

(2) 由于在 OARST 构建过程中，任何切边都不能从布线图中移除，否则布线图将被分成不连通的子图。因此，这些切边的导电率可以设置为相对较大的值，从而加速算法的收敛。

此外，可以从一系列实验中观察到，当几条具有相同代价的边连接到布线图中的同一个节点时，可能会产生无效解。到目前为止，以前大多数的研究都忽略了这个问题。如图 3.10 所示，假设图中的边 $e_{2,9}$ 和 $e_{7,9}$ 具有相同的长度，随着迭代次数的增加，两条边的导电率趋于相同的值。因此，假设两条边的导电率小于阈值，则能够同时从图中删除这两条边，从而产生不连通的图。为了解决这个问题，规定布线图中每个引脚节点的度在每次迭代中最多可以减少 1。这样，当从图中移除连接到相同节点且具有相同导电率的边时，至少一条边将被添加到集合 E_{cut} 中，从而确保布线图的连通性。

4. 优化

由于生成的 OARST 可能仍然包括一些次优的布线路径，所以在这一步中，实

现了两种优化技术来进一步优化 OARST 的结构，从而可以进一步减少总的线路长度。

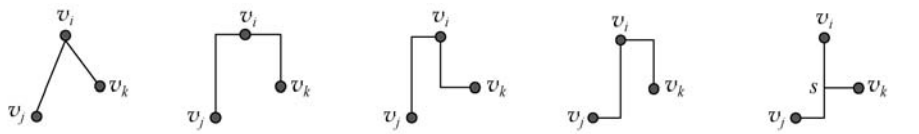
1）冗余弯曲消除

由于构建的 OARG 通常在每对引脚节点之间包含多条直线布线路径，这一特性可能为绒泡菌布线器提供更多的布线选择。然而，另一方面，一些冗余弯曲可能同时被加入到 OARST 中，因为绒泡菌布线器以随机方式选择引脚节点之间的布线路径。因此，提出了一种边合并技术来消除这些冗余点。对于所构建的 OARST 中的度数为 2 的非引脚节点 v_i，假设 v_j 和 v_k 是 v_i 的相邻节点，如果在 v_j 和 v_k 之间存在不与任何障碍物相交的 L 形路径（见图 3.9，每对引脚节点之间有两条可行的 L 形路径），且对应的线长不大于布线路径 $v_j \rightarrow v_i \rightarrow v_k$ 的线长，从 OARST 中删去 v_i，直接连接 v_j 和 v_k。重复执行上述步骤，直到 OARST 中没有冗余的弯曲。

2）Steiner 点重定位

此外，本节提出了另一种称为 Steiner 点重定位的技术，将 OARST 中的次优结构转换为最优结构。

例如，图 3.11(a)显示了一个在无障碍平面上的度数为 2 的引脚节点 v_i。图 3.11(b)～图 3.11(e)显示了 v_i 的所有可行布线路径组合。显然，图 3.11(e)中的路径组合是最优结构，因为 Steiner 点 s 的引入增加了公共路径的线长。在所提出的算法中，对于每个引脚节点 $p_i \in P$，假设 p_i 的度数为 w，枚举所有 2^w 种布线路径组合，选择线长最短的绕障结构作为 p_i 的布线结果。然后，根据公共路径的长度对所有布线路径组合进行降序排序，并将这些结构依次应用于原始的 OARST 中。值得注意的是，两个引脚节点之间的布线路径一旦在优化过程中确定，就不会改变，即使后来其他路径组合给出了不同的布线路径。



(a) 度数为2的v_i　(b) 可行布线路径1　(c) 可行布线路径2　(d) 可行布线路径3　(e) 可行布线路径4

图 3.11　Steiner 点重定位的描述

3.3.4　实验结果

所提出的 PORA 算法是用 C 语言实现的，并在一台具有 2.3GHz CPU 和 8GB 内存的 PC 上进行了测试。本节用 25 个基准电路来验证 PORA 的有效性，其中，rst01～rst10 是 RSMT 问题的基准电路，ind1～ind5 是来自 Synopsys 的用于 OARSMT 问题的工业电路，rc01～rc10 是绕障问题的基准电路。表 3.7 中的 Pin♯和 Obs♯分别代表基准电路中引脚节点和障碍物的数量。$w\%$ 计算公式为

(others-PORA)/others×100%。此外,本节中使用的参数设置如表 3.7 所示。

表 3.7　PORA 中的参数设置

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

1. 验证动态参数和加速策略

由于 PORA 可以应用于绕障和无障碍的布线问题,并且障碍物的存在只能增加 OARG 的规模,即得到一个有更多引脚节点和边的 OARG,直接在无障碍的基准测试问题上运行 PORA 来研究绒泡菌布线器的收敛性。这些电路中引脚节点的数量为 10～500。在实验中评估了 4 种不同的迭代策略。

(1) SP:仅使用一组静态参数的迭代策略。

(2) DP:仅使用动态参数的迭代策略。

(3) AM:仅使用加速方法的迭代策略。

(4) DP+AM:使用动态参数和加速方法的迭代策略。

图 3.12 展示了关于上述迭代策略的比较结果,其中横轴和纵轴分别表示电路中包含的引脚节点的数量和 PORA 执行的迭代次数。从图 3.12 可以得出以下结论:

(1) 加速方法和动态参数均可以提高绒泡菌布线器的执行效率(见图 3.12(a)和图 3.12(b))。此外,在加速算法方面,所提出的加速方法比动态参数更有效,主要是因为后者用于加强绒泡菌布线器的搜索能力,对算法收敛性的影响是间接的。

(2) 因为具有大量引脚节点的电路在每次迭代之后,通常会产生更多的非引脚叶节点和切边,所提出的加速方法的效果随着输入大小的增加而增强(见图 3.12(a))。

(3) 加速方式和动态参数互相兼容,两者结合可以进一步提高绒泡菌布线器的性能(见图 3.12(c)和图 3.12(d))。

2. 验证 RSMT 的构建

如上所述,RSMT 可以看作 OARSMT 的特例,而 PORA 可以为一组引脚节点生成 RSMT,而不需要任何修改。需要注意的是,如果输入电路只包含引脚节点,构建的 OARG 将退化为 Hanan 网格。

因此,下面首先将 PORA 与另一种生物启发的算法进行比较,即一种基于粒子群算法的 RSMT 构建方法。表 3.8 展示了比较结果。可以看出,在所有测试用例中,PORA 缩减的线长范围是－1.59%～2.44%,平均缩减率为 1.11%。分析

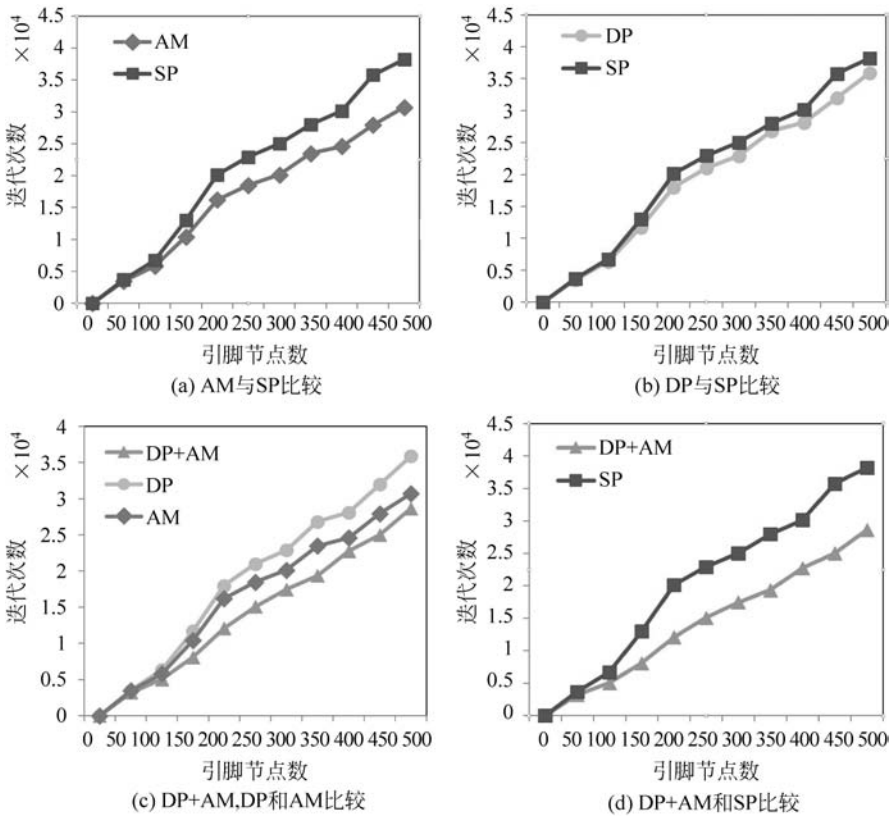


图 3.12 不同策略的比较结果

得知,有两个原因产生了这个结果:

(1) 由于构建出来的 MST 会作为 RSMT 的基本框架,这将最终的布线结构限制在相对较小的解空间内,从而导致一些 Steiner 点的位置不令人满意。因此,在大多数测试情况下,文献[44]中的布线结果比 PORA 的布线结果差。

(2) 在电路 rst02 和 rst09 中,尽管一些引脚节点零星分布在布线平面上,在先前构建的 MST 的帮助下,文献[44]的算法仍然可以找到连接这些引脚节点的一些最短路径。此外,遗传算子的加入,例如交叉和变异操作,进一步提高了粒子群策略的寻路效果。因此,就这两个测试样例而言,PORA 的结果稍差。另一方面,文献[49]提出了一种有效的绕障路线设计的启发式方法,该方法在不考虑障碍物的阻挡的情况下,可以生成连接一组引脚节点的 RSMT。然而,由于文献[49]中的方法首先构建连接引脚节点的 MST,然后将 MST 中的边直接转换成直线路径,候选 Steiner 点的位置被限制在非常小的空间内,产生了许多独立的布线路径。与文献[49]所提的方法相比,从表 3.8 可以看出,PORA 缩短的线长范围是−2.37%～3.19%,平均缩减率为 0.89%。

表 3.8　RSMT 构建的比较结果

基准测试问题	Pin#	Obs#	PORA	线长		Δw%	
				[44]	[49]	[44]	[49]
rst01	10	0	509	604	590	2.32	0.00
rst02	10	0	1983	1952	1937	−1.59	−2.37
rst03	50	0	46 224	46 997	46 533	1.64	0.66
rst04	50	0	52 548	53 660	54 280	2.07	3.19
rst05	80	0	43 551	44 071	44 762	1.18	2.71
rst06	80	0	72 351	73 246	72 476	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	23 421	24 007	23 820	2.44	1.68
平均值						1.11	0.89

表 3.9 中的第 2～6 行列出了 PORA、文献[44]和文献[49]所提出的 3 种在构建 RSMT 时 CPU 的运行时间。为了确保比较的公平性，从文献[44]和文献[49]的作者那里得到了相应的二进制文件，并在计算机上运行这两种算法。可以看到，PORA 有着很高的效率，并且在所有基准测试中的运行速度都高于文献[44]的方法。此外，文献[49]中的启发式算法的运行速度非常快，因为它采用了“一次性通过”的设计方式，但是在大多数测试用例中，构建出来的 RSMT 的质量是低于本算法的。

表 3.9　CPU 运行时间的比较

CPU 运行时间/s RSMT 构建(PORA/[44]/[49])				
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 构建(PORA/[49]/[79])				
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	2.25/0.00/0.02	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

3. 验证 OARSMT 的构建

本节将 PORA 算法与几种最先进的 OARSMT 算法进行了比较。表 3.10 展

示了比较结果。文献[47]提出了在λ几何布线平面的绕障算法，当λ设置为2时，它可以构建OARSMT。从表3.10可以看出，PORA在所有基准测试中的表现优于文献[47]的方法，平均线长减少了22.41%。显然，PORA的表现比文献[47]的方法好得多。这主要是因为文献[47]中的方法在构建OARSMT时引入了大量冗余的Steiner点，且OARSMT中大多数的布线路径是相互独立的。换句话说，由文献[47]的方法生成的布线路径很少彼此共享，从而产生较差的布线结果。与文献[48]中的方法(基于生成图的绕障布线算法)相比，PORA在基准测试中缩短的线长范围是−3.18%～5.00%，平均缩减率为1.72%。文献[49]提出了一种有效的四步绕障算法，它可以在直线和X结构中构建绕障最小Steiner树。从表3.10可以看出，在所有基准测试中，PORA缩短的线长范围是−0.65%～4.75%，平均缩减率为1.77%。分析得知，有两个原因产生了这样的结果：

(1) 由于文献[49]中的方法只从障碍物的边界选择Steiner点和拐点，相应的绕障路径被限制在有限的布线区域内，导致一些布线路径相对较长。相比之下，在本节所提出的方法中，生成的Steiner树中的中间节点是通过探索整个布线平面的设计空间来选择的，生成的Steiner点能有较为满意的位置。

(2) 在文献[49]方法中，最终的绕障最小Steiner树的结构主要由先前构建的MST来决定。相比之下，在所提出的方法中，最终的OARSMT的高层结构由SFCT决定。此外，引脚节点之间的详细布线由所提出的绒泡菌布线器来决定。换句话说，PORA为布线路径的构建提供了更大的灵活性，使得布线长度更短。

此外，将PORA算法与文献[79]中基于粒子群算法的混合布线方法进行了比较。考虑到文献[79]的方法允许在布线平面中使用对角线，为了确保比较的公平，将文献[79]方法生成的所有非直线路径替换为绕障直线路径。可以看出，PORA缩短的线长范围是−2.13%～3.24%，平均缩减率为1.16%，这进一步证明了所提出的绒泡菌布线器的强大优化能力。

表3.10中的第7～13行分别列出了PORA、文献[49]和文献[79]中方法在构建OARSMT时的CPU运行时间。表中没有列出文献[47]和文献[48]中方法的CPU运行时间，因为无法获得这两种算法的二进制文件/源代码。从表3.10中可以看出，文献[49]中方法的运行时间最短。此外，文献[49]中方法和PORA都可以在可接受的运行时间内为每个基准电路构建一个OARSMT。

表3.10 OARSMT构建的比较结果

基准测试问题	Pin#	Obs#	PORA	线长				Δw%			
				[47]	[48]	[49]	[79]	[47]	[48]	[49]	[79]
ind1	10	32	622	—	639	618	609	—	2.66	−0.65	−2.13
ind2	10	43	9500	—	10 000	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

续表

基准测试问题	Pin#	Obs#	PORA	线长				Δw%			
				[47]	[48]	[49]	[79]	[47]	[48]	[49]	[79]
rc01	10	10	26 334	30 410	27 540	27 630	27 015	13.40	4.38	4.69	2.52
rc02	30	10	42 462	45 640	41 930	43 290	43 882	6.96	−1.27	1.92	3.24
rc03	50	10	54 722	58 570	54 180	56 940	54 737	6.57	−1.00	3.90	0.03
rc04	70	10	60 925	63 340	59 050	61 990	60 800	3.81	−3.18	1.72	−0.21
rc05	100	10	75 146	83 150	75 630	75 685	75 685	9.63	0.64	0.71	0.71
rc06	100	500	84 030	149 725	86 381	84 662	85 808	43.9	2.72	0.75	2.07
rc07	200	500	113 056	181 470	117 093	113 598	113 672	37.7	3.45	0.48	0.54
rc08	200	800	118 277	202 741	122 306	119 177	122 057	41.7	3.29	0.76	3.10
rc09	200	1000	117 722	214 850	119 308	117 074	117 993	45.2	1.33	−0.55	0.23
rc10	500	100	167 781	198 010	167 978	167 219	169 443	15.3	0.12	−0.34	0.98
平均值								22.41	1.72	1.77	1.16

此外，图 3.13 展示了由 PORA 生成的两个工业电路布线图。可以看出，连接每对引脚节点的布线路径绕过了所有障碍物，同时使 OARSMT 的线长最小。

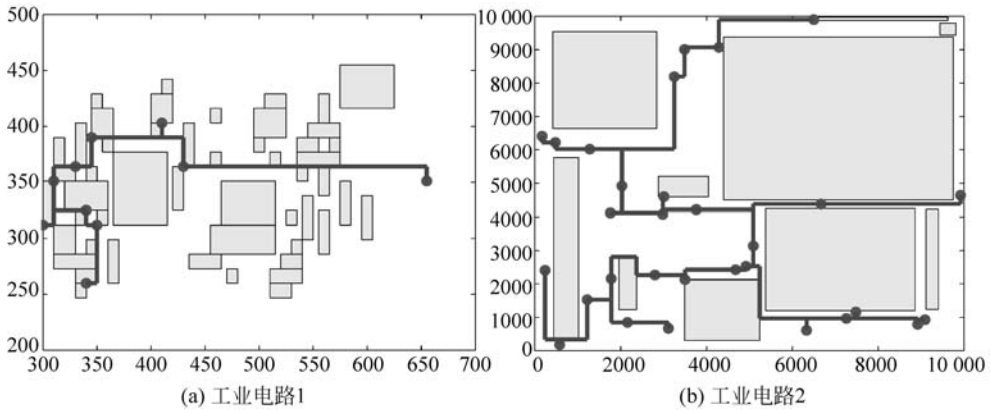


图 3.13　两个工业电路的布线结果

4. 关于受绒泡菌启发的 Steiner 树构建的讨论

如上所述，文献[77]还提出了一种基于绒泡菌生物学行为的 SMT 构建方法。因此，下面详细讨论 PORA 方法和文献[77]中的方法之间的差异，从而总结所提出的方法的主要优点。具体分析如下。

问题模型：文献[77]中的方法主要解决传统的 SMT 问题，以欧氏度量和直线度量构建 Steiner 树。相比之下，本节提出的问题是构建一个连接一组引脚节点的直线 Steiner 树，同时避免障碍物的拥塞，即 OARSMT 构建问题。由于传统的 SMT 问题即使不考虑障碍也已被证明是 NP-难的，障碍的存在会进一步增加问题的复杂性。换句话说，与文献[77]中的方法相比，PORA 具有更强的鲁棒性和更强

大的优化能力。

问题规模：在文献[77]中考虑的问题规模远远小于本问题中的规模。更准确地说，在所提出的方法中，除了有效的绒泡菌计算模型之外，还有几种启发式方法，包括分治策略、非引脚叶节点剪枝策略等，被结合到所提出的算法中，以从根本上提高绒泡菌计算的性能。此外，提出了两种优化技术，进一步有效地减少由绒泡菌布线器生成的绕障 Steiner 树的线长。所有上述特征使 PORA 能够应用于大规模优化问题。例如，在文献[77]中，每个测试用例中的引脚节点数小于 20，相比之下，在本实验中，总共使用了 25 个基准电路来测试 PORA 的性能。其中，最大的电路包含了 200 个引脚节点和 1000 个障碍物。换句话说，PORA 需要考虑的引脚节点数达到了 4200(在本问题模型中，每个障碍包括 4 个引脚节点)。

3.3.5 结论

本节研究了集成电路物理设计中的绕障布线问题，提出了一种被称为 PORA 的有效方法来系统地解决这个问题。对于给定的一组引脚节点和一组障碍物，通过数学模拟多头绒泡菌的生物行为，从而探索整个布线平面的设计空间，PORA 可以构建一棵 Steiner 树，以最小的总线长连接所有的引脚节点，同时避免障碍物的堵塞。此外，在所提出的绒泡菌模型中集成了若干启发式和动态参数策略，全方面提高 PORA 的性能。用来自工业界和学术界的多组基准测试问题来验证所提出方法的有效性。实验结果已经证实，与现有的几种启发式算法相比，PORA 算法可以得到更好的结果(平均减少线长 1.16%～22.41%)。这项工作首次将“绒泡菌计算”应用于集成电路的物理设计，为今后的研究提供了基础。未来将提出一个更有效的数学模型来模拟绒泡菌的生物行为。

3.4 本章总结

在现代 VLSI 设计中，在可布线区域内存在大量障碍物。在布线阶段，线网不能直接穿过障碍区域。基于此背景，提出了单层以及多层绕障直角结构 Steiner 最小树算法，主要的研究内容及创新之处如下：

(1) 为了应对在解决工程问题中，GSTP 问题规模较大，算法调用频率高，实时性要求高的挑战，本章在启发式算法基础上，通过引入改进策略，提出基于候选 Steiner 点的通用启发式算法框架 SPCF。实验表明，针对不同应用场景所提出来的求解质量以及运行时间的需求，可以利用不同的策略之间进行组合来实现两者之间的权衡。与现有的启发式算法相比较，SPCF 具有求解质量高、运行时间短的特点。

(2) 受到生物群体的智能行为的启发，研究发现多头绒泡菌在处理路径查找和网络构建问题中具有较好的性能。本章为了进一步挖掘绒泡菌算法的潜力，提

出了基于绒泡菌算法的绕障直角结构 Steiner 最小树算法 PORA。实验表明，在工业界以及学术界上多组测试问题中，相比现有的启发式算法，PORA 的表现较好。

参考文献

[1] Sherwani NA. *Algorithms For Vlsi Physical Design Automation*[M]. New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers, 2002.

[2] 徐宁，洪先龙. 超大规模集成电路物理设计理论与算法[M]. 北京：清华大学出版社，2009.

[3] Ganley JL, Cohoon JP. Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles[C]//Circuits and Systems, 1994 ISCAS '94, 1994 IEEE International Symposium on. 1994; 1: 113-116.

[4] Shen Z, Chu CCN, Ying-Meng L. Efficient rectilinear Steiner tree construction with rectilinearblockages[C]//Computer Design: VLSI in Computers and Processors, 2005 ICCD 2005 Proceedings 2005 IEEE International Conference on2005. p. 38-44.

[5] 朱祺，王垠，杨经洪. 考虑障碍的多端点最小直角 Steiner 树构造算法[J]. 计算机辅助设计与图形学学报，2005，17(2)：223-230.

[6] Wu P-C, Gao J-R, Wang T-C. A Fast and Stable Algorithm for Obstacle-Avoiding Rectilinear Steiner Minimal Tree Construction[C]//Proceedings of the 2007 Asia and South Pacific Design Automation Conference: IEEE Computer Society; 2007. p. 262-267.

[7] Li L, Young EFY. Obstacle-avoiding rectilinear Steiner tree construction[C]//Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. 2008, 523-528.

[8] Lin CW, Chen S-Y, Chi-Feng L, Yao-Wen C, Chia-Lin Y. Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2008, 27(4): 643-653.

[9] Long J, Zhou H, O. MS. EBOARST: An Efficient Edge-Based Obstacle-Avoiding Rectilinear Steiner Tree Construction Algorithm[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2008, 27(12): 2169-2182.

[10] 刘耿耿，王小溪，陈国龙，等. 求解 VLSI 布线问题的离散粒子群优化算法[J]. 计算机科学，2010，37(10)：197-201.

[11] Ajwani G, Chu C, Mak W-K. FOARS: FLUTE Based Obstacle-Avoiding Rectilinear Steiner Tree Construction[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2011, 30(2): 194-204.

[12] Tao H, Liang L, Young EFY. On the Construction of Optimal Obstacle-Avoiding Rectilinear Steiner Minimum Trees[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2011, 30(5): 718-731.

[13] Liu C-H, Kuo S-Y, Lee DT, Lin C-S, Weng J-H, Yuan S-Y. Obstacle-Avoiding Rectilinear Steiner Tree Construction: A Steiner-Point-Based Algorithm[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2012, 31(7): 1050-1060.

[14] Tao H, Young EFY. ObSteiner: An Exact Algorithm for the Construction of Rectilinear Steiner Minimum Trees in the Presence of Complex Rectilinear Obstacles[J]. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*. 2013, 32(6):

882-893.

[15] Chow W-K, Li L, Young EFY, Sham C-W. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach[J]. *Integration, the VLSI Journal*. 2014, 47(1): 105-114.

[16] Andrew B. Kahng, Jens Lienig, IgorL. Markov, Jin Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*[M]. Springer. 2011, 160-164.

[17] Karp R. Reducibility Among Combinatorial Problems[M]//Complexity of Computer Computations. Springer, Boston, MA, 1972: 85-103.

[18] Dreyfus SE, Wagner RA. The steiner problem in graphs[J]. *Networks*. 1972, 1(3): 195-207.

[19] Polzin T, Daneshmand SV. Improved algorithms for the Steiner problem in networks[J]. *Discrete Applied Mathematics*. 2001, 112(1-3): 263-300.

[20] Robins G, Zelikovsky A. Tighter Bounds for Graph Steiner Tree Approximation[J]. *SIAM J Discret Math*. 2005, 19(1): 122-134.

[21] Byrka J, Grandoni F, Rothvoss T, SanitA L. Steiner Tree Approximation via Iterative Randomized Rounding[J]. *J ACM*. 2013, 60(1): 1-33.

[22] Aragao M, Werneck R F. On the Implementation of MST-based Heuristics for the Steiner Problem in Graphs[C]// Revised Papers from the International Workshop on Algorithm Engineering & Experiments. Springer-Verlag, 2002, p. 1-15.

[23] Kou L, Markowsky G, Berman L. A fast algorithm for Steiner trees[J]. *Acta Informatica*. 1981, 15(2): 141-145.

[24] Takahashi H, Matsuyama A. An approximate solution for the Steiner problem in graphs [J]. *Math Japonica*. 1980, 6: 573-5777.

[25] Rayward-Smith VJ, Clare A. On finding steiner vertices[J]. *Networks*. 1986, 16(3): 283-294.

[26] Uchoa E, Werneck RF. Fast local search for the steiner problem in graphs[J]. *J Exp Algorithmics*. 2012, 17: 2.1-2.22.

[27] MP de Aragao CR, E Uchoa, RF Werneck. Hybrid Local Search for the Steiner Problem in Graphs[C]//Extended Abstracts of the 4th Metaheuristics International Conference. 2001: 429-433.

[28] Leung Y, Li G, Xu Z-B. A genetic algorithm for the multiple destination routing problems [J]. *Evolutionary Computation*, IEEE Transactions on. 1998, 2(4): 150-161.

[29] Wen-Liang Z, Jian H, Jun Z. A novel particle swarm optimization for the Steiner tree problem in graphs[C]//Evolutionary Computation, 2008 IEEE Congress on. 2008: 2460-2467.

[30] Qu R, Xu Y, Castro J, Landa-Silva D. Particle swarm optimization for the Steiner tree in graph and delay-constrained multicast routing problems[J]. *J Heuristics*. 2013, 19(2): 317-3142.

[31] 余燕平，仇佩亮. 一种改进的 Steiner 树启发式算法[J]. 通信学报，2002，23(11)：35-41.

[32] 李汉兵，喻建平，谢维信. 局部搜索最小路径费用算法[J]. 电子学报，2000，28(5)：92-95.

[33] 胡光岷，李乐民，安红岩. 最小代价多播生成树的快速算法[J]. 电子学报，2002，30(6)：880-882.

[34] 蒋廷耀,李庆华. 多播路由算法 MPH 的时间复杂度研究[J]. 电子学报,2004,32(10):1706-1708.

[35] 周灵,孙亚民. 基于 MPH 的时延约束 Steiner 树算法[J]. 计算机研究与发展,2008,5:810-816.

[36] 徐剑,倪宏,邓浩江,等. 改进的时延约束 Steiner 树算法[J]. 西安交通大学学报,2013,47(8):38-43.

[37] Koch T, Martin A, Vo S. *SteinLib: An Updated Library on Steiner Tree Problems in Graphs*[M]. Springer US,2001.

[38] Waxman BM, Imase M. Worst-case performance of Rayward-Smith's Steiner tree heuristic [J]. *Information Processing Letters*. 1988,29(6): 283-287.

[39] P. Yang, H. Yang, W. Qiu W et al. Optimal approach on net routing for VLSI physical design based on Tabu-ant colonies modeling[J]. *Appl. Soft Comput*. 2014,21: 376-381.

[40] C. Chu, Y. C. Wong, Fast and accurate rectilinear Steiner minimal tree algorithm for VLSI design[C]//Proceedings of the international symposium on Physical design, 2005, pp. 28-35.

[41] M. Hanan, On Steiner's problem with rectilinear distance[J]. *SIAM J. Appl. Math*. 1996, 14 (2): 255-265.

[42] M. Brazil, M. Zachariasen, Steiner trees for fixed orientation metrics[J]. *J. Global Optim*. 2009,43(1): 141.

[43] C. Chu, Y. C. Wong, FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design[J]. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*. 2009,27(1): 70-83.

[44] G. G. Liu, G. L. Chen, W. Z. Guo, Z. Chen, DPSO-based rectilinear Steiner minimal tree construction considering bend reduction[C]//Proceedings of International Conference on Natural Computation, 2011, pp. 1161-1165.

[45] C. H. Liu, S. Y. Yuan, S. Y. Kuo, S. C. Wang, High-performance obstacle-avoiding rectilinear steiner tree construction[J]. *ACM Trans. Des. Autom. Electron. Syst*. 2009, 14(3): 613-622.

[46] M. R. Garey, D. S. Johnson, The rectilinear Steiner tree problem is NP-complete[J]. *SIAM J. Appl. Math*. 1977,32(4): 826-834.

[47] T. T. Jing, Z. Feng, Y. Hu, et al.. λ-OAT: λ-geometry obstacle-avoiding tree construction with O(nlogn) complexity[J]. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*. 2007,26(11): 2073-2079.

[48] J. Y. Long, H. Zhou, S. O. Memik, EBOARST: an efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm[J]. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst*. 2008,27(12): 2169-2182.

[49] 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[J]. *ACM Trans. Des. Autom. Electron. Syst*. 2016,21(3): 1-30.

[50] L. Li, Z. C. Qian, E. F. Y. Young, Generation of optimal obstacle-avoiding rectilinear Steiner minimum tree[C]//Proceedings of International Conference on Computer-Aided Design-Digest of Technical Papers, 2009, 21-25.

[51] 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[J]. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2013,31(6): 882-893.

[52] C. K. Koh and P. H. Madden, Manhattan or non-Manhattan? A study of alternative VLSI routing architectures[C]//Proceedings of the ACM Great Lakes Symposium on VLSI, 2000,47-52.

[53] J. Luo, Q. Liu, Y. Yang et al.. An artificial bee colony algorithm for multi-objective optimisation[J]. *Appli. Soft Comput.* 2017,50: 235-251.

[54] J. Kennedy, Particle swarm optimization[C]//Proceedings of International Conference on Neural Networks, 1955, 1942-1948.

[55] Y. C. Chuang, C. T. Chen, C. Hwang. A simple and efficient real-coded genetic algorithm for constrained optimization[J]. *Appl. Soft Comput.* 2016,38: 87-105.

[56] T. Nakagaki, H. Yamada, A Toth, Intelligence: maze-solving by an amoeboid organism [J]. *Nature*. 407(28): 470-470.

[57] T. Nakagaki, M. Iima, T. Ueda, et al.. Minimum-risk path finding by an adaptive amoebal network[J]. *Phys. Rev. Lett.* 2007,99(6): 1-4.

[58] A. Adamatzky, Physarum machines: encapsulating reaction-diffusion to compute spanning tree[J]. *Naturwissenschaften*. 2007,94(12): 975-980.

[59] A. Tero, S. Takagi, T. Saigusa, et al.. Rules for biologically inspired adaptive network design[J]. *Science*. 2010,327(5964): 439-442.

[60] T. Atsushi, K. Ryo, N. Toshiyuki. A mathematical model for adaptive transport network in path finding by true slime mold[J]. *J. Theor. Biol.* 2007,244(4): 553-564.

[61] A. Adamatzky, Routing Physarum with repellents[J]. *Euro. Phys. Jour. E*, 2010,31(4): 403-410.

[62] S. Tsuda, J. Jones, A. Adamatzky et al.. Routing Physarum with electrical flow/current [J]. *Inter. Jour. Nanotech. Molec. Comput.* (IJNMC), 2011,3(2): 56-70.

[63] A. Adamatzky, Steering plasmodium with light: Dynamical programming of Physarum machine[J]. *arXiv preprint arXiv*. 2009.

[64] V. Evangelidis, J. Jones, N. Dourvas et al.. Physarum machines imitating a Roman road network: the 3D approach[J]. *Scient. Repor.*, 2017,7(1): 1-14.

[65] J. G. H. Whiting, R. Mayne, N. Moody et al. Practical circuits with physarum wires[J]. Biome. *Engin. Lett.*, 2016,6(2): 57-65.

[66] D. Schenz, Y. Shima, S. Kuroda et al.. A mathematical model for adaptive vein formation during exploratory migration of Physarum polycephalum: routing while scouting[J]. *Jour. Phys. D: Appl. Phys.*, 2017,50(43): 1-14.

[67] C. Gao, C. Yan, A. Adamatzky, Y. Deng. A bio-inspired algorithm for route selection in wireless sensor networks[J]. *IEEE Commu.* 2014, Lett. 18(11): 2019-2022.

[68] M. C. Zhang, C. Q. Xu, J. F. Guan, et al.. A novel Physarum-inspired routing protocol for wireless sensor networks[J]. *J. Distri. Sens. Netw.* 2013,9(6): 761-764.

[69] Y. N. Song, L. Liu, H. D. Ma, A. V. Vasilakos. A biology-based algorithm to minimal exposure problem of wireless sensor networks[J]. *IEEE Trans. Netw. Serv. Manag.* 2014,11(3): 417-430.

[70] K. Li, C. E. Torres, K. Thomas et al.. Slime mold inspired routing protocols for wireless sensor networks[J]. *Swarm Intelligence*, 2011, 5(3-4): 183-223.

[71] X. G. Zhang, S. Mahadevan, A bio-inspired approach to traffic network equilibrium assignment problem[J]. *IEEE Trans. Cybern.* 2018, 48(4): 1304-1315.

[72] M. A. I. Tsompanas, G. C. Sirakoulis, A. I. Adamatzky, Evolving transport networks with cellular automata models inspired by slime mould[J]. *IEEE Trans. Cybern.* 2015, 45(9): 1887-1899.

[73] H. Yang, M. Richard, Y. Deng. A bio-inspired network design method for intelligent transportation[J]. *Inter. Jour. Unconventional Comput.* 2019, 14(3/4): 199-215.

[74] H. Yang, Y. Deng, J. Jones. Network division method based on cellular growth and physarum inspired network adaptation[J]. *Inter. Jour. Unconventional Comput.* 2018, 13(6): 477-491.

[75] H. Yang, Q. Wan, Y. Deng. A bio-inspired optimal network division method[J]. *Physica A*. 2019, 527: 121259.

[76] 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[J]. *Inter. Jour. Produc. Reser.* 2017, 55(1): 244-263.

[77] L. Liu, Y. N. Song, H. Y. Zhang, et al.. Physarum optimization: a biology-inspired algorithm for the Steiner tree problem in networks[J]. *IEEE Trans. on Comput.* 2015, 64(3): 818-831.

[78] M. Caleffi, I. P. Akyildi, L. Paura. On the solution of the Steiner tree NP-hard problem via Physarum bionetwork[J]. *IEEE/ACM Trans. Netw.* 2015, 23(4): 1092-1106.

[79] 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[J]. *ACM Trans. Des. Autom. Electron. Syst.* 2015, 20(2): 1-28.

[80] S. Pettie, V. Ramachandran. An optimal minimum spanning tree algorithm[J]. *Jour. of ACM*. 2002, 49(1): 16-34.

[81] D. T. Lee, B. J. Schachter. Two algorithms for constructing a Delaunay triangulation[J]. *Inter. Jour. of Comput. Infor. Sci.* 1980, 9(3): 219-242.

[82] J. L. Ganley, J. P. Cohoon, Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles[C]//Proceedings of IEEE International Symposium on Circuits and Systems, 1994, pp. 113-116.

[83] W. K. Chow, L. Li, E. F. Y. Young, C. W. Sham. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach[J]. *Integ., the VLSI J.* 2014, 47(1): 105-114.