第 5 章

单层绕障 X 结构 Steiner 最小树算法

5.1 引言

单层绕障 Steiner 最小树(Obstacle Avoiding Steiner Minimum Tree, OASMT)的构建方法主要包含 4 类：先构造再替换法、不确定性算法、基于生成图的方法、精确算法。先构造再替换法是指先构造连接布线端点的 Steiner 最小树而不考虑障碍物，之后再将穿过障碍物的边替换为经过障碍物边界的布线边；不确定性算法采用元启发式策略；基于生成图的方法一般能够将引脚端点以及部分障碍物端点包含在生成图中从而降低求解空间的复杂度；精确算法能为绕障布线得到准确的方案。

为求解线长优化能力更强也更具难度的非曼哈顿结构单层绕障布线问题，5.2 节～5.4 节分别介绍了 4 种有效的单层绕障 X 结构 Steiner 最小树构造算法。

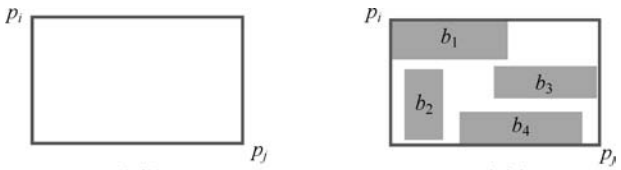
单层绕障 X 结构 Steiner 最小树(Obstacle Avoiding X-architecture Steiner Minimum Tree, OAXSMT)问题的数学模型可以简化如下：设 $P=\{P_1,P_2,P_3,\cdots,P_n\}$是线网中的一组引脚，其中每个 P_i 被赋予其坐标(x_i,y_i)。设 $B=\{B_1,B_2,B_3,\cdots,B_m\}$是芯片上的一组障碍物，其中每个 B_i 与两个坐标(x_{i1},y_{i1})和(x_{i2},y_{i2})相关联。(x_{i1},y_{i1})和(x_{i2},y_{i2})分别是矩形障碍物的左下角和右上角的坐标。关键的问题是构建一个 Steiner 树，将所有给定的引脚用 0°、45°、90°和 135°的线连接起来，绕开所有给定的障碍物并且使成本最小化。应该注意，树可以包括一些 pseudo-Steiner 点。

关于布线结构的 λ-几何学理论的具体定义已由 1.2.1 节的定义 1.2 给出。

定义 5.1 障碍物 在单层绕障 X 结构 Steiner 最小树问题中，障碍物可以是任意尺寸的矩形。任意两个障碍物不能互相重叠，但可以点与点接触或是边与边接触。

定义 5.2 引脚 引脚端点可以是在布线区域内的任意端点，不能处于任意障碍内，但可以分布在障碍的边上或是障碍角点处。

定义 5.3　边界框　两个端点 p_i 和 p_j 的边界框是由 p_i 和 p_j 形成的矩形。一组障碍的边界框是完全包含所有障碍物的最小矩形。图 5.1 显示了两个实例。



(a) 实例一　(b) 实例二

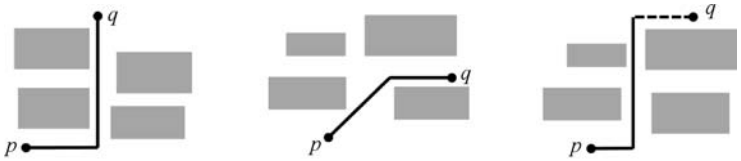
图 5.1　边界框

定义 5.4　矩形连接模型(Rec-Connected Model)　在布线模型中,如果一个引脚可以通过拐弯不超过一次连接到另一个引脚,并且布线选择仅限于选择 2 或选择 3,将其称为矩形连接模型。如图 5.2(a)所示,引脚 p 可以通过布线区域中的一次选择 3 与引脚 q 连接。

定义 5.5　八角连接模型(Oct-Connected Model)　在布线模型中,如果一个引脚可以通过拐弯不超过一次连接到另一个引脚,并且布线选择仅限于选择 0 或选择 1,将其称为八角连接模型。如图 5.2(b)所示,引脚 p 可以通过布线区域中的一次选择 1 与引脚 q 连接。

定义 5.6　完全连接模型(Fully Connected Model)　如果由两个引脚形成的布线模型既是矩形连接模型,也是八角连接模型,将其称为完全连接模型。

定义 5.7　断开连接模型(Disconnected Model)　障碍物的存在可能会阻止某些点被选为 pseudo-Steiner 点。如果一个引脚可以通过拐弯至少两次或更多次来连接到另一个引脚,如图 5.2(c)所示,则它形成断开连接模型。



(a) 矩形连接模型　(b) 八角连接模型　(c) 断开连接模型

图 5.2　3 种连接模型

5.2　基于离散粒子群优化的 X 结构绕障 Steiner 最小树算法

绕障 X 结构 Steiner 最小树问题(Obstacle Avoiding Octagonal Steiner Minimum Tree, OAOSMT)是构造一棵将布线区域上所有给定点及 Steiner 点连接起来的树,同时以最小代价绕过所有障碍。为了解决这个问题,第一步是构造 X 结构 Steiner 最小树。文献[2]提出了一种时间复杂度为 $O(|V|+|E|)$的算法来

构造一个与给定的直角结构 Steiner 树同构的 X 结构 Steiner 最小树。其中，$|V|$ 和 $|E|$ 是给定树的顶点和边。在绕障方面，2008 年文献[3]提出了一种通过修正构造的方法来解决绕障直角 Steiner 树（Obstacle Avoiding Rectilinear Steiner Minimum Tree，OARSMT）构造问题，它构建了一个障碍加权生成树在逃逸图上构造 OARSMT。文献[4]提出了一种基于迷宫布线的启发式方法来解决 OARSMT 问题，该方法基于迷宫布线的搜索过程，在解决方案质量和内存空间使用方面都能很好地处理多端线网。2009 年，文献[5]的算法改进并扩展了 GeoSteiner 方法，允许在布线区域内出现直线障碍。该算法可以在合理的时间内生成最优解。与以往的方法不同，文献[6]提出了一种新的算法，首先确定所有引脚的连接顺序，然后用贪心启发式算法迭代连接相邻的两个引脚。文献[7]应用计算几何技术开发了一种高效的算法，该算法可以获得质量优良的解，与之前已知的结果相比，速度有显著提高。然而，上述方法都是基于曼哈顿 Steiner 树，在布线长度上不能得到很好的优化。

作为一种基于群体的进化方法。由 Eberhart 和 Kennedy 在 1995 年提出的粒子群优化算法（Particle Swarm Optimization，PSO）被证明是一种全局优化算法。PSO 具有收敛快、全局搜索、稳定、高效等诸多优点。大量的实践证明，与其他方法相比，粒子群算法能够在全局优化问题上获得代价更低的最优解。

然而，PSO 的初始原型用于求解连续优化问题，但自 PSO 算法提出以来，已有许多工作尝试使用 PSO 来求解离散问题。例如，文献[9]提出了一种求解 TSP 的离散粒子群优化（Discrete Particle Swarm Optimization，DPSO）算法。文献[10]提出了一种基于 DPSO 的 X 结构 Steiner 最小树（Octagonal Steiner Minimal Tree，OSMT）算法来优化线长。

本节提出了一种有效的 OAOSMT 构造算法。它包括以下 5 个步骤。

（1）初始化。在该步骤中，有效地生成粒子群，其中每个粒子是连接所有给定引脚的 OSMT。

（2）预处理。在该步骤中，生成包含任何两个引脚的所有连接信息的表，为后续步骤提供快速的信息查询。

（3）飞行。在此步骤中，通过引入两个新的基于并查集的遗传算子来更新每个粒子。

（4）调整。在该步骤中，如果需要，调整群体的最终全局最优粒子 g_f 以确保所有边绕开所有障碍。

（5）精炼。在该步骤中，应用精炼的方法，进一步提高了最终全局最优粒子 g_f 的质量。此外，采用后续处理方法来确保可用于当前的制造技术。

5.2.1 算法细节

1. 初始化

为每个引脚和障碍提供唯一的序列号。然后，使用一组最小生成树边来表示

候选布线树，并向每个边添加一个额外变量，表示 pseudo-Steiner 点的布线选择，以便随后将其转换为 X 结构 Steiner 树。每个 pseudo-Steiner 点的布线选择包括 4 种类型，如定义 2.7～2.10(见 2.5.2 节)所示。如果网中有 n 个引脚，则最小生成树将包括 $n-1$ 个边，$n-1$ 个 pseudo-Steiner 点，以及一个额外变量，即粒子的适应度值(见式(5.1))。因此，一个粒子的长度为 $3(n-1)+1$。例如，如图 5.3 所示，粒子可以表示为以下数字串：1 3 2 2 3 0 3 4 1 3 5 3 27.3，其中 23.7 代表粒子的适应度值。前 3 个数字 1 3 2 表示一条边，即根据选择 2 来连接引脚 1 和引脚 3。

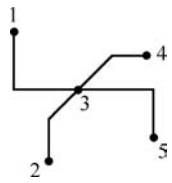


图 5.3　X 结构 Steiner 树

因为步骤 1 基于最小生成树(Minimum Spanning Tree，MST)算法，所以可以通过使用一些基于图形的方法(例如，Prim 算法或 Kruskal 算法)或更高效的基于 Delaunay 三角剖分的算法来轻松获得 X 结构 MST。应当注意，每条边的初始布线方向是从 4 个给定布线选择中随机选择的，因此可以穿过障碍物。

2. 预处理

根据 PSO 算法的基本模型，在初始化粒子群后，每个粒子将通过更新其速度和位置来移动，并且整个群体将相互合作以搜索期望的目标。但是，在介绍该粒子更新方法之前，首先描述预处理策略，主要是由于以下 3 个问题。

第一，在粒子飞行步骤中，当评估刚刚在运动之后生成的粒子时，除了计算每条边的长度之外，还应检查每条边是否避开所有障碍物。然后赋予每个粒子一个适当的适应度值，算法可以选择正确的全局最优粒子 g_i 和个体最优粒子 p_i。

第二，在操作调整方法之前，必须确定最终全局最优粒子 g_f 的每条边是否绕开所有障碍物。应该决定是否有必要进行调整。如有必要，应该调整哪些边，以及哪些障碍物已经被穿过？应计算所有这些信息。

第三，在精炼过程中，不断改变某些边的路径方向，以尽可能地增加公共边的长度。但是，怎样才能确保调整后的边能够避开所有障碍物？此信息也需要计算。

由于现代芯片的密度急剧增加，问题的规模变得越来越大。如果算法每次计算前面提到的信息，无疑会对整个算法的性能产生负面影响。因此，提出了一种预处理策略，该策略可以通过生成预先计算的查找表来为步骤 3～步骤 5 提供快速信息查询。

预处理策略的想法简单易行。令 $P=\{P_1,P_2,P_3,\cdots,P_n\}$ 为网络中的一组引脚，并且 $B=\{B_1,B_2,B_3,\cdots,B_m\}$ 芯片上的一组障碍物。对于每对引脚，例如 p_i 和 p_j，计算边 $p_ip_j(c)$ 穿过的障碍物的数量，并记录这些障碍物为集合 $\{B_k\}$，其中 c 表示 p_i 和 p_j 之间的布线选择。所有可能的边将构成最终的查找表。图 5.4 显示了预处理原理。线网中有 5 个引脚(1～5)，芯片上有 6 个障碍物(b_1～b_6)。以引脚 1 为例，图 5.4(a)显示了引脚 1 与基于选择 2 和选择 3 的其他引脚的连接图，而图 5.4(b)包括选择 0 和选择 1。可以看出，一些线段穿过障碍物而其他线则没有。

基于图 5.4,可以容易地生成引脚 1 的记录。如表 5.1 所示的引脚的连接信息,很容易查询关于引脚 1 的所有连接信息。例如,当引脚 1 基于选择 1 连接到引脚 4 时,边穿过障碍物 b_3 和障碍物 b_4。但是,当引脚 1 根据选择 2 连接到引脚 3 时,就避开了所有障碍物。

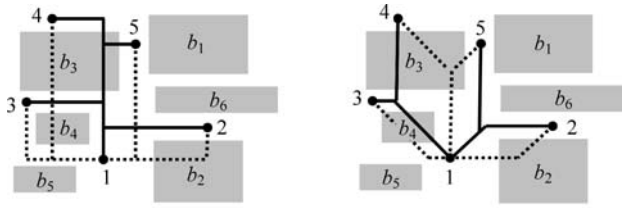


图 5.4 预处理原理

表 5.1 引脚 1 的连接信息

引 脚	布线选择			
	0	1	2	3
2	b_2	—	—	b_2
3	b_4	b_4	—	—
4	b_3	b_3,b_4	b_3	b_3,b_4
5	b_3	—	b_3	—

表 5.2 显示了具有 2.9GHz CPU 和 2GB 内存的 PC 中的查找表生成时间。Pin♯和 Obs♯分别代表引脚和障碍物的数量。因为只需要为每组布线数据生成一次查找表,所以在实现实验时只使用传统的数据结构,例如数组。在生成边记录时扫描每个障碍物 b_k,如果没有 b_k 的角点位于两个针脚的矩形边界框中,那么直接检查下一个障碍物 b_{k+1}。此外,45°和 135°段的存在无疑会增加问题的复杂性。当输入的比例很大时,可能需要几秒钟,但这是可以接受的。

表 5.2 查找表生成时间

引脚数/障碍物数	CPU 时间	引脚数/障碍物数	CPU 时间
10/32	10/43	10/50	25/79
0.000s	0.000s	0.000s	0.000s
33/71	10/10	30/10	50/10
0.000s	0.000s	0.000s	0.000s
70/10	100/10	100/500	200/500
0.000s	0.000s	0.140s	0.578s
200/800	200/1000	500/100	1000/100
0.889s	1.139s	0.795s	3.073s

3. 飞行

1) 粒子的适应度函数

在 PSO 算法的基本模型中，适应度函数是最重要的组件之一。p_i 和 g_i 的正确性直接影响适应度函数的准确性。因为优化的目标是线长，所以很明显使用 OSMT 的总长度作为一个粒子的适应度值是一个很好的选择；它可以直接反映一个部分的卓越程度。事实上，在定义的 OSMT 模型中，有 3 种边，可以定义如下。给定一个布线树 Ts 和两个引脚 p 和 q，有以下内容。

定义 5.8　Oseg(p,q)　表示 p 和 q 之间的边，其不穿过布线平面中的任何障碍物。

定义 5.9　Dseg(p,q)　表示穿过一个或多个障碍物的 p 和 q 之间的边，并且可以仅通过改变 p 和 q 之间的布线选择来绕开它们。

定义 5.10　Cseg(p,q)　表示穿过一个或多个障碍物的 p 和 q 之间的边，但是必须从这些障碍物中选择一些角点以绕开障碍物。

实际上，Cseg(p,q)是断开连接模型，Oseg(p,q)和 Dseg(p,q)可能是 3 个模型中的任何一个。如图 5.5(a)所示，A 和 B 之间的边属于 Oseg(A,B)，A 和 D 之间的边属于 Dseg(A,D)，因为它只能通过改变布线选择来绕开障碍物[即从选择 3 改为选择 1(见图 5.5(b)]。A 和 C 之间的边属于 Cseg(A,C)，因为必须选择障碍物的角点 C_1 作为中间节点(见图 5.5(b))。应该注意，AC_1 和 CC_1 之间可以有其他布线选择。

定义 5.11　ADseg(p,q)　表示在改变 p 和 q 之间的布线选择之后从 Dseg(p,q)获得的 p 和 q 之间的边，并且避免了所有障碍物。

定义 5.12　ACseg(p,q)　表示在添加障碍物的一个或多个角点之后从 Cseg(p,q)获得的 p 和 q 之间的一组边，并且避免了所有障碍物。

在图 5.5(b)中，A 和 D 之间的边属于 ADseg(A,D)。另外，A 和 C 之间的两条边(AC 和 $CC1$)属于 ACseg(A,C)。假设 $L(r)$ 等于 r 的总长度。在前面的定义的基础上，适应度函数如下。

$$\text{Fitness}(P)=\Big(\sum_{\text{Oseg}(p,q)} L(\text{Oseg}(p,q))+\sum_{\text{Dseg}(p,q)} L(\text{Dseg}(p,q))+\sum_{\text{Cseg}(p,q)} L(\text{Cseg}(p,q))\Big) \tag{5.1}$$

在式(5.1)中，可以通过查找表容易地识别 Oseg(p,q)、Dseg(p,q)、ADseg(p,q)和 Cseg(p,q)。对于 ACseg(p,q)，其计算类似于调整过程，但只是计算 L(ACseg(p,q))的结果而不是实现实际的加法和删除操作。

2) 粒子的更新方法

算法 5.1 显示了粒子更新的伪代码。本节算法具体的更新公式与 4.2.2 节介绍的更新方式类似，此处不再赘述。

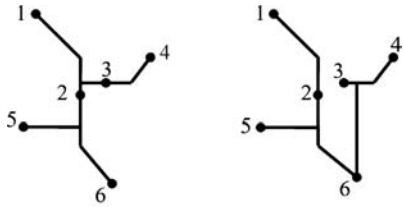
引理 5.1　假设 Cp 是粒子 p 的当前位置，Ap 是更新后的位置。Fp 是 p 正在

(a) Oseg(A, B)和Dseg(A, D)以及Cseg(A, C)　　(b) ADseg(A, D)和ACseg(A, C)

图 5.5　3 种边对应绕障方法

寻找的目标，dis(t)表示位置 t 和 Fp 之间的距离。可能存在 dis(Ap)＞dis(Cp)。

证明：很明显，因为更新方法以随机模式操作，所以可能导致 p 的一些好因素被一些意外因素(线长不减小，反而增加，因为 PSO 是随机算法)所取代。如图 5.6 的粒子更新示意图所示，当更新 p 的速度(见图 5.6(a))时，如果选择边 2 3 2 并由边 3 6 3 替换，则将获得图 5.6(b)。显然，dis(b)＞ dis(a)。



(a) 粒子更新前　　(b) 粒子更新后

图 5.6　粒子更新示意图

算法 5.1　粒子的更新方法

输入：惯性权重 w，加速因子 c_1 和 c_2，粒子 p，全局(个体) 最优粒子 $g_i(p_i)$

输出：新的粒子 p

```
for each particle p in Swarm do
    Generates r1 = Random(0,1);
    if (r1 < w) then
        mutation operator(p);//变异操作
    end if
    Generates r2 = Random(0,1);
    if(r2 < c1) then
        crossover operator(p, pi);//交叉操作
    end if
    Generates r3 = Random(0,1);
    if(r3 < c2) then
```

```
    crossover operator(p, g_i);//交叉操作
  end if
  Update(p_i);//更新个体最优粒子 p_i
end for
Update(g_i);//更新全局最优粒子 g_i
```

4. 调整

在粒子飞行结束后，选择全局最佳粒子 g_f 作为最终的 X 结构布线树。粒子 g_f 可能非常优秀，直接避开了所有障碍物。当然，它可能也很糟糕，穿过了许多障碍物。对于后者，当粒子的边 pq 穿过障碍物时，只有两种可能的场景：Dseg(p,q)和 Cseg(p,q)。两者都可以通过表查找直接识别，第一个也可以通过查找表来调整。然而，第二个可能需要更多的技术支持，因此提出这种调整方法。详细步骤如下。

(1) 对于 g_f 的一个边 pq，通过查找表来判断是否避开了所有障碍物。如果避开，则重复此步骤以检查下一条边，直到检查完所有的边；否则，转到步骤(2)。

(2) 通过查找表来检查边 pq 的选择 0 和选择 1。如果它们中的任何一个可以避开所有障碍物，则用当前布线选择替换原始布线选择；否则，删除边 pq，然后转到步骤(3)。

(3) 通过查找表列出 pq 贯穿的所有障碍物，并根据 p 与障碍物中心之间的距离以非递减顺序对这些障碍物进行排序。假设排序列表是 $B_k=\{B_{k,1},B_{k,2},\cdots,B_{k,n}\}$和当前起始点 $s=p$，并且当前障碍物 $B=B_{k,j}$，$j=1$。

(4) 从 B 中选择最接近直线 sq 的角点 c。计算 sc 的连接信息并将该信息添加到查找表中。同时，优先使用选择 0 和选择 1 连接 sc。然后设置 $s=c$。如果 $j=n$ 并转到步骤(5)；否则，重复此步骤，$B=B_{k,j+1}$。

(5) 计算 sq 的连接信息，并将该信息添加到查找表中。然后根据选择 0 和选择 1 的标准连接 sq。

在这种调整方法中，步骤(2)用于处理 Dseg(p,q)，因为它只需要表查找。步骤(3)～步骤(5)用于求解 Cseg(p,q)。图 5.7 是粒子调整示意图，图 5.7(a)显示了 pq 的原始连接图，它穿过障碍物 B_1 和 B_2。删除边 pq 后，根据 p 与 B_1 和 B_2 中心之间的距离，首先检查 B_1。从图 5.7(b)可以看出，c_1 是与直线 pq 最近的角点，因此它被选择为中间节点并连接到 p。然后检查障碍物 B_2。在图 5.7(c)中，c_2 连接到 c_1，因为它是与直线 c_1q 最近的角点。最后，c_2 连接到图 5.7(d)中的 q。有 3 点需要注意。首先，应将新边的所有连接信息添加到查找表中，因为它对精炼策略很有用。其次，步骤(2)仅检查选择 0 和选择 1。即使选择 2 或选择 3 可以避免所有障碍物，也不实施替换操作。最后，采用一种策略，其中在步骤(4)和步骤(5)中优选选择 0 和选择 1。虽然这可能在少数情况下获得非最佳结果，但由于精炼策略

的存在而无关紧要。

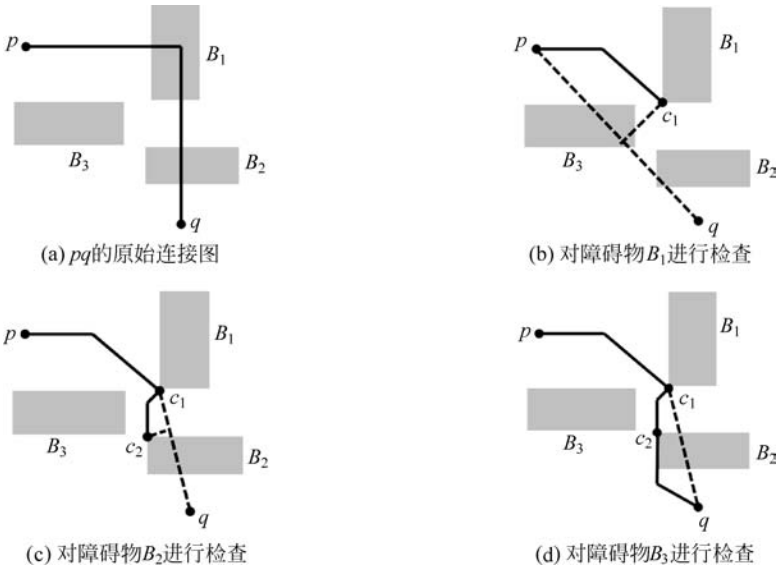


图 5.7 粒子调整示意图

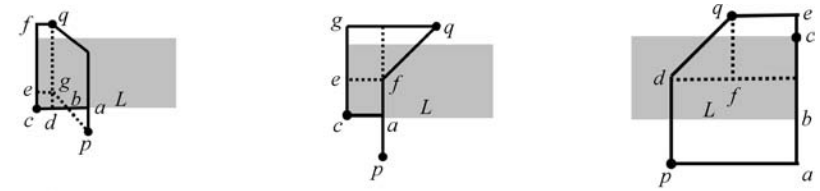
在以下部分中，假设 Rec(p,q)表示使用选择 2 或选择 3 时 p 和 q 之间的距离。Oct(p,q)表示使用选择 0 或选项 1 时的距离，以及 L(p,q)表示 4 个布线选择中的任何一个的距离。

引理 5.2 假设引脚 p 和引脚 q 之间的边形成断开连接模型，并且该边 pq 穿过障碍物 b。为了避开 b，将 b 的一个角点作为中间节点添加会使线长最多增加 $1.3L \sim 1.6L$，其中 L 是 b 的边长。

证明：假设 MAX(a)和 MIN(a)分别是 a 的最大值和最小值。在不失一般性的情况下，假设 p 低于障碍物并且靠近 b 的左下角，q 高于障碍物，并且 pq 形成断开连接模型。图 5.8 是引理 5.2 的实例图，图 5.8(a)显示了 q 在 p 的左侧的情况，从而可以根据调整方法选择 b 的左下角和左上角两者作为中间节点。因为两个角点的 Rec(p,c)+Rec(c,q)是相同的，所以只给出第一个角点的证明。另一种情况是 q 位于 p 的右侧，因此可以选择 b 的左下角和右上角作为中间节点。图 5.8(b)和图 5.8(c)分别显示了连接图。对于图 5.8(a)，有以下内容。

$$\begin{aligned}
&\mathrm{MAX}(1(p,c)) + \mathrm{MAX}(1(c,q)) - \mathrm{MIN}(1(p,q)) \\
&= \mathrm{Rec}(p,c) + \mathrm{Rec}(c,q) - \mathrm{Oct}(c,q) \\
&= pa + ab + bd + dc + ce + ef + fq - pb - bg - gq \\
&= (pa + ab - pb) + (bd + ce - bg) + dc + (ef - gq) + fq \\
&\approx 0.6 \times ad + 2 \times dc < 0.3L + L = 1.3L \left(ad < \frac{L}{2}, dc < \frac{L}{2}\right)
\end{aligned}$$

对于图 5.8(b)，有以下内容。



(a) 引脚q在p的左侧　(b) 连接图一　(c) 连接图二

图 5.8　引理 5.2 的实例

$$\begin{aligned}&\mathrm{MAX}(1(p,c))+\mathrm{MAX}(1(c,q))-\mathrm{MIN}(1(p,q))\\&=\mathrm{Rec}(p,c)+\mathrm{Rec}(c,q)-\mathrm{Oct}(c,q)\\&=pa+ac+ce+eg+gh+hq-pa-af-fq\\&=ac+eg+gh+hq-fq\\&=ac+gh+(eq+hq-fq)\\&\approx 2\times gh+0.6\times hq<L+0.6L=1.6L\left(gh<\frac{L}{2},hq<L\right)\end{aligned}$$

对于图 5.8(c)，有以下内容。

$$\begin{aligned}&\mathrm{MAX}(1(p,c))+\mathrm{MAX}(1(c,q))-\mathrm{MIN}(1(p,q))\\&=\mathrm{Rec}(p,c)+\mathrm{Rec}(c,q)-\mathrm{Oct}(c,q)\\&=pa+ab+bc+ce+eq-pd-dq\\&=pa+eq+df-dq\\&\approx pa+eq-0.4fq<L+0.5L=1.5L\left(gh<\frac{L}{2},hq<L\right)\end{aligned}$$

因此，增加的线长为 $1.3L\sim1.6L$。对于具有不同 p 和 q 位置的其他情况，可以使用类似的证明，因此这里省略它们。

引理 5.3　如果点 c 是点 p 和点 q 的中间节点，则如果 c 位于 p 和 q 的矩形边界框中，则 $\mathrm{Rec}(p,q)=\mathrm{Rec}(p,c)+\mathrm{Rec}(c,q)$。如果 c 位于由 p 和 q 形成的 45°或 135°方向形成的平行四边形中，则 $\mathrm{Oct}(p,q)=\mathrm{Oct}(p,c)+\mathrm{Oct}(c,q)$。

证明：根据 p、c 和 q 形成的分段之间的平行关系，可以很容易地得到方程。因此，此处不再赘述。

引理 5.4　当引脚 p 和引脚 q 之间的边形成矩形连接模型并穿过障碍物 b 时，与仅改变 pq 的布线选择相比，添加一个角点 c 作为中间节点可以使线长减少。

证明：图 5.9 是引理 5.4 实例，如图 5.9(a)所示，pq 贯穿障碍物 b。由于边 pq 形成了一个矩形连接模型，可以通过将 pq 的布线选择从选择 0 替换为选择 2 来绕开障碍物 b。但是，如图 5.9(b)所示，如果选择 c 点作为中间节点，将有 $\mathrm{oct}(p,c)\leqslant\mathrm{Rec}(p,c)$，$\mathrm{oct}(c,q)\leqslant\mathrm{Rec}(c,q)$，所以当且仅当 p、c 和 c、q 分别共线时，等号成立。根据引理 5.3，由于 c 位于 pq 的矩形边界框中，因此 $\mathrm{Rec}(p,c)+\mathrm{Rec}(c,q)=\mathrm{Rec}(p,q)$。因此，有 $\mathrm{Oct}(p,c)+\mathrm{Oct}(q,c)\leqslant\mathrm{Rec}(p,q)$。请注意，在最坏的情况下，$pc$ 和 cq 会形成矩形连接模型，因此线长减少为零。

(a) 从选择0替换为选择2　　(b) 选择点c为中间点

图 5.9　引理 5.4 实例

定理 5.1　使用调整方法可以增加线长或减少线长。

证明：如上所述，当存在断开连接模型时，线长可能会增加 $1.3L \sim 1.6L$。然而，当存在矩形连接模型时，在引理 5.4 中已经证明，与边直接避开障碍物的情况相比，线长也可以减小。此外，对于八角连接模型与完全连接模型，如果存在一个穿过障碍物的边，则调整方法将用选择 0 或选择 1 来替换原始布线选择。它是 p 和 q 之间的最短路径。该策略对于减少线长具有重要意义。

5. 精炼

引理 5.1 证明了粒子在更新后的位置可能会变差。事实上，粒子以随机模式飞行是 PSO 算法具有如此强大的搜索能力的主要原因之一。此外，在调整步骤中，优先使用选择 0 和选择 1 的规则并不总是有效的，因此获得的结果可能仍然包括或多或少的非最优结构。图 5.10 是连接子结构的示意图，如图 5.10 所示(为了简单起见，只给出水平距离大于两垂直距离的两个引脚的图)，p 是一个 2 度引脚，分别连接到引脚 q 和引脚 g。有两条边，每条边有 4 种布线选择，因此可以得到 16 个子结构，但只有一个是最优的。假设 Hor(p,q)表示 p 和 q 之间的水平距离。

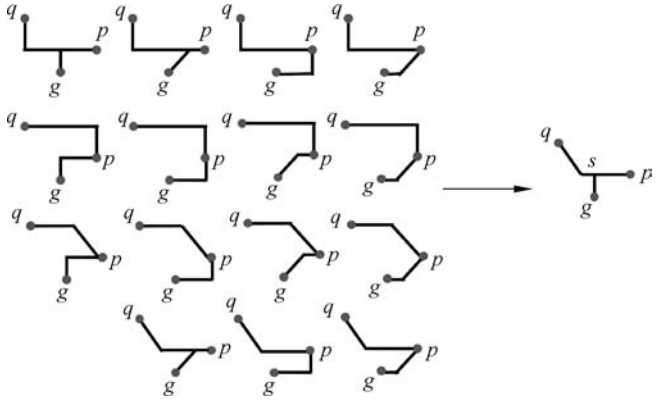


图 5.10　连接子结构的示意图

在图 5.11 中，Hor(p,q)＞Hor(p,g)，s 是 pq 使用选择 0 时的 Steiner 点。当 g 在 p 和 s 之间(包括边界)时，最优结构将是 $pq0$ $pg3$。图 5.11 是 g 在 q 和 s 的连接示意图。在这种情况下，p 和 g 之间的布线选择取决于实际长度的增加。

也就是说，当 $ge+es<gd$ 或 $es>ed>0$ 时，$pg3$ 将被选择，否则，$pg0$ 将被完成。此外，图 5.12 为一个 4 度模型，它可以被分成两个 2 度模型。图 5.12 右侧的 9 个结构是最佳结构的候选。

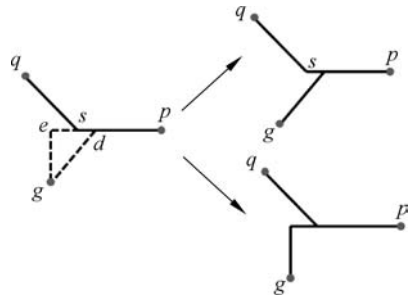


图 5.11　连接示意图

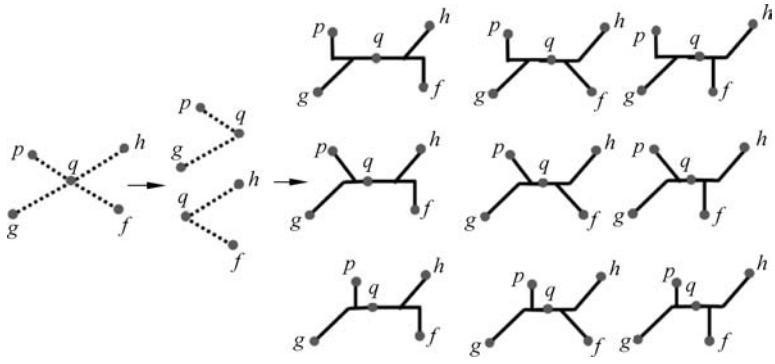


图 5.12　4 度模型

在前面分析的基础上，本节提出了一种基于公共边策略的面向引脚的精炼方法，该方法能够将所有这些非最优结构转换为最优结构，同时绕开所有障碍。具体步骤如下。

(1) 扫描粒子 g_f 的每个边以计数每个端点 p 的度，同时记录所有连接到 p 的端点。

(2) 对于每个端点 p，如果 p 的度为 d，则枚举 p 的所有 4^d 的布线选择组合，并选择获得最小适应度值的组合，同时避开所有障碍物。

在细化过程中，有两点需要注意。首先，关于终端的组合是否避开所有障碍物的问题可以通过查表直接确定，因为在调整过程中添加了障碍物角点的必要连接信息。其次，对于每个绕障组合，为了为整个粒子选择最优组合，可能需要计算整个粒子的适应度值，而不仅仅是组合本身的长度。图 5.13 显示了一个示例冲突。在计算引脚 p 的最优组合时，将得到 $pq0$ $pg3$。然而，当$sq>$

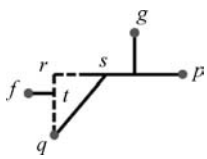


图 5.13　示例冲突

$sr+rt$ 时，$pq3\ fq3$ 才是最佳结果，因此需要从全局角度来决定 pq 的布线选择。

实际上，图 5.14 是粒子改进的飞行过程，利用这种改进方法，粒子群不必在飞行步骤期间搜索期望的目标，因此提高了算法的效率。

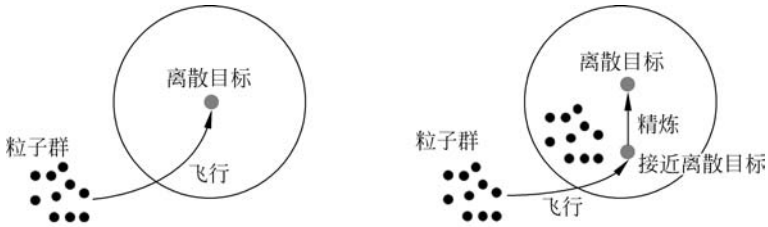


图 5.14 粒子改进的飞行过程

算法 5.2 显示了该改进方法的伪代码。

算法 5.2 改进方法

输入：经过调整后的全局最优粒子 g'_f，引脚总数 n

输出：新的粒子 p

```
Initialize list() to empty; //初始化每个引脚所连接的引脚列表为空
Initialize degree() to zero; //初始化每个引脚的度为 0
for each edge l(u,v) of g'_f do //粒子 g'_f 中的每条边，u 和 v 是一条边的两个引脚
  add u to list(v);
  degree(v)+1;
  add v to list(u);
  degree(u)+1;
end for
f=Fitness(g'_f);
g= g'_f;
for each pin p do //对于粒子 g'_f 中的每个引脚 p
  for each routing combination t of p do //枚举所有编码组合
    if check(t)=true then //如果当前的编码组合绕开障碍物或满足松弛条件
      replace(g, t); //使用当前编码组合替换原始编码组合
      fmid=Fitness(g'_f); //更新粒子适应度值
      if fmid<f then //如果在当前编码组合下整个粒子 g_f' 的适应度值更小
        g'_f=g;
        f=fmid;
      else
        g= g'_f;
      end if
```

```
    end if
  end for
end for
```

degree(p)用于计算 p 的度，list(p)用于记录与 p 连接的所有终端，replace(g，t)表示用组合 t 代替粒子 g 的相应部分，check(t)表示通过表查找检查组合 t 的每条边是否绕开了所有障碍物。

5.2.2　考虑可制造性的后续操作

到目前为止，已经在全局最优粒子被精炼后获得了优秀的基于 X 结构的 OASMT。但是，应该注意的是，这个基于 X 结构的 OASMT 可能包括一些 45°角。这些锐角由两个相交的边形成，例如，如图 5.15(a)所示，两个边 $pg0$ 和 $pq3$ 形成角度 $\theta=45°$。考虑到工业生产的实际情况，应该去掉这些 45°角，因此提出了一种简单的后续处理方法。

通过分析锐角产生的原因和这些锐角的结构，将它们分为两类。第一个可以通过直接改变两条相交边的一个边的布线选择来解决。图 5.15 是消除锐角的示意图，以图 5.15(a)为例，可以通过改变边 pg 从选择 0 到选择 3 的布线选择来消除锐角。如图 5.15(b)所示，得到一个没有锐角的新布线结构。图 5.15(c)显示了第二种锐角。仅通过改变一个边的布线选择并不能消除这种锐角，因为这样做会产生交叉或新的锐角。因此，采用一种策略，通过删除边并生成新边来消除锐角。如图 5.11(d)所示，通过删除边 $pq0$ 并生成边 $pg3$，可以成功地移除 45°角。当然，也可以删除边 $gq0$ 并生成新的边 $gp2$ 以消除这个 45°角(见图 5.15(e))。应当注意，原始边的布线选择也可以在新生成的结构中改变，并且应该保证新的布线选择避开所有障碍物。很明显，可以通过直接查找表来执行此操作。例如，边 pq 的布线选择在图 5.15(e)中从选择 0 改变为选择 3。因为线长是优化目标，算法将选择具有最短线长的结构。根据精炼过程中 degree(p)和 list(p)的结果，可以快速识别所有相交的边。在将该后处理方法应用于最终的全局最优粒子 g_f 之后，可以有效地消除所有锐角。

算法 5.3 显示了后处理方法的伪代码。在算法 5.3 中，change_routing_choice(l)用于改变边 l 的布线选择并且选择不产生任何锐角的那个，同时实现最短的线长。Routing_combination(l_1，l_2)用于计算边 l_1 和 l_2 的布线选择组合，并且在获得最短线长时不获得任何锐角。Select(r_1，r_2)用于选择 r_1 和 r_2 之间的一个组合，并且必须保证该结果避开所有障碍物且实现更短的线长。从算法 5.3 中可以看出，首先尝试使用第一种类型的方法然后使用第二种类型的方法去除锐角。注意，可以通过直接查找表来获得边的所有连接信息，因此该过程是有效的。

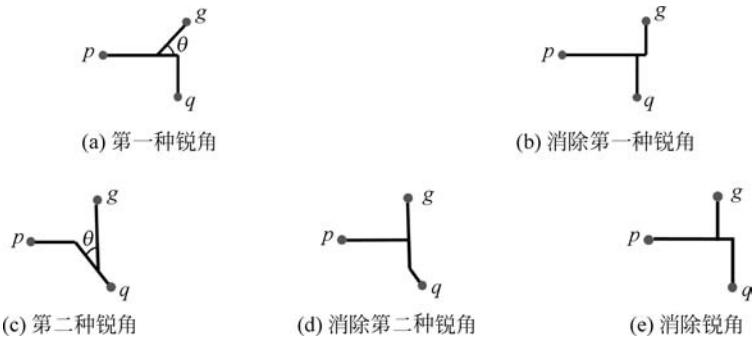


图 5.15 消除锐角的示意图

5.2.3 参数策略

性质 5.1 粒子的局部开发能力和全局探索能力受惯性权重影响。

在 PSO 算法中，粒子飞行的动力由惯性分量提供，体现了先前速度对飞行轨迹的影响，惯性权重则决定了这种影响程度。较大的惯性权重使粒子能够搜索未知的区域，从而跳出局部极值；反之，较小的惯性权重使粒子在目前所达区域附近搜索，使算法加速收敛。

文献[15]、[16]中提出了一种基于线性减小惯性权重的 PSO 算法，并在 4 种不同的基准函数上进行了仿真实验，实验结果表明这种参数策略能够提高 PSO 算法的性能。

性质 5.2 较大的加速因子 c_1 可能使粒子在本地范围内游荡，较大的加速因子 c_2 将使算法过早收敛。

加速因子 c_1 和 c_2 用于粒子之间的通信。在早期使用较大的 c_1 和较小的 c_2，并在之后进行反转。通过这种方式，该算法将保证在局部范围内进行详细搜索，以避免在早期阶段直接移动到全局最优的位置，并加快后续阶段的收敛。同样地，该算法的实验取得了很好的效果。

算法 5.3 后处理方法

输入：最终的全局最优粒子 g_f

输出：最终的布线结果

```
for each pair of intersected edges pg and pq do//对于每对相交的边 pg 和 pq
  if intersected_angle(pg,pq)=45° then//若其形成 45°角
    result1=change_routing_choice(pg);//改变边 pg 的布线选择并选择不产生锐角的那个
    if obstacle_avoiding(result1)=true then //若该结果避开所有障碍物
      set result1 as a result;
```

```
        continue;
      end if
      result2=change_routing_choice(pq); //改变边 pq 的布线选择并选择不产生锐角的
                                         //那个
      if obstacle_avoiding(result2)=true then
        set result2 as a result; //将其设置为最终布线结果
        continue;
      end if
      generate a new edge gq;//生成一条新的边 gq
      result3=routing_combination(pg,gq);//计算边 pg 和 gq 的布线选择组合
      result4=routing_combination(pq,gq); //计算边 pq 和 gq 的布线选择组合
      fr=select(result3,result4);//选择 result3 和 result4 之间的一个组合
      set fr as a result;
    end if
  end for
```

5.2.4　实验结果

本节算法使用C/C++语言实现，在具有2.9 GHz CPU和2GB内存的PC上进行所有实验。并使用MATLAB来模拟最终的布线图。通常使用多组基准作为绕障问题的测试用例。前5组是Synopsys的工业测试案例。其余的是绕障问题的基准。注意，引脚可以位于障碍物的边界。

表5.3显示了精炼之前和之后线长的比较结果，其中Pin#和Obs#分别表示引脚和障碍物的数量。改进率等于((before－after)/before)×100%。如前所述，粒子群在飞行过程中只需要搜索一个次优结果，然后使用精炼方法得到最优结构。从表5.3可以看出，经过精炼后，线长可以得到0.00%～10.56%的改善，平均改善也可以达到7.93%。

众所周知，这是专门解决基于X结构的单层绕障问题的第一项工作。以往的X结构的研究大多集中在无障碍布线平面上，然后对布线长度或时延进行优化。表5.4显示了线长与λ-几何结构的比较，其中列3是通过该算法获得的线长。

与OARSMT(λ=2)相比，线长减少9.61%～48.16%，平均减少25.08%。对于OAOSMT(λ=4)，当问题的规模较小(即只有少数障碍物的小规模网络)时，文献[5]的算法在大多数测试用例中与该结果相似。然而，当问题的规模很大时，特别是当障碍物的数量多于引脚的数量时，由于文献[5]的算法可能会引入一些冗余的角点，该算法具有很大的优势，其中对于具有200个引脚和1000个障碍物的测试用例，最大减少率可以达到42.36%。X架构的平均改进率为16.42%。

表 5.3 精炼前后的线长比较结果

引脚数	障碍数	精炼前	精炼后	优化率/%
10	32	618	562	9.06
10	43	9724	9431	3.01
10	59	574	574	0.00
25	79	1128	1033	8.42
33	71	1440	1288	10.56
10	10	26 761	24 717	7.64
30	10	44 822	40 751	9.09
50	10	57 253	52 033	9.12
70	10	61 725	57 250	7.25
100	10	80 486	72 738	9.63
100	500	85 992	78 643	8.55
200	500	116 905	105 542	9.72
200	800	126 820	116 204	8.37
200	1000	123 177	111 385	9.57
500	100	172 126	157 520	8.49
1000	100	239 231	219 037	8.44
均值				7.93

表 5.4 与 λ-几何结构算法的线长比较结果

引脚数	障碍数	本节算法线长	线长		线长减少率/%	
			λ=2	λ=4	λ=2	λ=4
10	10	24 717	30 410	27 279	18.72	9.39
30	10	40 751	45 640	41 222	10.71	1.14
50	10	52 033	58 570	52 432	11.16	0.76
70	10	57 250	63 340	57 699	9.61	0.78
100	10	72 738	83 150	73 090	12.52	0.48
100	500	78 643	149 725	135 454	47.48	41.94
200	500	105 542	181 470	162 762	41.84	35.16
200	800	116 204	202 741	182 056	42.68	36.17
200	1000	111 385	214 850	193 228	48.16	42.36
500	100	157 520	198 010	176 497	20.45	10.75
1000	100	219 037	250 570	222 758	12.58	1.67
均值					25.08	16.42

此外，表 5.5 将所提出的算法的线长与近年来发表的基于直角结构的一些启发式算法(文献[31]、[32]、[11]、[12]、[19]和文献[20]提出的算法)进行了比较，并显示了迄今为止的最佳结果。为了简单起见，在实验表格中按照前面的顺序将对比文献标记为 A～F。

2014年，在GPU的帮助下，文献[32]提出了构造OARSMT的并行方法，与之相比得到了－0.60%～6.95%的线长改善，平均线长减少了3.87%，只有一个测试情况更糟(0.60%)。虽然文献[31]的算法提出时间稍早，但在这6篇论文中效果最好。其中一个主要原因是文献[31]的算法提出了一种优秀的4步Steiner点选择方法，该方法能够以令人满意的位置添加Steiner点，从而减少线长。与文献[31]的算法相比，本节算法获得了－0.47%～7.72%的线长改善，平均达到4.23%。此外，从表5.5中可以看出，与C、D、E、F 4种优秀算法相比，本节算法可以获得平均4.55%～6.68%的线长改善。

表5.5　与一些直角结构启发式算法的线长比较结果

P	B	Our_L	线长			优化率$(X-\text{Our_L})/X$/%		
			[31]A	[32]B	[11]C	X=A	X=B	X=C
10	32	562	609	604	604	7.72	6.95	6.95
10	43	9431	9500	9600	9500	0.73	1.76	0.73
10	59	574	600	600	600	4.33	4.33	4.33
25	79	1033	1092	1092	1129	5.40	5.40	8.50
33	71	1288	1345	1353	1364	4.24	4.80	5.57
10	10	24 717	25 980	25 980	25 980	4.86	4.86	4.86
30	10	40 751	41 740	41 350	42 110	2.37	1.45	3.23
50	10	52 033	55 500	54 360	56 030	6.25	4.28	7.13
70	10	57 250	60 120	59 530	59 720	4.77	3.83	4.14
100	10	72 738	75 390	74 720	75 000	3.52	2.65	3.02
100	500	78 643	81 340	81 290	81 229	3.32	3.26	3.18
200	500	105 542	110 952	110 851	110 764	4.88	4.79	4.71
200	800	116 204	115 663	115 516	116 047	－0.47	－0.60	－0.14
200	1000	111 385	114 275	113 254	115 593	2.53	1.65	3.64
500	100	157 520	167 830	166 970	168 280	6.14	5.66	6.34
1000	100	219 037	235 866	234 875	234 416	7.13	6.74	6.56
均值						4.23	3.87	4.55
P	B	Our_L	线长			优化率$(X-\text{Our_L})/X$/%		
			[12]D	[19]E	[20]F	X=D	X=E	X=F
10	32	579	636	632	639	11.64	11.08	12.05
10	43	9489	9600	9600	10 000	1.76	1.76	5.69
10	59	588	613	613	623	6.36	6.36	7.87
25	79	1085	1116	1121	1126	7.44	7.85	8.26
33	71	1321	1364	1364	1379	5.57	5.57	6.60
10	10	25 130	25 980	26 900	27 540	4.86	8.12	10.25
30	10	40 924	42 070	42 210	41 930	3.14	3.46	2.81
50	10	54 039	54 630	55 750	54 180	4.75	6.67	3.96

续表

P	B	Our_L	线长			优化率$(X-\text{Our_L})/X/\%$		
			[12]D	[19]E	[20]F	$X=$D	$X=$E	$X=$F
70	10	58 536	59 270	60 350	59 050	3.41	5.14	3.05
100	10	74 063	75 410	76 330	75 630	3.54	4.71	3.82
100	500	78 643	82 300	83 365	86 381	4.44	5.66	8.96
200	500	105 542	111 752	113 260	117 093	5.56	6.81	9.86
200	800	116 204	116 646	118 747	122 306	0.38	2.14	4.99
200	1000	111 385	113 781	116 168	119 308	2.11	4.12	6.64
500	100	157 520	167 540	170 690	167 978	5.98	7.72	6.23
1000	100	219 037	234 097	236 615	232 381	6.43	7.43	5.74
均值						4.83	5.91	6.68

对于运行时间，一方面，与传统的直角结构相比，X 结构引入了 45°和 135°的布线方向。这无疑会增加问题的复杂性。另一方面，由于 PSO 算法具有非常强的搜索能力，所以使用它来优化线长。但从本质上讲，PSO 算法是一种基于群的方法，与普通的启发式方法相比非常耗时。表 5.6 给出了算法的详细运行时间。第 3 列显示了预处理时间，可以看到，大多数测试用例只需要不到一秒钟。第 4 列显示了粒子飞行步骤的运行时间。在这里，有两点要解释。首先，对于所有测试案例，由于在下一步骤中将会使用调整方法和精炼方法，粒子群只需要搜索次优结果，甚至可以包括一些穿过障碍物的边。因此大大减少了搜索时间。其次，当障碍物的数量大于引脚的数量时，很明显会形成许多断开连接模型，因此无论如何增加搜索过程的强度，粒子都无法避开所有障碍物。因此，借助于调整方法仅需要相对少量的搜索时间。表 5.6 第 11 行～第 14 行属于这种类型。第 5 列描述了有关调整方法的信息，其中 null 表示最终全局最佳粒子直接避免了所有障碍的情况，即没有用于调整的情况。在 a/b 中，a 是调整的运行时间，b 是通过调整选择的角点的总数。可以看出，大多数测试用例选择了几个角点。表 5.6 第 11 行～第 14 行选择更多的角点，因为它们包含更多断开连接模型，因此调整时间也相对较长。第 6 列是精炼方法的运行时间，这是算法中最耗时的部分。主要原因如下。为了选择一个端点的最佳布线组合，需要计算粒子的适应度值，而不仅仅是绕障组合本身的长度。问题是公共边应该只计算一次，因为由于存在 45°和 135°段，处理和判断它们需要花费更多的时间。最后一列显示了所提算法的总运行时间，其中超过一半的测试用例可以在一秒内获得很好的结果。对于最后一组测试用例，最长的运行时间是 9.329s，但这是一个相当优异的结果。从表 5.6 可以看出，所有运行时间都是合理且可接受的。

表 5.6　算法的详细运行时间

引脚数	障碍数	预处理时间	粒子飞行时间	调整时间	精炼时间	合　计
10	32	0.000s	0.015s		0.000s	0.015s
10	43	0.000s	0.015s	0.000s/7	0.000s	0.015s
10	59	0.000s	0.015s		0.000s	0.015s
25	79	0.000s	0.015s	0.000s/8	0.000s	0.015s
33	71	0.000s	0.015s	0.000s/19	0.031s	0.046s
10	10	0.000s	0.015s	0.000s/4	0.000s	0.015s
30	10	0.000s	0.015s	0.000s/5	0.000s	0.015s
50	10	0.000s	0.063s	0.000s/4	0.031s	0.094s
70	10	0.000s	0.094s	0.000s/4	0.095s	0.189s
100	10	0.000s	0.172s	0.000s/6	0.083s	0.255s
100	500	0.140s	0.016s	0.015s/91	0.385s	0.556s
200	500	0.546s	0.016s	0.015s/87	1.373s	1.950s
200	800	0.889s	0.032s	0.015s/76	2.015s	2.951s
200	1000	1.139s	0.078s	0.016s/209	2.030s	3.263s
500	100	0.795s	0.093s	0.000s/39	4.515s	5.403s
1000	100	3.073s	0.312s	0.000s/25	5.944s	9.329s

5.2.5　小结

随着制造技术的进步，在 X 结构模型中可以允许 45°和 135°对角线段。本节算法提出了一种基于离散 PSO 在 X 架构中构建 OASMT 的有效方法。同时，集成了一些启发式策略和 GA 算法的两个算子，进一步提高了算法的性能。经过实验证明所提出的算法在合理的运行时间内获得了很好的结果，与基于直角结构的最新成果相比，获得了线长提高 3.87%～6.68%。对于 X 架构，与相关的工作相比较，本节算法平均减少了 16.42%的线长。

5.3　快速绕障 X 结构 Steiner 最小树算法

5.3.1　引言

自从 1966 年人们提出 Hanan 网格以来，直角 Steiner 最小树（Rectilinear Steiner Minimum Tree，RSMT）被广泛应用于超大规模集成电路（Very Large Scale Integration，VLSI）的布线问题。另一方面，由于 VLSI 电路的密度急剧增加，许多可重复使用的组件集成在现代芯片中，例如 IP 块。在布线过程中无法移动这些组件。因此，在过去的几年中，OARSMT 的构建问题得到了广泛的研究。

随着 VLSI 电路制造技术的进步，X 结构布线模型中可以允许 45°和 135°的斜线段。与传统的直角结构相比，X 结构可以极大地优化线长，从而降低了时间延

迟、线容量和拥塞的成本。例如，文献[23]提出的 X 结构 Steiner 最小树(OSMT)算法显示了其与 RSMT 相比，通孔数量、总线长和管芯尺寸分别减少了 40%、20%和 11%。因此，近年来学术界和工业界都致力于使用 X 结构。

作为一个基本问题，研究人员已经提出了一些用于解决 OSMT 构造问题的算法。例如，文献[23]和文献[24]提出了构造时间驱动的 OSMT 算法，其可以显著提高芯片的性能。此外，文献[18]还提出了对应方法来解决 λ-几何绕障 Steiner 树构造问题。

本节提出了一种 OAOSMT 构建算法。主要贡献总结如下：

(1) 算法生成了两个有关引脚和障碍物的查找表，为整个算法提供了快速的信息查询。

(2) 在快速查表的基础上，提出了一种绕障策略和一种精炼技术。这两种方法都是有效和高效的。

(3) 与以前的算法相比，本节算法的线长和运行时间都是最好的。对科研和工业生产都具有价值。

5.3.2 算法框架

图 5.16 是 OAOSMT 的构建输入引脚和障碍物。本节算法可以分为以下 4 个步骤：

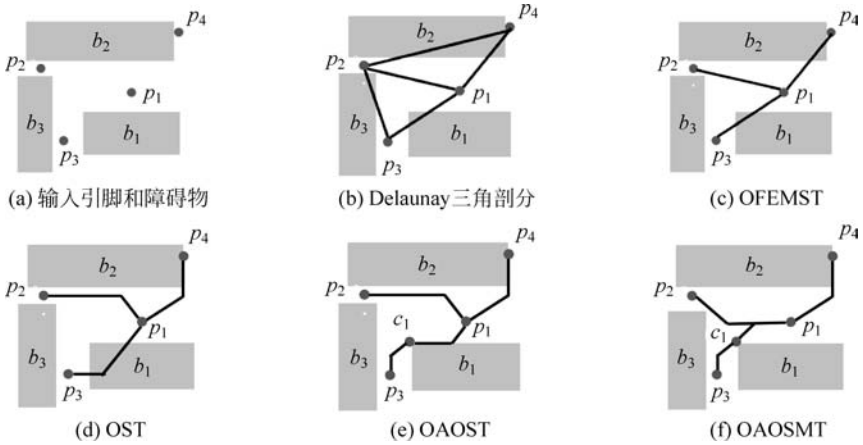


图 5.16 OAOSMT 的构建

步骤 1，首先在给定的引脚顶点上构造 Delaunay 三角剖分(DT)(见图 5.16(b))，并构造 OFEMST(见图 5.16(c))。

步骤 2，生成两个查找表，用于记录有关 OFEMST 边的连接信息。它们可以为第 3 步和第 4 步提供快速信息查询。

步骤 3，OFEMST 首先转换为 X 结构 Steiner 树(OST)(见图 5.16(d))。然后利用障碍物的一些角生成 OAOST。在图 5.16(e)中，c_1 被选择为 p_1p_3 之间的中

间节点。

步骤 4，采用一种能够充分利用共享边原理的精炼技术，来生成最终的 OAOSMT(见图 5.16(f))。

5.3.3　算法细节

1. OFEMST 的构建

在无障碍平面上，有许多现有的最小生成树构造算法。本节采用了一种基于 DT 的策略，首先根据给定的引脚构建 DT。然后可以使用 Prim 或 Kruskal 算法在 $O(n\log n)$时间内构造 OFEMST。之后直接使用文献[25]中的 Sweepline 算法来构造 DT，此处不再赘述。

2. 预查表生成

本节算法计算了 OFEMST 边在一个 X 结构平面上的连接信息。在此步骤中生成两个预先计算的查找表。第一种称为边-障碍表(Edge-Obstacle Table，EOT)，记录每条 X 结构边穿过的障碍物信息。另一个称为边-段表(Edge-Segment Table，EST)，记录每条 X 结构边的两个线段的坐标信息。

由于 OFEMST 具有 $n-1$ 个边，并且每条边具有 4 个布线选择，因此总共有 $4(n-1)$条 X 结构边。对于每一条 X 结构边，记录穿过的障碍物集合。同时，将值设置为这些障碍物的半周长之和。所有 $4(n-1)$条边和其集合构成了最终的 EOT。此外，每条边包括两条线段。记录这两个段的坐标。所有 X 结构边的信息构成最终 EST。

查找表生成的伪代码在算法 5.4 中给出。有两点需要注意。首先，在第 3 行，对于障碍物 b，仅在 b 的至少一个拐角点与 p_ip_j 的边界框重叠时检查它。其次，如果存在一个 45°或 135°段，将其绕原点顺时针旋转 45°，形成一个新的水平或竖直段(第 9～14 行)，然后记录新段的坐标。这有利于总线长和局部线长的计算，因为可以轻松识别公共的 45°和 135°段，而无须更改线长。

算法 5.4　查找表生成

```
输入：OFEMST
输出：EOT，EST
  for each edge p_i p_j of OFEMST do
    for each routing choice k do
      for each obstacle b do //EOT 生成
        if run_through(p_i s_k ,b)=True or run_through(s_k p_j ,b)=True then
          L_ijk = L_ijk + semi-perimeter(b);//将 L_ijk 值设置为这些障碍物的半周长之和
          Add b to {B_ijk};//将障碍物 b 加入 p_i p_j k 穿过的障碍物集合{B_ijk}
        end if
      end for
```

```
        if p_i s_k = 45° or p_i s_k = 135° then//EST 生成
          Rotate(p_i s_k ,45°);//将 p_i s_k 绕原点顺时针旋转 45°
        end if
        if s_k p_j = 45° or s_k p_j = 135° then
          Rotate(s_k p_j ,45°); //将 p_i s_k 绕原点顺时针旋转 45°
        end if
        record the coordinates of p_i s_k ;//记录新段 p_i s_k 的坐标
        record the coordinates of s_k p_j ;//记录新段 s_k p_j 的坐标
      end for
    end for
```

3. 绕障策略

1) OST 生成

在该步骤中,基于查找 EOT,将 OFEMST 转换为 OST,细节如下。

检查 OFEMST 的每条边 p_ip_j。如果 X 结构的边 p_ip_j0 或 p_ip_j1 可以避开所有障碍物,直接将 p_ip_j2 的布线选择设置为选择 0 或选择 1;否则,如果 p_ip_j2 或 p_ip_j3 可以避开所有障碍物,仍然设置(根据 L_{ij0} 和 L_{ij1} 的值)选择 0 或选择 1。如果所有 4 个布线选择都通过障碍物,那么选择具有最小 L_{ijk} 的布线选择。应该注意的是,所有判断都可以通过直接查找 EOT 来执行。因此,这个过程非常快。OST 生成的伪代码在算法 5.5 中给出。

算法 5.5 OST 生成

```
输入: OFEMST,EOT
输出: OST
  for each edge p_i p_j of OFEMST do
    if {B_ij0} = ∅ then //如果 X 结构的边 p_i p_j 0 可以避开所有障碍物
      p_i p_j k = p_i p_j 0;//将 p_i p_j 的布线选择设置为选择 0
    end if
    else if{B_ij1} = ∅ then //如果 X 结构的边 p_i p_j 1 可以避开所有障碍物
      p_i p_j k = p_i p_j 1; //将 p_i p_j 的布线选择设置为选择 1
    end if
    else if{B_ij2} = ∅ or {B_ij3} = ∅ then//如果 p_i p_j 2 或 p_i p_j 3 可以避开所有障碍物
      //根据 L_ij0 和 L_ij1 的值设置选择 0 或选择 1
    if L_ij0 <= L_ij1 then
      p_i p_j k = p_i p_j 0;
    else
```

```
        p_i p_j k = p_i p_j 1;
      end if
    else//如果所有 4 个布线选择都通过障碍物
        select min(L_ijt), t=0,1,2,3;//选择具有最小 L_ijt 的布线选择
        p_i p_j k = p_i p_j t;
    end if
  end for
```

2）OAOST 生成

显然，OST 的某些边可能会穿过障碍物。因此，提出了一种绕障方法来帮助这些边绕开障碍。算法 5.6 中显示了 OAOST 生成的伪代码。

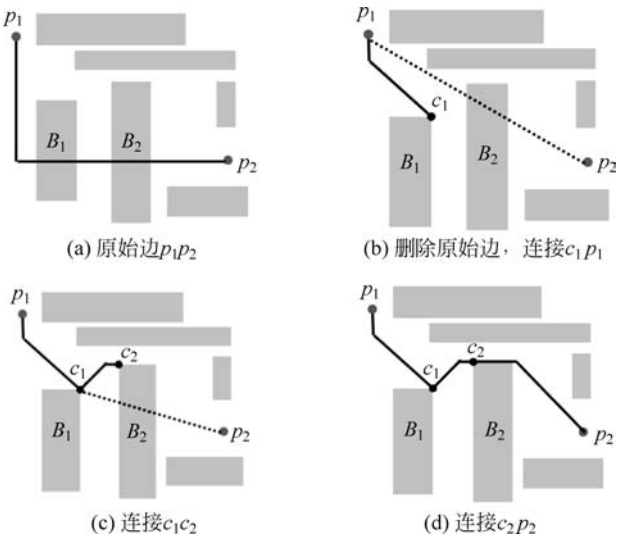
算法 5.6　OAOST 生成

```
输入：OST,EOT
输出：OAOST
  for each edge p_i p_j k of OST do
    if {B_ijk} ≠ ∅ then //如果边 p_i p_j k 穿过了障碍物
      delete(p_i p_j k);//删除 p_i p_j k
      sort({B_ijk});//根据 Dis(p_i,o)按非递减顺序对{B_ijk}进行排序,o 是障碍物的中心点
    end if
    s = p_i;//设起点为 p_i
    for each obstacle b in {B_ijk} do//逐一检查障碍集中的障碍物
        c = select_corner(b);//从当前障碍物中选择一个拐点 c
        add sc to EOT and EST;//计算边 sc 的连接信息,并将这些信息添加到 EOT 和 EST 中
        connect s with c;//将 s 连接到 c
        s = c;//用 c 替换 s 作为当前起点
    end for
    add sp_j to EOT and EST;//计算边 sp_j 的连接信息,并将这些信息添加到 EOT 和
                            //EST 中
    connect s with p_j;//将最后一个选定的拐点 s 连接到 p_j
  end for
```

对于 OST 的每条边 p_ip_jk（k 是 OST 中的布线选择），如果它穿过了障碍物，则直接删除 p_ip_jk，并根据 Dis(p_j,o)按非递减顺序对障碍物集$\{B_{ijk}\}$进行排序，其中 o 是障碍物的中心点。然后设起点 s 为 p_j。对于已排序的集合$\{B_{ijk}\}$，逐一检查这些障碍物。从当前障碍物中选择一个拐点 c，它是最接近直线 sp_j 的。然后计算边 sc 的连接信息，并将这些信息添加到 EOT 和 EST 中。将 s 连接到 c 后，

用 c 替换 s 作为当前起点，并继续检查下一个障碍物，直到最后一个选定的拐点连接到 p_j。

这种绕障方法可以操作多次，直到所有的边都绕开所有的障碍物。图 5.17 展示了一个简单绕障实例。图 5.17(a)是原始边 p_1p_2，它穿过障碍物 B_1 和 B_2（$\{B_{123}\}=\{B_1,B_2\}$）。删除边 p_1p_2 后，由于 $\mathrm{Dis}(p_1,o_1)<\mathrm{Dis}(p_1,o_2)$，首先检查 B_1。由于 c_1 是 B_1 与直线 p_1p_2 最接近的拐点，因此选择它作为中间节点并连接到 p_1（见图 5.17(b)）。然后检查 B_2。在图 5.17(c)中，由于 c_2 是与直线 c_1p_2 最近的拐点，所以 c_2 连接到 c_1。最后，在图 5.17(d)中，c_2 连接到 p_2。注意，新边的所有连接信息都应该添加到 EOT 和 EST，因为它们对于精炼是有用的。



(a) 原始边p_1p_2　(b) 删除原始边，连接c_1p_1

(c) 连接c_1c_2　(d) 连接c_2p_2

图 5.17　绕障实例

4. 精炼

事实上，对于任何一个引脚，其互连总是存在一个最佳结构。图 5.18 显示了具有 2 度端点 p_3 的一个示例，其中 p_1 和 p_2 连接到 p_3。在不损失一般性的情况下，假设两个引脚之间存在 $\Delta x>\Delta y$。由于每个边有 4 个布线选择，总共有 16 个子结构。但当 p_2 在 s_1 和 p_3 之间时，其中 s_1 是边 p_1p_2 的 pseudo-Steiner 点，则图 5.18(a)是 p_3 的唯一最佳结构。当 p_2 在 p_1 和 s_1 之间（见图 5.18(b)）时，如果 $p_2e+es_1<p_2d$ 或 $es_1>ed$，则图 5.18(c)成为最佳结构；否则，图 5.18(d)是最佳的。障碍物存在的平面也遵循类似的原理。但是最佳结构可能成为 16 个子结构中的任意一个。例如，如图 5.18(e)所示，p_2 只能通过选择 3 连接到 p_3。因此，图 5.18(e)成为最佳结构之一。

基于上述分析，提出了一种精炼技术（见算法 5.7）。通过扫描 OAOST 的边一次，首先计算每个引脚 p_i 的度，并将连接到 p_i 的引脚记录为列表。然后计算每

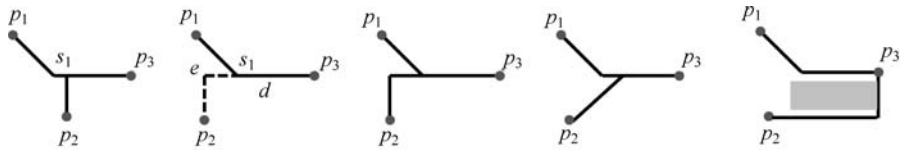


图 5.18　精炼实例

个引脚 p_i 的最佳结构(os_i)。假设 p_i 的度为 d。列举了 p_i 的所有 4^d 个布线选择组合,并选择了具有最小线长的且绕障的那一个作为 os_i。此外,计算每个 os_i 的公共边(se_i)的长度。接着根据 se_i 的值按降序对所有引脚进行排序。最后,根据已排序的引脚列表将每个引脚的 os_i 应用于原始的 OAOST,直到确定了所有 OAOST 边的布线选择。

算法 5.7　精炼

输入：OAOST,EOT,EST
输出：OAOSMT

```
for each edge p_i p_j k of OAOST do
    degree(p_i)=degree(p_i)+1;//计算每个引脚 p_i 的度
    degree(p_j)=degree(p_j)+1;//计算每个引脚 p_j 的度
    add p_j to list(p_j);//将连接到 p_j 的引脚记录为列表
    add p_i to list(p_i);//将连接到 p_i 的引脚记录为列表
end for
    //计算每个引脚 p_i 的最佳结构 os_i
    Initialize each length(os_i)=+∞;
    for each terminal p_i of OAOST do
        for each substructure st of p_i do//对于 p_i 的所有布线选择组合
            if obstacle-avoiding(st)=True and length(st)<length(os_i) then
                os_i=st;//选择具有最小线长的且绕障的那一个作为 os_i
            end if
        end for
        se_i=length_shared_edges(os_i);//计算每个 os_i 的公共边的长度 se_i
    end for
    sort({P}) in decreasing order according se_i;//根据 se_i 的值按降序对所有引脚进行排序
    for each terminal p_i of OAOST do
        apply os_i to OAOST;//将每个引脚的 os_i 应用于原始的 OAOST
        if all edges of OAOST have been decided then //直到确定了所有 OAOST 边的布线选择
```

```
            break;
        end if
    end for
```

有两点需要注意。首先，可以通过直接查找表来确定布线选择组合是否避免所有障碍(第 10 行)。其次，如果已经确定了 OAOST 边的布线选择，则即使当前 os_i 对该边具有不同的选择，也不会改变它(第 18 行)。

定理 5.2 该算法的最坏时间复杂度为 $O(nm)$，其中 n 和 m 分别是引脚和障碍物的数。

5.3.4 实验结果

所有算法都已用 C 语言实现和执行，实验在具有 2.9GHz CPU 和 2GB 存储器的 PC 上进行。IND1～IND5 是 Synopsys 的工业测试用例。RC01～RC12 是绕障问题的基准。目前解决单层 OAOSMT 问题的唯一相关工作是文献[18]提出来的，是为 λ-几何问题设计的。此外，还有近年来文献[31]、[32]、[12]所提出的 OARSMT 构建算法。表 5.7 显示了本节算法与其他算法的线长比较结果。

表 5.7 与其他算法的线长比较结果

测试用例	引脚数	障碍物数	本节算法线长	线长		优化率/%	
				文献[18] λ=4	文献[32]	文献[18] λ=4	文献[32]
IND1	10	32	569	—	609	—	6.57
IND2	10	43	9549	—	9500	—	−0.52
IND3	10	59	575	—	600	—	4.17
IND4	25	79	1070	—	1092	—	2.01
IND5	33	71	1378	—	1345	—	−2.45
RC01	10	10	25 084	27 279	25 980	8.05	3.45
RC02	30	10	39 488	41 222	41 740	4.21	5.40
RC03	50	10	54 177	52 432	55 500	−3.33	2.38
RC04	70	10	59 988	57 699	60 120	−3.97	0.22
RC05	100	10	72 833	73 090	75 390	0.35	3.39
RC06	100	500	78 079	135 454	81 340	42.20	4.01
RC07	200	500	105 950	162 762	110 952	34.90	4.51
RC08	200	800	113 943	182 056	115 663	37.41	1.49
RC09	200	1000	111 258	193 228	114 275	42.42	2.64
RC10	500	100	156 329	176 497	167 830	11.43	6.85
RC11	1000	100	216 937	222 758	235 866	2.61	8.03
RC12	1000	10 000	703 669	1 564 170	762 124	55.01	7.67
均值						19.29	3.52

续表

测试用例	引脚数	障碍物数	本节算法线长	线　长		优化率/%	
				文献[18] $\lambda=4$	文献[32]	文献[18] $\lambda=4$	文献[32]
IND1	10	32	569	604	639	5.79	10.95
IND2	10	43	9549	9500	10 000	−0.52	4.52
IND3	10	59	575	600	623	4.17	7.70
IND4	25	79	1070	1129	1126	5.23	4.97
IND5	33	71	1378	1364	1379	−1.03	0.01
RC01	10	10	25 084	25 980	27 540	3.45	8.92
RC02	30	10	39 488	42 110	41 930	6.23	5.82
RC03	50	10	54 177	56 030	54 180	3.31	0.00
RC04	70	10	59 988	59 720	59 050	−0.45	−1.59
RC05	100	10	72 833	75 000	75 630	2.89	3.70
RC06	100	500	78 079	81 229	86 381	3.88	9.61
RC07	200	500	105 950	110 764	117 093	4.35	9.52
RC08	200	800	113 943	116 047	122 306	1.81	6.84
RC09	200	1000	111 258	115 593	119 308	3.75	6.75
RC10	500	100	156 329	168 280	167 978	7.10	6.93
RC11	1000	100	216 937	234 416	232 381	7.56	6.65
RC12	1000	10 000	703 669	756 998	842 689	7.04	16.50
均值						3.80	6.34

表 5.7 中第 4 列是本节算法的线长，与文献[18]所提算法相比，当问题的规模较小时(即只有几个障碍的小规模网络)，文献[18]所提算法具有与本节算法相似的平均表现(RC01～RC05)。而在大规模的用例中，即在引脚数大于 70 的线网中，本节算法的线长优化能力比文献[18]的算法优秀很多，最大减少了 55.01%的线长。平均改善率也可达到 19.29%。与文献[32]相比，该算法产生的线长优化提高了−0.52%～8.03%，平均减少了 3.52%，只有两个测试用例比它差。此外，该算法平均比文献[31]所提算法优化 3.80%，比文献[12]所提出的算法优化 6.34%。

该算法在实际问题上的处理速度非常快。有 3 个主要原因。首先，步骤 3、4 中使用的查找表的方法只需要 $O(1)$的复杂度。其次，因为步骤 1 生成 MST，两个引脚的矩形边界框很小，除非它们彼此非常远。因此，在步骤 2 和步骤 3 中不需要检查大多数的障碍物。最后，对于步骤 3 中的 OAOST 生成，只需要检查穿过障碍物的边。在表 5.8 中提供与其他算法的运行时间对比。最后一行中的值是标准化结果。与文献[18]的算法相比，本节算法快了 5.93 倍。此外，比文献[32]的算法快 42 倍，比文献[12]的算法快 11.92 倍。特别是，文献[31]提出的算法实现了 4 种算法中最快的运行时间。与之相比，本节算法的平均速度依然提高了 2.35 倍。

表 5.8 与其他算法的运行时间对比

测试用例	CPU 时间/s				
	文献[18]λ=4	文献[32]	文献[31]	文献[12]	本节算法
IND1	—	0.05	0.00	0.01	0.00
IND2	—	0.06	0.00	0.01	0.00
IND3	—	0.05	0.00	0.01	0.00
IND4	—	0.09	0.00	0.02	0.00
IND5	—	0.08	0.00	0.02	0.00
RC01	0.00	0.05	0.00	0.01	0.00
RC01	0.00	0.06	0.00	0.01	0.00
RC03	0.00	0.07	0.00	0.01	0.00
RC04	0.01	0.06	0.00	0.02	0.00
RC05	0.01	0.09	0.00	0.02	0.00
RC06	0.07	0.38	0.03	0.13	0.02
RC07	0.08	0.31	0.03	0.15	0.02
RC08	0.12	0.46	0.05	0.27	0.03
RC09	0.16	0.61	0.06	0.36	0.03
RC10	0.03	0.20	0.02	0.08	0.02
RC11	0.05	0.34	0.03	0.14	0.03
RC12	3.03	21.78	1.19	5.88	0.45
合计	3.56	24.74	1.41	7.15	0.60
比率	5.93	42.23	2.35	11.92	1.00

5.3.5 小结

本节提出了一种高效、有效的 OAOSMT 构建算法。经过多组标准测试用例的测试，实验结果表明，本节设计的算法在线长和运行时间均取得了最佳效果。它在 VLSI 布线过程中非常实用和有效。未来的工作可能集中在三维绕障问题上。

5.4 X 结构绕障 Steiner 最小树四步启发式算法

表 5.9 和表 5.10 给出了本节的主要符号的描述和主要缩写的描述。

表 5.9 主要符号的描述

符　　号	描　　述
n	引脚顶点的数量
m	障碍物的数量
p_i	第 i 个引脚的序列号
b_i	第 i 个障碍物的序列号

续表

符　　号	描　　述
p_ip_jk	当使用布线选择 k 时，引脚 p_i 和 p_j 的 X 结构边
p_is_k	p_ip_jk 的第一个线段
s_kp_j	p_ip_jk 的第二个线段
rec(p_i，p_j)	p_ip_j2 或 p_ip_j3 的线长
oct(p_i，p_j)	p_ip_j0 或 p_ip_j1 的线长
dis(p_i，p_j)	引脚 p_i 和 p_j 的欧氏距离
$\{B_{ijk}\}$	p_ip_jk 穿过的障碍集
box(t)	t 的直角边界框
$sp(t)$	t 的半周长
L_{ijk}	box($\{B_{ijk}\}$)的半周长
len(p_i，p_j)	引脚 p_i 和 p_j 的绕障线长
os_i	引脚 p_i 的最优拓扑结构
se_i	os_i 的公共边长度

表 5.10　主要缩写的描述

缩　　写	描　　述
SMT	Steiner 最小树
OST	X 结构 Steiner 树
OSMT	X 结构 Steiner 最小树
RSMT	直角结构 Steiner 最小树
OAOST	绕障 X 结构 Steiner 树
OAOSMT	绕障 X 结构 Steiner 最小树
OARSMT	绕障直角结构 Steiner 最小树
OFEMST	不考虑障碍的欧氏最小生成树
DT	德劳内三角剖分
MST	最小生成树
HPWL	半周长线长
EOT	边-障碍表
EST	边-段表
TDST	二维线段树

图 5.19 是 OAOSMT 四步启发式算法实例。输入引脚和障碍物如图 5.19(a)所示。有 4 个引脚和 4 个障碍物。算法可以分为以下 4 个步骤。

步骤 1，OFEMST 构建。在此步骤中，OFEMST 用于构造 X 结构 Steiner 树(OST)的初始解，因为它可以很容易地生成并转换为 OST。首先在给定的引脚顶点上构造 DT(见图 5.19(b))。然后使用 Kruskal 算法生成 OFEMST(见图 5.19(c))。

步骤 2，查找表生成。在此步骤中，将生成两个查找表，用于记录有关 OFEMST 边的连接信息。两个表都可以为整个算法提供信息支持。

步骤 3，绕障策略。在此步骤中，OFEMST 首先根据表查找转换为 OST(见图 5.19(d))。然后选择布线平面上的一些点作为 pseudo-Steiner 点以生成 OAOST。在图 5.19(e)中，s_1 和 c_1 被选择为 p_1 和 p_3 之间的两个 pseudo-Steiner

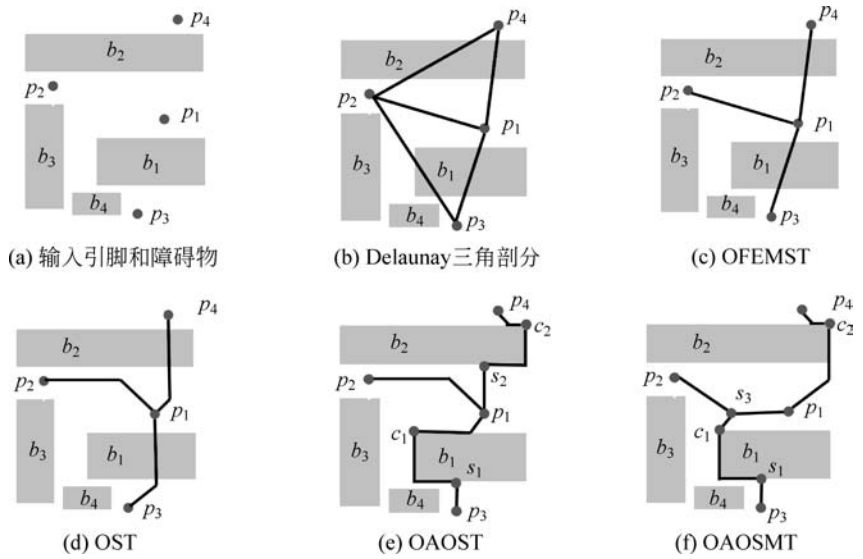


图 5.19 OAOSMT 四步启发式算法实例

点，并且 s_2 和 c_2 被选择为 p_1 和 p_4 之间的两个 pseudo-Steiner 点。

步骤 4，精炼。在该步骤中，采用包括 3 种策略的精炼方法，即冗余 pseudo-Steiner 点消除、pseudo-Steiner 点连接优化和布线选择优化，来生成最终的 OAOSMT。例如，通过共享 s_3 和 p_1 之间的布线路径并移除冗余的 pseudo-Steiner 点 s_2，图 5.19(e)中的 OAOST 被转换为图 5.14(f)中的 OAOSMT。

图 5.20 是 OAOSMT 四步启发式算法流程图。本节根据框架按顺序给出了所提算法的细节。

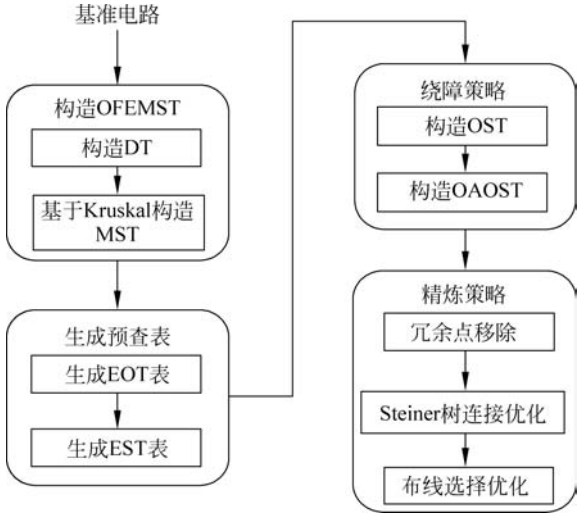


图 5.20 OAOSMT 四步启发式算法流程图

5.4.1　算法细节

1. OFEMST 的构建

OFEMST 通常用于在无障碍平面上构建 OST 的初始骨架，因为它可以很容易地生成并转换为 OST，所以首先研究 OFEMST 构造问题。有许多现成的 OFEMST 构造算法。文献[18]使用绕障约束 DT(Obstacle-Avoiding Constrained Delaunay Triangulation，OACDT)来构造绕障 MST(OAMST)算法。因为 OACDT 包含至少一个 OAMST 并且仅具有 $O(n)$个候选边，因此，本节首先根据给定的引脚构建 DT，然后可以使用 Prim 或 Kruskal 算法在 $O(n)$时间内构造 OFEMST。由于文献[25]的扫描线算法是 Voronoi 图(Voronoi Diagram，VD)构造中的一种非常成熟的技术，在此步骤中直接使用它来构造 VD，然后通过将 VD 转换为对偶图来生成 DT。

2. 查找表的生成

在此步骤中生成两个预先计算的查找表：第一种称为边障碍表(EOT)，记录每条边穿过的障碍物信息；另一个称为边段表(EST)，记录每条边的两个线段的坐标信息。

由于 OFEMST 具有 $n-1$ 条边，并且每条边具有 4 个布线选择，因此总共有 $4(n-1)$条边。对于每一条边 p_ip_jk，记录一个集合$\{B_{ijk}\}$作为 p_ip_jk 穿过的障碍物。同时，计算每个$\{B_{ijk}\}$的边界框，并将 L_{ijk} 值设置为该边界框的 HPWL。所有 $4(n-1)$的$\{B_{ijk}\}$和 L_{ijk} 构成了最终的 EOT。此外，每条边 p_ip_jk 包括两条线段(p_is_k 和 s_kp_i)。记录这两个段的坐标。注意，对于 45°和 135°的线段，绕原点顺时针旋转 45°然后记录新段的坐标。所有 $4(n-1)$的边的信息构成最终 EST。

具体步骤如下。

(1) 初始化 $t=1$。

(2) 检查 OFEMST 的第 t 条边 p_ip_j。针对每个布线选择 k 计算两条分段 p_is_k 和 s_kp_j 的起始坐标和结束坐标。

(3) 对于每个活动障碍物 B_p($0<p<m$)，如果 p_is_k 和 s_kp_j 穿过 B_p，则将 B_p 添加到相应的集合$\{B_{ijk}\}$。

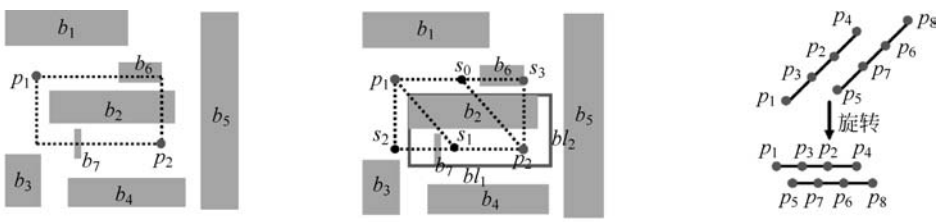
(4) 计算每个$\{B_{ijk}\}$的 L_{ijk}。然后将 L_{ijk} 和$\{B_{ijk}\}$记录到 EOT 中。

(5) 检查每个 p_is_k 和 s_kp_j。如果它是 45°或 135°段，顺时针旋转 45°形成一个新的水平段或竖直段。

(6) 将每条 p_is_k 和 s_kp_j 的坐标记录到 EST 中。然后 $t=t+1$；如果 $t<n$，则返回步骤(2)。否则，退出该过程。

有两点需要注意。首先，在步骤(3)中，对于障碍物 B_p，仅在 B_p 与边界框重叠时检查它。该障碍物也称为“活动障碍物”。图 5.21 是片段旋转的实例，例如，如图 5.21(a)所示，对于边 p_1p_2，只有两个引脚顶点 p_1 和 p_2；b_2、b_6 和 b_7 是 3 个活动障碍物；其余障碍物则不需要检查，因为不可能与边 p_1p_2 的任何布线选择相交。由于二维线段树(Two-Dimensional Segment Tree，TDST)通常用于记录平面

区域的一些属性信息，为了以有效的方式找到给定引脚对的有效障碍物集，采用TDST来组织所有障碍物。更具体地，TDST的外部线段树用于组织水平轴上的间隔，内部分段树用于组织竖直轴上的间隔。另外，根据5.1节中描述的问题模型，每个障碍物 B_p 具有唯一的序列号 p；因此，对于构造的TDST的任何一个区域，如果该区域被障碍物 B_p 覆盖，则将序列号 p 添加到该区域。这样，在将所有障碍物添加到TDST的相应区域之后，最终的TDST将包含关于整个布线平面上的障碍物的所有信息，并且当给出一对引脚顶点 p_i 和 p_j 时，可以通过查询 p_ip_j 的边界框所覆盖的区域快速找到所有活动的障碍物。注意，一旦构建了TDST，它就不需要更新，并且可以为算法的所有后续步骤提供活动障碍物查询。由于线段树通常是完整的二叉树，因此每个操作都可以在对数时间内完成，例如，插入和查询。其次，对于步骤(5)中的45°或135°段采用旋转调整为水平或竖直方向。例如，如图5.21(c)所示，有4个45°段 p_1p_2、p_3p_4、p_5p_6 和 p_7p_8。在计算线长时，需要同时考虑所有4个45°段的位置，并且路径 p_2p_3 和 p_6p_7 应仅计算一次，因为两者都是公共边。由于实际电路的输入规模可能非常大，在布线平面中可能存在相当多的45°段，并且需要相对更多的计算量来获得这些公共边(如果考虑所有可能的45°段对的位置，最坏情况下它可以达到二次复杂度)。但是，如果事先将所有45°段绕原点顺时针旋转45°，则可以轻松计算所有公共边，因为所有生成的段都是水平的(见图5.21(c))，如果两个45°段具有公共边，那么两个生成的水平线段必须具有相同的 y 坐标；通过这种方式，可以首先根据其 y 坐标对这些旋转的段进行排序，然后通过仅扫描已排序的段一次来计算公共边的长度(排序过程主导此过程；它可以在 $O(e\log e)$ 时间内完成，其中 e 是45°段的数量)。135°段的旋转具有类似的原理。



(a) 引脚与障碍物分布情况　(b) 线段分布情况　(c) 选择45°段

图5.21　线段旋转

$$\begin{cases} y_{\text{new}} = \dfrac{\sqrt{2}}{2} \times (y_i - x_i) \\ x_{\text{new1}} = \dfrac{\sqrt{2}}{2} \times (x_i + y_i) \\ x_{\text{new2}} = \dfrac{\sqrt{2}}{2} \times (x_j + y_j) \end{cases} \tag{5.2}$$

$$\begin{cases} x_{\text{new}} = \dfrac{\sqrt{2}}{2} * (x_i + y_i) \\ y_{\text{new1}} = \dfrac{\sqrt{2}}{2} * (y_i - x_i) \\ y_{\text{new2}} = \dfrac{\sqrt{2}}{2} * (y_j - x_j) \end{cases} \tag{5.3}$$

假设 $v_i(x_i, y_i)$ 和 $v_j(x_j, y_j)$ 是 45°或 135°段的两个端点。式(5.2)用于将 45°段旋转到水平段，其中 y_{new} 是新段的 y 坐标，x_{new1} 和 x_{new2} 分别是两个新端点的 x 坐标。类似地，式(5.3)用于将 135°段旋转到竖直段，其中 x_{new} 是新段的 x 坐标，y_{new1} 和 y_{new1} 分别是两个新端点的 y 坐标。

对于 EOT 和 EST 生成，将图 5.21(b)作为一个简单的例子。在一般情况下，假设在 X 结构布线平面中存在两个引脚 $p_1(3,4)$ 和 $p_2(5,3)$ 以及 7 个障碍(b_1～b_7)。显然，在这个例子中，OFEMST 只包含一条边 p_1p_2。表 5.11 显示了 EOT 和 EST 的生成示例。

表 5.11　EOT 和 EST 的生成示例

EOT	EST
$\{B_{120}\}=\{b_2\}, L_{120}=\text{sp}(b_2)$	$p_1s_0\{(3,4),(4,4)\}$ $s_0p_2^*\{(4\sqrt{2},-\sqrt{2}),(4\sqrt{2},0)\}$
$\{B_{121}\}=\{b_2, b_7\}, L_{121}=\text{sp}(\text{box}(\{B_{121}\}))$	$p_1s_1^*\{(7/\sqrt{2},-\sqrt{2}/2),(7/\sqrt{2},\sqrt{2}/2)\}$ $s_1p_2\{(4,3),(5,3)\}$
$\{B_{122}\}=\{b_7\}, L_{122}=\text{sp}(b_7)$	$p_1s_2\{(3,4),(3,3)\}$ $s_2p_2\{(3,3),(5,3)\}$
$\{B_{123}\}=\{b_2, b_6\}, L_{123}=\text{sp}(\text{box}(\{B_{123}\}))$	$p_1s_3\{(3,4),(5,4)\}$ $s_3p_2\{(5,4),(5,3)\}$

可以快速查询有关 OFEMST 的所有连接信息。例如，从表 5.11 可以看出，边 p_1p_21 穿过障碍物 b_2 和 b_7。两条分段 p_1s_1 和 s_1p_2 的坐标分别为{$(7/\sqrt{2}, -\sqrt{2}/2)$、$(7/\sqrt{2},\sqrt{2})/2$}和{(4,3),(5,3)}。

应该注意的是，$p_1s_1^*$ 表示 p_1s_1 旋转后的线段坐标。另外，如果一条边 p_ip_jk 可以避开所有障碍物($\{B_{ijk}\}=\varnothing$)，那么 $L_{ijk}=0$。如果 p_ip_jk 只穿过一个障碍物 $b(\{B_{ijk}\}=\{b\})$，则 L_{ijk} 直接设置为 b 的 HPWL。否则，如果 p_ip_jk 经过多个障碍物，例如，p_1p_21 通过图 5.21(b)中的 b_2 和 $b_7(\{B_{ijk}\}=\{b_2,b_7\})$，则 L_{121} 设置为边界框的 HPWL($\{B_{121}\}$)，等于 bl1+bl2，如图 5.21(b)中实线矩形框所示。

两个查找表的功能可以总结如下。

(1) 在绕障策略中，转换过程可以根据 EOT 提供的信息有效地执行。

(2) 在精炼过程中，基于 EOT 和 EST 可以有效地计算各个端点的最优结构。

此外，通过查找表，可以获得所需的关于冗余 pseudo-Steiner 点消除和 pseudo-Steiner 连接优化过程的所有连接信息。

(3) 利用 EST 可以有效地计算局部和全局的线长。

3. 绕障策略

1) OST 生成

在该部分中，基于查找 EOT，将步骤 1 中生成的 OFEMST 转换为 OST。

检查 OFEMST 的每条边 p_ip_j。如果边 p_ip_j0 或 p_ip_j1 可以避开所有障碍物，直接将 p_ip_j 的布线选择设置为选择 0 或选择 1。否则，如果 p_ip_j2 或 p_ip_j3 可以避开所有障碍物，仍然设置(根据 L_{ij0} 和 L_{ij1} 的值)选择 0 或选择 1。如果所有 4 个布线选择都通过障碍物，选择具有最小 L_{ijk} 的布线选择。应该注意的是，所有判断都可以通过直接查找 EOT 来执行。因此，这个过程非常快。OST 生成的伪代码在算法 5.8 中给出。

算法 5.8 OST 生成

```
输入：OFEMST，EOT
输出：OST
  for each edge p_ip_j of OFEMST do
    if {B_ij0}=∅ then //如果 X 结构边 p_ip_j0 可以避开所有障碍物
      p_ip_jk=p_ip_j0;//将 p_ip_j 的布线选择设置为选择 0
    end if
    else if {B_ij1}=∅ then //如果 X 结构边 p_ip_j1 可以避开所有障碍物
      p_ip_jk=p_ip_j1; //将 p_ip_j 的布线选择设置为选择 1
    end if
    else if{B_ij2}=∅ or {B_ij3}=∅ then//如果 p_ip_j2 或 p_ip_j3 可以避开所有障碍物
      //根据 L_ij0 和 L_ij1 的值设置选择 0 或选择 1
      if L_ij0<= L_ij1 then
        p_ip_jk=p_ip_j0;
      else
        p_ip_jk=p_ip_j1;
      end if
    else//如果所有 4 个布线选择都通过障碍物
      select_min(L_ijt), t=0,1,2,3;//选择具有最小 L_ijt 的布线选择
      p_ip_jk=p_ip_jt;
    end if
  end for
```

具体步骤如下。

(1) 初始化 $t=1$。

(2) 对于 OFEMST 的第 t 条边 p_ip_j，检查选择 0 和选择 1；如果它们中的任何一个可以避免所有障碍物，那么将 p_ip_j 的布线选择设置为当前的布线选择。重复此步骤以检查第$(t+1)$条边，直到 $t+1\geqslant n$；否则，转到步骤(3)。

(3) 如果选择 2 或选择 3 可以避免所有障碍物，请转到步骤(4)；否则，转步骤(5)。

(4) 如果 $L_{ij0}<L_{ij1}$，则将 p_ip_j 的布线选择设置为 0。否则，将其设置为选择 1，然后 $t=t+1$；如果 $t<n$，转到步骤(2)；否则，退出流程。

(5) 选择最小的 $p_ip_j(k=0,1,2,3)$，并将 p_ip_j 的布线选择设置为选择 k。然后 $t=t+1$；如果 $t<n$，转到步骤(2)；否则，退出流程。

步骤(2)采用策略，在 OST 生成过程中优先选择 0 和选择 1(见定理 5.3)。另外，步骤(3)和步骤(4)意味着即使选择 2 或选择 3 可以绕开障碍物，仍然选择布线选择 0 或选择 1 作为结果(见定理 5.4)。

定理 5.3　对于任意两个端点 p_ip_j，相比于 $\mathrm{rec}(p_i,p_j)$，$\mathrm{oct}(p_i,p_j)$可使线长减少$(2-2\sqrt{2})\times\min(\Delta x,\Delta y)$。

证明：假设 $\Delta x<\Delta y$，$\mathrm{oct}(p_i,p_j)=\Delta y-\Delta x+\sqrt{2}\times\Delta x$，并且 $\mathrm{rec}(p_i,p_j)=\Delta x+\Delta y$。这样，$\mathrm{rec}(p_ip_j)-\mathrm{oct}(p_ip_j)=(2-2\sqrt{2})\times\Delta x$。对于 $\Delta x>\Delta y$ 的情况也可同样证明。

引理 5.5　c 是边 p_ip_j 的中间节点，则 $\mathrm{rec}(p_i,p_j)=\mathrm{rec}(p_i,c)+\mathrm{rec}(c,p_j)$，当且仅当 $c\subset\mathrm{box}(p_i,p_j)$。

引理 5.6　c 是边 p_ip_j 的中间节点，然后 $\mathrm{oct}(p_ip_j)=\mathrm{oct}(p_i,c)+\mathrm{oct}(c,p_j)$，当且仅当 $c\subset\mathrm{para}(p_ip_j)$，其中 $\mathrm{para}(p_ip_j)$是由 p_i 和 p_j 形成的具有 45°或 135°方向的平行四边形。

根据 p_i,c 和 c,p_j 形成的分段之间的平行关系，可以很容易地实现引理 5.5 和引理 5.6 中的等式。因此，这里省略了证明过程。

定理 5.4　对于每条边 p_ip_j，如果$\{B_{ij0}\}\neq\varnothing$且$\{B_{ij1}\}\neq\varnothing$，但$\{B_{ij2}\}\neq\varnothing$或$\{B_{ij2}\}\neq\varnothing$，则在框($\{B_{ij0}\}$)或框($\{B_{ij1}\}$的边界上选择一些 pseudo-Steiner 点作为中间节点可以使线长减少$(2-\sqrt{2})\times\min(\Delta x,\Delta y))$。

证明：图 5.22 是定理 5.4 实例，$\{B_{ij0}\}=\{B_{ij1}\}=\{b_1,b_2\}$，但$\{B_{ij2}\}=\varnothing$。但是，如图 5.22(b)所示，如果选择 pseudo-Steiner 点 c 作为中间节点，将得到 $\mathrm{oct}(p_j,c)\leqslant\mathrm{rec}(p_j,c)$和 $\mathrm{oct}(c,p_j)\leqslant\mathrm{rec}(c,p_j)$，当且仅当 p_i、c 和 p_j 共线时，等式才成立。根据引理 5.5，如果 $c\subset\mathrm{box}(p_ip_j)$，则 $\mathrm{rec}(p_j,c)+\mathrm{rec}(c,p_j)=\mathrm{rec}(p_i,p_j)$。因此，有 $\mathrm{oct}(p_i,c)+\mathrm{oct}(c,p_j)\leqslant\mathrm{rec}(p_ip_j)$。最好情况下，根据定理 5.3 得到$(2-\sqrt{2})\times\min(\Delta x,\Delta y)\times(\mathrm{oct}(p_i,c)+\mathrm{oct}(c,p_j))=\mathrm{rec}(p_ip_j)$。

注意，在最坏的情况下，p_ic 和 cp_j 都可以通过仅使用选择 2 或选择 3 来避免所有障碍，然后线长减小为零。

2) OAOST 生成

显然，根据 OST 生成方法，OST 的某些边可能会穿过障碍物。实际上，可以通过直接查找 EOT 来识别这些边。但它可能需要更多的技术支持以绕开障碍物。

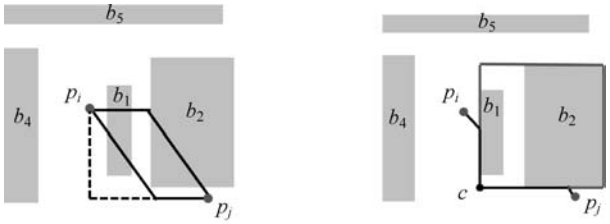


图 5.22 定理 5.4 实例

因此，提出了一种 pseudo-Steiner 点选择方法，它可以有效地帮助所有边绕开障碍物。

参照图 5.23(a)所示的每个点 p，将平面划分为 8 个区域；每个区域不包括图 5.23(b)所示的两条边界线。显然，图 5.23(a)中的每条边界线($d_1 \sim d_8$)是布线平面中点 p 的合法布线路径。

在选择 pseudo-Steiner 点之前，首先扫描 OST 的所有边，对于每条边 p_ip_jk(k 是 OST 中的布线选择)；如果 $x_i > x_j$，则改变点 p_i 和 p_j 的位置，即将 p_ip_jk 变换为 $p_ip_jk^*$，其中相应的 k^* 值如图 5.23(c)所示。这样，对于每条边，可以确保起点位于终点的左侧，从而简化了边的可能场景。在变换之后，当检查边 p_ip_jk 时，只需要考虑点 p_i 的 X 形分区的右半轴和点 x_j 的 X 形分区的左半轴。

接下来，检查转换后的 OST 的每条边 p_ip_jk；如果它穿过了障碍物，则直接删除 p_ip_jk，并计算$\{B_{ijk}\}$的直线边界框 bx。然后沿着 5 个方向将 p_i 投影到它的右半轴。图 5.23 是合法路径划分这 5 个方向，包括图 5.23(a)中的 $d1 \sim d5$。假设 p_id_x 和 bx 之间的第一个交点是 t_x($x=1,2,3,4,5$)；应该注意，t_x 可能不存在，因为 p_id_x 没有与 bx 相交。此外，计算直线 p_id_x 和 p_ip_j 之间的角度 a_x，然后选择具有最小 a_x 的 t_x 作为 pseudo-Steiner 点，因为 s 可能不能直接连接到 p_i，所以需要选择更多的 pseudo-Steiner 点，以便 p_i 和 p_j 可以连接。为了实现这个目标，检查 bx 的每个拐角点 c_y($y=1,2,3,4$)，并选择一个使得 $\mathrm{rec}(s,c_y)+\mathrm{rec}(c_y,p_j)$ 最小且具有较大的 $\mathrm{rec}(s,c_y)$的拐点作为另一个 pseudo-Steiner 点 c。最后，计算边 p_is、sc 和 cp_j 的连接信息，并将这些信息添加到 EOT 和 EST 中。此时，可以通过直接查找表来生成边 p_isk_1、sck_2 和 cp_jk_3。

考虑到集合$\{B_{ijk}\}$的障碍物可能在布线平面中广泛分散，在这种情况下，p_i 和 p_j 之间新生成的布线路径可能很长，因为它沿着$\{B_{ijk}\}$的边界框绕行。因此，为了确保高布线质量，在以下两种情况下采用其他 pseudo-Steiner 点选择策略。

(1) 两个选定的 pseudo-Steiner 点之间的布线路径穿过障碍物。

(2) 如果将 p_i 和 p_j 之间新生成的路径的长度除以 $\mathrm{rec}(p_i,p_j)$并且得到的值大于 β(在实验中 β 设置为 2)。然后，根据 p_i 与障碍物之间的距离按递增顺序对$\{B_{ijk}\}$的所有障碍物进行排序，并根据上述投影方法按顺序选择每个障碍物上的

pseudo-Steiner 点。以这种方式，生成可以从$\{B_{ijk}\}$的大边界框逃逸的绕障路径。

OAOST 生成技术的详细步骤如下。

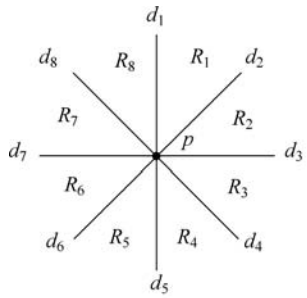
(1) 对于 OST 的每个边 p_ip_jk，如果 $x_i>x_j$，则将 p_ip_jk 变换为 $p_ip_jk^*$。

(2) 初始化 $t=1$。

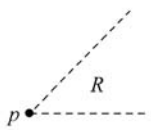
(3) 对于变换后的 OST 的第 t 个直线边 p_ip_jk，如果它穿过障碍物，则转到步骤(4)。否则，继续检查第$(t+1)$个边沿，直到 $t+1\geqslant n$。

(4) 删除边 p_ip_jk。计算 bx=box($\{B_{ijk}\}$)。沿 $d1\sim d5$ 投影 p_i 并计算每个 p_id_x($x=1,2,3,4,5$)和 bx 之间的第一个交点 t_x。

(5) 计算每个 p_id_x 和直线 p_ip_j 之间的角度 a_x。然后选择具有最小 a_x 的 t_x 作为 pseudo-Steiner 点 s。



(a) 平面划分



(b) 区域边界线

k	k^*
0	1
1	0
2	3
3	2

(c) k值对应表

图 5.23　合法路径划分

(6) 从 bx 中选择一个角点 c，使 rec(s,c_y)+rec(c_y,p_j)最小，其中 c_y 是 bx 的第 y 个角点，$y=1,2,3,4$。

(7) 计算边 p_is、sc 和 cp_j 的连接信息，生成边 p_is、sc 和 cp_j。

(8) 如果(len(p_i,s)+len(s,c)+len(c,p_j))/rec(p_i,p_j)$>\beta$ 或边穿过障碍物，则对$\{B_{ijk}\}$排序，并按顺序在每个障碍物上选择 pseudo-Steiner 点，并生成从 p_i 到 p_j 的新路径。

引理 5.7　对于 OST 的边 p_ip_jk，如果$\{B_{ijk}\}\neq\varnothing$，则投影($p_i$)$\cap$box($\{B_{ijk}\}$)$\neq\varnothing$或投影($p_j$)$\cap$box($\{B_{ijk}\}$)$\neq\varnothing$，其中投影($p_i$)和投影($p_j$)分别是 p_i 和 p_j 沿其 5 个方向在其半轴上的投影。

3) 证明

使用归谬法来证明这个引理。图 5.24 是引理 5.7 实例，图 5.24(a)和图 5.24(b)分别显示了根据定义的 p_i 和 p_j 的半轴。如果投影(p_i)$\cap$box($\{B_{ijk}\}$)$=\varnothing$，则 box($\{B_{ijk}\}$)必须完全位于 p_i 右半轴的一个区域。类似地，如果投影(p_j)$\cap$box($\{B_{ijk}\}$)$=\varnothing$，则 box($\{B_{ijk}\}$)必须完全位于 p_j 的左半轴的一个区域中。因此，如果投影(p_i)$\cap$box($\{B_{ijk}\}$)$=\varnothing$和投影(p_j)$\cap$box($\{B_{ijk}\}$)$=\varnothing$都是真的，那么在不失一般性的情况下，假设边界框($\{B_{ijk}\}$)$\subset R_i$($R_i\in\{R_1,R_2,R_3,R_4\}$ of

p_i)和 box($\{B_{ijk}\}$)$\subset R_i$($R_i \in \{R_5, R_6, R_7, R_8\}$of p_j),R_i 和 R_j 的两个边界线必须形成一个合法的 X 结构布线路径,这意味着 p_i 和 p_j 可以直接连接而不会遇到任何障碍,这是矛盾的。例如,如图 5.24(c)所示,p_i 的框($\{B_{ijk}\}$)$\subset R$ 和 p_j 的框($\{B_{ijk}\}$)$\subset R_7$,p_i 和 p_j 可以通过路径 $p_i s_1 p_j$ 或 $p_i s_2 p_j$ 连接。

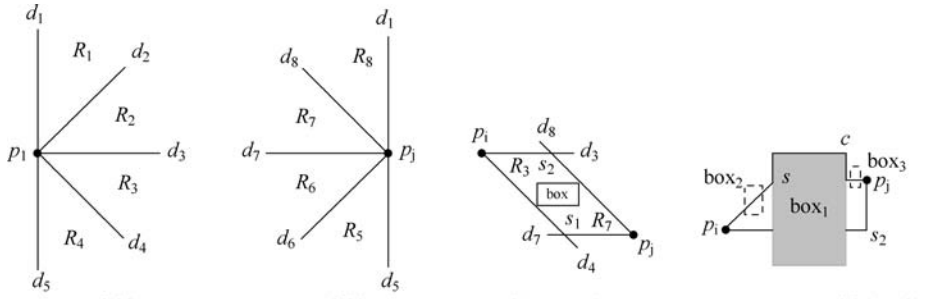


图 5.24 引理 5.7 实例

上面已经指出 t_x 可能不存在,因为 $p_i d_x$ 在步骤(4)中不与 bx 相交。因此,可能存在所有 5 个交叉点 t_x 都不存在的情况。在这种情况下,根据引理 5.7,将沿着图 5.24(a)中的 5 个方向(d_1,$d_5 \sim d_8$)将 p_j 投影到其左半轴。此外,由于没有考虑其他障碍物,p_i 和 p_j 之间新生成的路径仍可能遇到障碍。因此,这种 OAOST 生成技术可以多次操作,直到所有的边都避开所有障碍物。幸运的是,当新生成的路径穿过障碍物时,这些障碍物的直角边界框的至少两个边界通常被限制在一定范围内。例如,如图 5.24(d)所示,p_i 和 p_j 通过选择 3 连接,并且它穿过 box_1,s 和 c 被选为两个 pseudo-Steiner 点。由于已经确保 s 和 c 之间的路径可以避开所有障碍,因此只有路径 $p_i s$ 和 cp_j 仍然穿过障碍物。假设 box_2 是路径 $p_i s$ 穿过的障碍物的边界框。box_2 的底部必须位于 $p_i s_2$ 上方,box_2 的右侧受 box_1 右侧的限制。box_3 也受 $p_i s_2$ 和 box_1 的限制。显然,这种情况可以在恒定的时间内解决。其他 3 个选择遵循类似的原则。类似地,当在每个障碍物上选择 pseudo-Steiner 点时,新生成的路径的边界框也在至少两个方向上受到限制。它也可以在恒定的时间内解决。应该注意,该 OAOST 生成过程也会生成冗余的 pseudo-Steiner 点。

以图 5.25 作为一个简单的例子来进一步解释这个过程。图 5.25 是 OAOST 构建实例,图 5.25(a)是原始布线图,其中 $p_1 p_2$ 穿过障碍物 b_1、b_2 和 b_3(即$\{B_{120}\} = \{b_1, b_2, b_3\}$)。删除边 $p_1 p_2$ 后,首先计算框($\{B_{120}\}$);结果 bx 在图 5.25(b)中用实线矩形框显示。然后沿着它的 5 个方向投射 p_1。很明显,$p_1 d_1$ 和 $p_1 d_5$ 不与 bx 相交,因此,p_1 沿其 5 个方向投射得到与 bx 的交叉点集 $T_1 = \{t_2, t_3, t_4\}$。接下来,计算直线 $p_1 p_2$ 和 p_1 的每条投影线之间的角度,得到 $a_2 = \angle d_2 p_1 p_2$、$a_3 = \angle d_3 p_1 p_2$、$a_4 = \angle d_4 p_1 p_2$。因为 $a_4 < a_3 < a_2$,所以选择 t_4 作为 pseudo-Steiner 点 s。在图 5.25(c)中,检查了 bx 的 4 个角点;有 $\mathrm{rec}(sc_2) + \mathrm{rec}(c_1 p_2) = \mathrm{rec}(sc_2) +$

$\mathrm{rec}(c_2p_2)<\mathrm{rec}(sc_3)+\mathrm{rec}(c_3p_3)=\mathrm{rec}(sc_4)+\mathrm{rec}(c_4p_2)$，因为 $\mathrm{rec}(sc_1)<\mathrm{rec}(sc_2)$，$c_2$ 被选为另一个 pseudo-Steiner 点。最后，计算并记录边 p_1s、sc_2 和 c_2p_2 的连接信息到 EOT 和 EST 中。最终的边可以通过表查找生成（见图 5.25(d)）。注意，生成的 OAOST 可能包括一些有进一步优化的空间的布线路径。

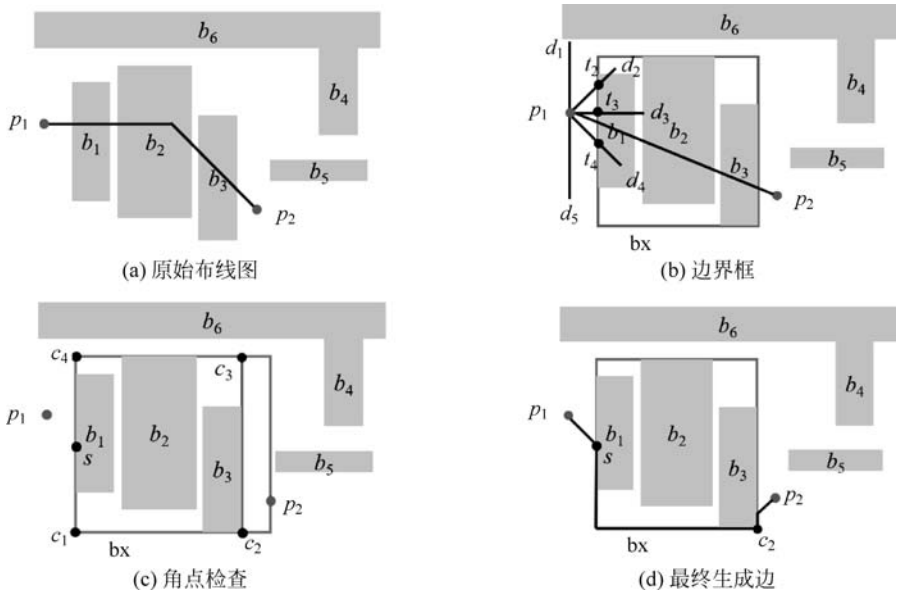


图 5.25　OAOST 构建实例

4. 精炼

由于生成的 OAOST 可能仍然包括一些次优的布线路径，因此在该部分中，提出了 3 种精炼策略以进一步减少线路长度。

1）冗余 pseudo-Steiner 点消除

在 OAOST 生成过程中，可以根据 OAOST 生成方法引入一些冗余的 pseudo-Steiner 点。例如，如图 5.19(e)所示，对于边 $p_1p_4$1，选择 s_2 和 c_2 作为两个 pseudo-Steiner 点。但是，s_2 是多余的，因为 p_1 和 c_2 可以直接连接，而 $\mathrm{len}(p_1s_2)+\mathrm{len}(s_2c_2)>\mathrm{len}(p_1c_2)$，因此，$s_2$ 被删除，并且 p_1 连接到图 5.19(f)中所示的 c_2。为了消除这些冗余点，在 OAOST 中扫描每个选定的 pseudo-Steiner 点 s 并检查它的两条边 p_isk_1 和 sp_jk_2，如果 p_ip_j 可以直接连接并且 $\mathrm{len}(p_is)+\mathrm{len}(sp_j)\geqslant\mathrm{len}(p_ip_j)$，删除 pseudo-Steiner 点 s 和相关边 p_isk_1 和 sp_jk_2，然后将 p_i 连接到 p_j。注意，原始边 p_isk_1 和 sp_jk_2 的连接信息可以通过查找表来获得，因此，这个过程也非常快。

2）pseudo-Steiner 点连接优化

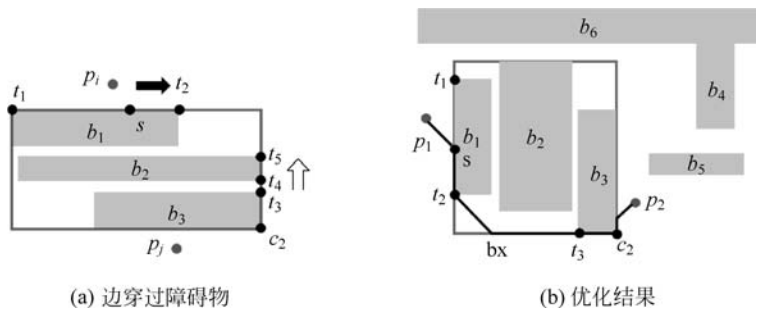
尽管 OAOST 生成技术可以使 OST 边避免所有障碍物，但 pseudo-Steiner 点

之间的布线路径可能不够好。例如，图 5.25(d)中 s 和 c_2 之间的布线路径可以进一步优化，因为在障碍物的直角边界框中存在一些未使用的布线资源。当然，如果 OST 边仅穿过一个障碍物，或者在每个障碍物上选择了 pseudo-Steiner 点，则所选择的 pseudo-Steiner 点之间的布线路径没有优化空间。另一方面，如果两个选定的 pseudo-Steiner 点共线，则最佳布线路径是直线，并且也不能优化。因此，对于任何一对 pseudo-Steiner 点，如果可以进一步优化，它至少应满足两点：

(1) 原始 OST 边穿过多个障碍物，并且在边界上选择 pseudo-Steiner 点；

(2) 两个选定的 pseudo-Steiner 点不共线。

对于这种 pseudo-Steiner 点，使用一种称为滑动的操作来尝试减少线长。由于两个选定的 pseudo-Steiner 点位于直角边界框的两侧，因此必须在此边界框的两侧存在合法的布线路径。另外，边界框的两侧必须与至少一个障碍物线接触。图 5.26 是对 pseudo-Steiner 点连接优化，如图 5.26(a)所示，p_i 和 p_j 之间的边穿过障碍物 b_1、b_2 和 b_3。根据 OAOST 生成技术，选择点 s 和角 c_2 作为两个 pseudo-Steiner 点。在这种情况下，s 和 c_2 只能通过选择 3 连接。此外，b_1、b_2 和 b_3 分别与 t_1t_2、t_4t_5 和 c_2t_3 之间的直角框线接触。然而，如果沿着 s 和 c_2 之间的布线路径滑动两个 pseudo-Steiner 点并且在线接触的障碍角 t_5 处停止每个点，则 t_5 是距离原始 pseudo-Steiner 点最远的一个。然后，这两个角可以通过选择 0 或选择 1 连接，因为可以使用未使用的布线资源。例如，将 s 滑动到 t_2 和 c_2 到 t_5，然后 t_2 和 t_5 可以形成一个边 t_2t_50。因此，与先前的结果相比，线长减小。图 5.26(b)显示了图 5.25(d)中的结果优化后的布线图。当然，当两个所选择的 t_5 之间存在其他障碍物，或者这两个 t_is 彼此重叠时，也没有优化线长的空间。



(a) 边穿过障碍物　　(b) 优化结果

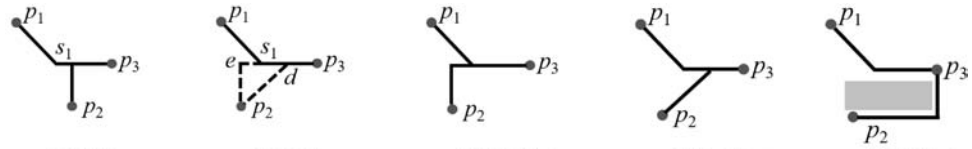
图 5.26　pseudo-Steiner 点连接优化

3) 布线选择优化

除了前两点，还没有考虑一个重要问题，即公共边。换句话说，可以进一步优化所获得的布线树。对于 OAOST 的任何一个引脚 p_i，其互连至少有一个最佳结构。

通过只讨论 2 度引脚，其他度的引脚遵循相似的原理。由于每条边有 4 个布线选择，对于一个 2 度引脚，总共有 16 个子结构。图 5.27(a)显示了无障碍平面中

的 2 度端点 p_3，其中 p_1 和 p_2 连接到 p_3。在不失一般性的情况下，假设两个引脚之间存在 $\Delta x > \Delta y$。图 5.27 是布线选择优化的实例，当 p_2 在 s_1 和 p_3 之间时，其中 s_1 是边 $p_1p_3$1 的 pseudo-Steiner 点，图 5.27(a)是 p_3 的唯一最佳结构。当 p_2 在 p_1 和 s_1 之间(见图 5.27(b))时，如果 $p_2e + es_1 < p_2d$ 或 $es_1 > ed$，则图 5.27(c)成为最佳结构。否则，图 5.27(d)是最优的。障碍物存在的平面也遵循类似的原理。但是由于障碍物的存在，最优结构可能成为 16 个子结构中的任意一个。例如，如图 5.27(e)所示，p_2 只能通过选择 3 连接到 p_3。因此，图 5.27(e)是存在障碍物时的最佳结构。当然，p_1 也可以通过图 5.27(e)中的选择 0 连接到 p_3。总之，无论引脚有几度，其互连至少有一个最佳结构。



(a) 互连拓扑一　(b) 互连拓扑二　(c) 互连拓扑三　(d) 互连拓扑四　(e) 互连拓扑五

图 5.27　布线选择优化实例

基于这一事实，提出了一种面向引脚的布线选择优化方法。首先，计算每个引脚 p_i 的度以及与其连接的引脚列表。之后算法计算每个引脚 p_i 的最佳结构(os_i)，即 p_i 所有布线选择组合中具有线长最佳同时满足绕障的子结构。此外，计算每个 os_i 的公共边(se_i)的长度最后，根据 se_i 的值按降序对所有引脚进行排序，并用各引脚的 os_i 代替其原来的布线选择组合，以构建 OAOST。详细步骤如下。

(1) 扫描 OAOST 的边以计算每个引脚 p_i 的度，并同时记录连接到 p_i 的引脚作为列表。

(2) 对于每个引脚 p_i，如果 p_i 的度数是 d，则枚举 p_i 的所有 4^d 的布线选择组合。然后选择具有最小线长的且避开所有障碍物的那一个组合作为 os_i，同时计算每个 os_i 的 se_i。

(3) 根据 se_i 按递减顺序对 OAOST 的所有引脚进行排序。

(4) 对于每个引脚 p_i 按顺序，将 os_i 的布线选择组合应用于 OAOST，直到所有 OAOST 边都已确定。

有两点需要注意，第一，在步骤(2)中，可以通过直接查找 EOT 来确定布线选择组合是否避免所有障碍物。第二，在步骤(4)中，当前 os_i 不能改动之前已调整过的子结构。此外，可以通过直接查找 EST 来计算局部线长和总线长。

5.4.2　复杂性分析

定理 5.5　该算法的时间复杂度为 $O((m+n)\log x_m \log y_m + nm\log m)$，其中 n 和 m 分别是布线平面的顶点数和障碍物数，x_m 和 y_m 分别是布线平面的最大 x 坐标和 y 坐标。

证明：在步骤 1 中，可以在 $O(n\log n)$时间中生成 DT。然后可以使用 Kruskal 算法在 $O(n)$时间生成 OFEMST，因为 DT 中只有 $O(n)$个边。

在步骤 2 中，TDST 可以以 $m\times(\log x_m \log y_m)$时间构造。其中包括两个 for 循环，由于有 n 条边，所以 $O(n)$支配第一个循环。对于第二个循环，其时间复杂度为常数 4，即布线选择的数量。并且第二个循环内的每个查询都可以在 $O(\log x_m \log y_m)$时间内执行。因此，步骤 2 的时间复杂度为 $O((m+n)*(\log x_m \log y_m))$。

在步骤 3 中，OST 生成需要 $O(n)$时间，因为存在 n 个边并且表查找仅需要 $O(1)$时间。对于 OAOST 生成过程，外部 for 循环由 n 控制。对于每个 OST 边，如果它穿过障碍物，在最坏情况下找到边界框需要 $O(m)$时间。如果在每个障碍物上选择 pseudo-Steiner 点，则对障碍物排序主导该过程，在 $O(m\log m)$时间内完成执行。此外，当生成新路径的连接信息时，每个查询需要 $O(\log x_m \log y_m)$时间。因此，步骤 3 的时间复杂度是 $O(n\times(m\log m+\log x_m \log y_m))$。

在步骤 4 中，由于存在 n 条边并且需要计算新边的连接信息，冗余 pseudo-Steiner 消除和 pseudo-Steiner 连接优化都需要 $O(n(\log x_m \log y_m))$时间。在布线选择优化过程中，扫描 OAOST 的每条边需要 $O(n)$时间。当计算每个引脚 p_i 的 os_i 时，两个 for 循环分别由 n 和 4^d 支配。另外，所有引脚都可以使用快速排序算法在 $O(n\log n)$时间内进行排序。而将每个引脚的最优结构应用于 OAOST 也需要 $O(n)$时间，因此，布线选择优化由 $O(n\log n)$支配。步骤 4 的时间复杂度是 $O(n(\log x_m \log y_m+\log n))$。

总之，在最坏情况下，该算法的时间复杂度是 $O((m+n)\log x_m \log y_m+nm\log m)$。

5.4.3 实验结果

所有算法都已用 C 语言实现和执行。此外，MATLAB 被用于模拟最终的布线图。实验在具有 2.9GHz CPU 和 2GB 存储器的 PC 上进行。共有 17 个基准电路。ind1～ind5 是 Synopsys 的工业测试用例。rc01～rc12 是绕障问题的基准。另外，一些随机生成的电路用于进一步测试所提出的算法的可扩展性。在所有这些测试案例中，引脚和障碍物的数量分别为 10～1000 和 10～10 000。

1. 精炼策略的有效性

精炼过程对于算法的最终布线质量非常重要。它通过尽可能多地增加公共边的线长来构建最终的 OAOSMT，同时充分利用布线资源生成更多的对角线段。为了研究优化线长的精炼过程的有效性，通过比较使用该方法之前和之后的线长来说明精炼的效果。表 5.12 显示了精炼前后的线长比较结果。从表 5.12 可以看出，在使用精炼后可以实现 2.74%～9.76%的线长减小。此外，平均线长减少也可达到 6.16%。

表 5.12　精炼前后的线长比较结果

测试用例	引脚数	障碍物数	精炼前	精炼后	优化率/%
ind1	10	32	584	568	2.74
ind2	10	43	10 066	9548	5.15
ind3	10	59	601	574	4.49
ind4	25	79	1179	1069	9.33
ind5	33	71	1402	1334	4.85
rc01	10	10	27 716	25 084	9.50
rc02	30	10	43 758	39 488	9.76
rc03	50	10	56 048	54 177	3.34
rc04	70	10	64 011	59 643	6.82
rc05	100	10	79 844	72 738	8.90
rc06	100	500	83 842	77 592	7.45
rc07	200	500	113 528	105 480	7.09
rc08	200	800	122 917	113 110	7.98
rc09	200	1000	120 048	110 642	7.84
rc10	500	100	161 346	155 579	3.57
rc11	1000	100	222 132	216 401	2.58
rc12	1000	10 000	726 837	702 544	3.34
均值					6.16

2. 与最好的 OAOSMT 算法对比

到目前为止，文献[28]的算法是解决 OAOSMT 问题的最佳方案。其提出了一个基于 PSO 的框架，可以在合理的时间内产生出色的 OAOSMT。为了验证 OAOSI 构造的快速四步启发式算法(a Fast Four-Step Heuristic for Obstacle-Avoiding Octilinear Steiner Tree Construction，FH-OAOS)的有效性，将本节算法与文献[28]提出的算法在上述基准电路中进行了比较，表 5.13 显示了两个算法的线长和运行时间的比较结果。从实验数据可以观察到，当输入电路的规模很大时(即具有许多障碍的较大规模网)，FH-OAOS 产生了更好的结果。例如，对于测试用例 rc06～rc12，FH-OAOS 的线长小于文献[28]的线长。此外，从表 5.13 可以看出，与文献[28]相比，可以减少 －4.18%～3.10% 的线长。平均而言，FH-OAOS 的表现与文献[28]中所述表现相当。FH-OAOS 的线长仅比文献[28]提及的线长长 0.36%。分析这个结果主要有两个原因。第一，当输入电路的规模很小时(即只有少量障碍物的小规模网络)，文献[28]可以通过适度增加 PSO 算法的迭代次数来搜索出色的结果，同时保持合理的运行时间。此外，文献[28]引入了两个遗传算法的算子，可以进一步提高 PSO 算法的搜索能力。第二，当输入电路的规模变大时，如果在文献[28]中适度增加迭代次数，效果有限。如果在文献[28]中大大增加迭代次数，虽然它可能会得到很好的结果，但运行时间将变得不可接受；因此，需要在线长和运行时间之间进行折中。

表 5.13 的第 7 列和第 8 列中提供了两种算法的运行时间。尽管文献[28]的算法得到的线长结果看起来相当不错，但其运行时间并不令人满意。这主要是由于该算法的性质，它是一种基于粒子群的算法，需要更多的迭代次数来搜索解空间。此外，文献[28]的算法中候选边数量是 $O(n^2)$。因此，它需要更多的时间来计算这种边信息。表 5.13 的最后一行在 FH-OAOS 上标准化。可以看出，FH-OAOS 算法速度很快。例如，rc12 包括 1000 个引脚顶点和 10 000 个障碍物，FH-OAOS 能在仅仅 0.41s 内产生很好的效果，而文献[28]的算法需要 13.12s。平均而言，FH-OAOS 比文献[28]的方法快 66.39 倍。因此在工业生产中更实用。

表 5.13 与文献[28]的线长和运行时间的比较结果

测试用例	引脚数	障碍物数	线长		优化率/%	CPU 时间/s	
			FH-OAOS	[28]		FH-OAOS	[28]
ind1	10	32	568	562	−1.07	0.00	0.02
ind2	10	43	9548	9431	−1.24	0.00	0.02
ind3	10	50	574	574	0.00	0.00	0.02
ind4	25	79	1069	1033	−3.48	0.00	0.02
ind5	33	71	1334	1288	−3.57	0.00	0.05
rc01	10	10	25 084	24 717	−1.34	0.00	0.02
rc02	30	10	39 488	40 751	3.10	0.00	0.02
rc03	50	10	54 177	52 033	−4.12	0.00	0.10
rc04	70	10	59 643	57 250	−4.18	0.00	0.17
rc05	100	10	72 738	72 738	0.00	0.00	0.26
rc06	100	500	77 592	78 643	1.34	0.02	0.57
rc07	200	500	105 480	105 542	0.06	0.02	1.91
rc08	200	800	113 110	116 204	2.66	0.03	2.93
rc09	200	1000	110 642	111 385	0.07	0.03	3.20
rc10	500	100	155 579	157 520	1.23	0.02	5.42
rc11	1000	100	216 401	219 037	1.20	0.03	9.33
rc12	1000	10 000	702 544	724 425	3.02	0.41	13.12
均值(优化率(%))/合计(时间)					−0.36	0.56	37.18
比率(时间)					—	1.00	66.39

3. 与 λ-Geometry 算法对比

下面对 FH-OAOS 与文献[18]提出的 λ 几何算法进行比较。根据定义，当 λ 设置为 2 时，文献[18]的算法可以生成 OARSMT；当 λ 设定为 4 时，文献[18]的算法可以生成 OAOSMT。表 5.14 显示了线长和运行时间的比较结果。可以看出，FH-OAOS 优于文献[18]所提出的算法。对于 $\lambda=2$ 的所有测试用例，平均线长减少可达到 27.83%。另外，对于 $\lambda=4$ 的情况，当问题的规模很小时，文献[18]

具有与该算法相似的表现(rc01～rc05)。然而,当问题的规模很大(rc06～rc12)时,特别是当障碍物的数量大于引脚数时,本书算法具有显著的优势,线长减少−3.37%～55.09%。平均改善率也可达到19.53%。换句话说,文献[18]算法生成的最终布线树相对较差,主要原因是文献[18]的算法可能会引入更多的冗余点,以及文献[18]的布线树不能尽可能地共享相同的布线路径。此外,最后3列显示了两种算法的运行时间。可以看出,FH-OAOS比文献[18]的算法平均快5.79倍和6.34倍(当$\lambda=2$,$\lambda=4$时)。很明显,文献[18]的算法比文献[28]的算法在运行时间方面好。但是,对于最终布线树的质量,文献[28]的算法比文献[18]的算法好。因此,FH-OAOS弥补了文献[18]和文献[28]两个算法的缺点。

表5.14　与文献[18]的线长和运行时间的比较结果

测试用例	引脚数	障碍物数	FH-OAOS	[18]		优化率/%		CPU时间/s		
				$\lambda=2$	$\lambda=4$	$\lambda=2$	$\lambda=4$	FH-OAOS	$\lambda=2$	$\lambda=4$
rc01	10	10	25 084	30 410	27 279	17.51	8.05	0.00	0.00	0.00
rc02	30	10	39 488	45 640	41 222	13.48	4.21	0.00	0.00	0.00
rc03	50	10	54 177	58 570	52 432	7.50	−3.33	0.00	0.00	0.01
rc04	70	10	59 643	63 340	57 699	5.84	−3.37	0.00	0.00	0.01
rc05	100	10	72 738	83 150	73 090	12.52	0.48	0.00	0.00	0.01
rc06	100	500	77 592	149 725	135 454	48.18	42.72	0.02	0.06	0.07
rc07	200	500	105 480	181 470	162 762	41.87	35.19	0.02	0.06	0.08
rc08	200	800	113 110	202 741	182 056	44.21	37.87	0.03	0.10	0.12
rc09	200	1000	110 642	214 850	193 228	48.50	42.74	0.03	0.13	0.15
rc10	500	100	155 579	198 010	176 497	21.43	11.85	0.02	0.03	0.03
rc11	1000	100	216 401	250 570	222 758	13.64	2.85	0.03	0.04	3.03
rc12	1000	10 000	702 544	1 723 990	1 564 170	59.25	55.09	0.41	2.82	3.03
均值						27.83	19.53	0.56	3.24	3.55
比例								1.00	5.79	6.34

4. 与最新的3个OARSMT算法对比

为了进一步验证这一事实,即与直角结构相比,X结构对于线长优化有巨大的优势,在这一部分将本文算法与近年来提出的3种最新的OARSMT算法进行了比较。表5.15显示比较结果。表5.15的第4列显示了本节算法的线长。文献[32]是最新的关于OARSMT文献,它提出了借助GPU的并行方法,并以有效的方式取得了巨大的成果。与此并行算法相比,线长优化提高了−0.51%～8.25%,平均减少了3.96%,只有一个测试用例(ind2)比它差。此外,本节算法比文献[31]的算法优4.23%,比文献[12]的算法优6.77%。虽然45°和135°布线方向的引入增加了问题的复杂度,但本节算法也比OARSMT算法更快。在表5.15中提供不同算法的运行时。可以看出,与OARSMT算法相比,比文献[32]的算法快44.18倍,比文献[12]的算法快12.77倍。文献[31]提出的算法实现了3种算法中最快

的运行时间，因为它是基于预计算的查找表的算法。与之相比，本节算法的平均速度依然提高了 2.52 倍。

此外，为了进行各方面一一对应的比较，尝试通过对 FH-OAOS 进行以下更改来生成 OARSMT。首先，从整个算法中删除选择 0 和选择 1。因此，算法的步骤 3 将生成的 OFEMST 转换为直角 Steiner 树（RST）。然后从布线平面中选取一些 pseudo-Steiner 点，生成 OARST。应该注意，在步骤 3 中，布线平面参照每个点 p 被划分为 4 个直角区域（即，图 5.24 的 d_1、d_3、d_5 和 d_7 是可用方向）。其次，pseudo-Steiner 点连接优化过程基于 X 结构，因此变得不可用，在布线选择优化过程中每个点只应考虑直角组合。此外，由于只有直线路径，冗余 pseudo-Steiner 点消除也不能减少线长，这是因为它被设计成尽可能用合法的 X 结构边来代替冗余直角边。可以看出，在选择 0 和选择 1 被删除之后，精炼过程几乎没有效果。这主要是因为 X 结构和直角结构的优化技术不是通用的。例如，文献[30]算法中的边替换法和“U 形图案细化”法被广泛用于优化直角结构，但不适用于 X 结构。

表 5.15 为与 3 个文献的算法关于线长和运行时间的比较结果，可以看出，FH-OAOS 的性能平均比文献[32]的算法差 2.66%，比文献[31]的算法差 2.38%，比文献[12]的算法优 0.36%。此外，对于 VLSI 布线的另一个重要指标，即运行时间，FH-OAOS 仍然具有巨大的优势。FH-OAOS 比文献[31]、[32]、[12]的算法分别快 2.52 倍、44.18 倍和 12.77 倍。

表 5.15 与 3 个文献的算法关于线长和运行时间的比较结果

测试用例	引脚数	障碍物数	线长		优化率/%	CPU 时间/s	
			FH-OAOS	[32]		FH-OAOS	[32]
ind1	10	32	568(618)	609	6.73(−1.48)	0.00(0.00)	0.05
ind2	10	43	9548(9800)	9500	−0.51(−3.16)	0.00(0.00)	0.06
ind3	10	50	574(613)	600	4.33(−2.17)	0.00(0.00)	0.05
ind4	25	79	1069(1146)	1092	2.11(−4.95)	0.00(0.00)	0.09
ind5	33	71	1334(1412)	1345	0.82(−4.98)	0.00(0.00)	0.08
rc01	10	10	25 084(27 630)	25 980	3.45(−6.35)	0.00(0.00)	0.05
rc02	30	10	39 488(43 290)	41 740	5.40(−3.71)	0.00(0.00)	0.06
rc03	50	10	54 177(56 940)	55 500	2.38(−2.59)	0.00(0.00)	0.07
rc04	70	10	59 643(61 990)	60 120	0.79(−3.11)	0.00(0.00)	0.06
rc05	100	10	72 738(75 685)	75 390	3.52(−0.39)	0.00(0.00)	0.09
rc06	100	500	77 592(84 662)	81 340	4.61(−4.08)	0.02(0.02)	0.38
rc07	200	500	105 480(113 598)	110 952	4.93(−2.38)	0.02(0.02)	0.31
rc08	200	800	113 110(119 177)	115 663	2.21(−3.04)	0.03(0.02)	0.46
rc09	200	1000	110 642(117 074)	114 275	3.18(−2.45)	0.03(0.02)	0.61
rc10	500	100	155 579(167 219)	167 830	7.30(0.37)	0.02(0.01)	0.20
rc11	1000	100	216 401(234 107)	235 866	8.25(0.75)	0.03(0.02)	0.34
rc12	1000	10 000	702 544(775 263)	762 089	7.81(−1.73)	0.41(0.36)	21.78
均值(优化率/%)/合计(时间)					3.96(−2.66)	0.56(0.47)	24.74
比率(时间)					—	1.00(1.00)	44.18(52.64)

续表

测试用例	引脚数	障碍物数	线　长		优化率/%		CPU 时间/s	
			[31]	[12]	[31]	[12]	[31]	[12]
ind1	10	32	604	639	5.96(−2.32)	11.11(3.29)	0.00	0.01
ind2	10	43	9500	10 000	−0.51(−3.16)	4.52(2.00)	0.00	0.01
ind3	10	50	600	623	4.33(−2.17)	7.87(1.61)	0.00	0.01
ind4	25	79	1129	1126	5.31(−1.51)	5.12(−1.78)	0.00	0.02
ind5	33	71	1364	1379	2.20(−3.52)	3.26(−2.39)	0.00	0.02
rc01	10	10	25 980	27 540	3.45(−6.35)	8.92(−0.33)	0.00	0.01
rc02	30	10	42 110	41 930	6.23(−2.80)	5.82(−3.24)	0.00	0.01
rc03	50	10	56 030	54 180	3.31(−1.62)	0.00(−5.09)	0.00	0.01
rc04	70	10	59 720	59 050	0.13(−3.80)	−1.00(−4.98)	0.00	0.02
rc05	100	10	75 000	75 630	3.02(−0.91)	3.82(−0.07)	0.00	0.02
rc06	100	500	81 229	86 381	4.48(−4.23)	10.17(1.99)	0.03	0.13
rc07	200	500	110 764	117 093	4.77(−2.56)	9.92(2.98)	0.03	0.15
rc08	200	800	116 047	122 306	2.53(−2.70)	7.52(2.56)	0.05	0.27
rc09	200	1000	115 593	119 308	4.28(−1.28)	7.26(1.87)	0.06	0.36
rc10	500	100	168 280	167 978	7.55(0.63)	7.38(0.45)	0.02	0.08
rc11	1000	100	234 416	232 381	7.69(0.13)	6.87(−0.74)	0.03	0.14
rc12	1000	10 000	756 998	842 689	7.19(−2.41)	16.63(8.00)	1.19	5.88
均值(优化率(%))/合计(时间)					4.23(−2.38)	6.77(0.36)	1.41	7.15
比率(时间)					—	—	2.52(3.00)	12.77 (15.21)

5. 与最优的 OARSMT 算法对比

文献[33]中提出的算法是一种精确的算法，它可以通过在障碍物中连接完整的 Steiner 树来构建最优的 OARSMT。表 5.16 展示了与文献[33]的算法得到的线长和运行时间的比较结果。表 5.16 第 4 列显示，FH-OAOS 平均优于文献[33]的方法 2.35%。这些结果进一步证明，对于降低 VLSI 设计中的布线成本，X 架构具有很大的优势。此外，第 6 列显示 FH-OAOS 比文献[33]的方法快 267 387.53 倍，这主要是因为文献[33]的方法具有指数最坏情况时间复杂度。

表 5.16　与文献[33]的算法的线长和运行时间的比较结果

测试用例	线　长		优化率/%	CPU 时间/s	
	FH-OAOS	文献[33]		FH-OAOS	文献[33]
ind1	568	604	5.96	0.00	0.11
ind2	9548	9500	−0.52	0.00	0.25
ind3	574	600	4.33	0.00	0.19
ind4	1069	1086	1.57	0.00	0.87
ind5	1334	1341	0.52	0.00	1.09
rc01	25 084	25 980	3.45	0.00	0.16

续表

测试用例	线长		优化率/%	CPU 时间/s	
	FH-OAOS	文献[33]		FH-OAOS	文献[33]
rc02	39 488	41 350	4.50	0.00	0.52
rc03	54 177	54 160	−0.03	0.00	0.68
rc04	59 643	59 070	−0.97	0.00	0.95
rc05	72 738	74 070	1.80	0.00	1.31
rc06	77 592	79 714	2.66	0.02	335
rc07	105 480	108 740	3.00	0.02	541
rc08	113 110	112 564	−0.49	0.03	24 170
rc09	110 642	111 005	0.33	0.03	14 174
rc10	155 579	164 150	5.22	0.02	176
rc11	216 401	230 837	6.25	0.03	706
均值(优化率(%))/合计(时间)			2.35	0.15	40 108.13
比率(时间)			—	1.00	267 387.53

6. OSMT 生成的比较

OSMT 可视为 OAOSMT 的特例，实际上，FH-OAOS 可以生成 OSMT 而无须任何修改。在删除障碍后，本节算法对现有的基准测试进行了实验。如表 5.17 所示，将结果与文献[28]的算法进行比较，其中 FH-OAOS* 表示使用精炼方法之前的线长。表 5.17 是与文献[28]的算法的关于线长和运行时间的比较结果可以看出，精炼方法平均减少了 1.79%的线长。此外，本节的线长结果也优于文献[28]的算法。与文献[28]的算法相比，本节算法可以实现−2.19%～1.87%的线长减小，平均线长减少为 0.27%。在运行时间方面，本节算法比文献[28]的算法快 201.38 倍。

表 5.17 与文献[28]的算法的关于线长和运行时间的比较结果

测试用例	引脚数	障碍物数	FH-OAOS	线长		优化率/%		CPU 时间/s	
				FH-OAOS*	文献[28]	FH-OAOS*	文献[28]	FH-OAOS	文献[28]
ind1	10	32	563	578	559	2.60	−0.72	0.00	0.01
ind2	10	43	8814	8838	8814	0.27	0.00	0.00	0.01
ind3	10	50	547	559	547	2.15	0.00	0.00	0.01
ind4	25	79	955	975	963	2.05	0.83	0.00	0.01
ind5	33	71	1152	1163	1147	0.95	−0.44	0.00	0.02
rc01	10	10	24 098	24 311	24 123	0.88	−0.10	0.00	0.01
rc02	30	10	35 734	36 450	36 203	1.96	1.30	0.00	0.01

续表

测试用例	引脚数	障碍物数	FH-OAOS	线长		优化率/%		CPU 时间/s	
				FH-OAOS*	文献[28]	FH-OAOS*	文献[28]	FH-OAOS	文献[28]
rc03	50	10	48 553	49 816	48 819	2.54	0.54	0.00	0.07
rc04	70	10	51 737	52 992	50 627	2.37	−2.19	0.00	0.13
rc05	100	10	67 814	69 358	69 105	2.23	1.87	0.00	0.20
rc06	100	500	72 247	73 479	72 996	1.68	1.03	0.00	0.22
rc07	200	500	98 341	100 137	97 541	1.79	−0.82	0.00	1.02
rc08	200	800	101 239	103 002	101 479	1.71	0.24	0.00	1.23
rc09	200	1000	98 724	100 193	99 774	1.47	1.05	0.00	1.37
rc10	500	100	150 080	152 713	151 443	1.72	0.90	0.02	2.29
rc11	1000	100	214 841	219 484	214 073	2.12	−0.36	0.03	4.35
rc12	1000	10 000	698 164	712 154	707 993	1.96	1.39	0.03	5.15
均值(优化率(%))/合计(时间)						1.79	0.27	0.08	16.11
比率(时间)						—	—	1.00	201.38

7. 随机生成电路及仿真结果实验

此外，为了研究本节算法的可扩展性，还测试了一些随机生成的具有额外障碍的情况。在这些情况下，障碍物的数量远远超过引脚的数量。这些情况更类似于实际布线应用程序，例如详细布线或工程更改指令布线。如表 5.18 的仿真实验结果所示，本节算法可以为所有测试用例高效地生成 OSMT 和 OAOSMT。

表 5.18　仿真实验结果

测试用例	引脚数	障碍物数	线长		CPU 时间/s	
			OSMT	OAOSMT	OSMT	OAOSMT
random1	10	500	1710	2320	0.00	0.01
random2	50	500	41 948	46 013	0.00	0.01
random3	100	500	7084	8235	0.00	0.02
random4	100	1000	7190	12 233	0.00	0.03
random5	200	2000	40 257	50 037	0.00	0.13

5.4.4　小结

随着 VLSI 制造技术的快速发展，在一条直线布线平面上可以允许 45°和 135°的对角线段。本节基于 X 结构设计了一种高效的绕障算法，即 OAOSMT 和 OSMT 构造的快速四步启发式算法。与几种最先进的算法相比，实验结果表明，本节算法在线长和运行时均取得了很好的效果，在 VLSI 布线过程中非常实用和有效。

5.5 本章总结

本章主要介绍了4种有效的单层绕障X结构Steiner最小树构造算法。首先，由于离散粒子群在全局优化问题上的独特优势能够使布线结果有着较好的优化，本章基于离散粒子群优化算法构造X结构Steiner最小树；其次，介绍了快速绕障来构造X结构Steiner最小树以及提出了一种高效、有效的OAOSMT构建算法；最后，介绍了四步构建算法：OFEMST的构建、查找表的生成、绕障策略、精炼。本章通过实验展示了所提出算法的有效性，对未来的研究提供了新的思路以及方向。

参 考 文 献

[1] Coulston C S. Constructing exact octagonal Steiner minimal trees[C]//Proceedings of the 13th ACM Great Lakes symposium on VLSI. 2003：1-6.

[2] Chiang C，Chiang C S. Octilinear Steiner tree construction[C]//The 2002 45th Midwest Symposium on Circuits and Systems，2002. MWSCAS-2002. IEEE，2002，1：I-603.

[3] Chang Y T，Tsai Y W，Chi J C，et al. Obstacle-Avoiding rectilinear steiner minimal tree construction[C]//2008 IEEE International Symposium on VLSI Design，Automation and Test(VLSI-DAT). IEEE，2008：35-38.

[4] Li L，Young E F Y. Obstacle-avoiding rectilinear Steiner tree construction[C]//2008 IEEE/ACM International Conference on Computer-Aided Design. IEEE，2008：523-528.

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

[6] Chuang J R，Lin J M. Efficient multi-layer obstacle-avoiding preferred direction rectilinear Steiner tree construction[C]//16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011). IEEE，2011：527-532.

[7] Liu C H，Chen I C，Lee D T. An efficient algorithm for multi-layer obstacle-avoiding rectilinear Steiner tree construction [C]//Proceedings of the 49th Annual Design Automation Conference. 2012：613-622.

[8] Eberhart R，Kennedy J. A new optimizer using particle swarm theory[C]//MHS′95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science. Ieee，1995：39-43.

[9] Clerc M. Discrete particle swarm optimization，illustrated by the traveling salesman problem[M]//New optimization techniques in engineering. Springer，Berlin，Heidelberg，2004：219-239.

[10] Liu G，Chen G，Guo W. DPSO based octagonal steiner tree algorithm for VLSI routing [C]//2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI). IEEE，2012：383-387.

[11] Lin C W, Chen S Y, Li C F, et al. Obstacle-avoiding rectilinear Steiner tree construction based on spanning graphs [J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008, 27(4): 643-653.

[12] Long J, Zhou H, Memik S O. EBOARST: An efficient edge-based obstacle-avoiding rectilinear Steiner tree construction algorithm[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008, 27(12): 2169-2182.

[13] Liu C H, Yuan S Y, Kuo S Y, et al. An O(n log n) path-based obstacle-avoiding algorithm for rectilinear Steiner tree construction [C]//Proceedings of the 46th Annual Design Automation Conference. 2009: 314-319.

[14] Liu C H, Yuan S Y, Kuo S Y, et al. Obstacle-avoiding rectilinear Steiner tree construction based on Steiner point selection [C]//2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers. IEEE, 2009: 26-32.

[15] Shi Y, Eberhart R C. Parameter selection in particle swarm optimization[J]. In: Porto V. W., Saravanan N., Waagen D., Eiben A. E. (eds) Evolutionary Programming VII. Lecture Notes in Computer Science 1998, 447: 591-600.

[16] Shi Y, Eberhart R C. Empirical study of particle swarm optimization[C]//Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). IEEE, 1999, 3: 1945-1950.

[17] Ratnaweera A, Halgamuge S K, Watson H C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients [J]. *IEEE Transactions on evolutionary computation*, 2004, 8(3): 240-255.

[18] Jing T T, Feng Z, Hu Y, et al. λ-OAT: λ-Geometry Obstacle-Avoiding Tree Construction With O(nlogn) Complexity[J]. *Computer-Aided Design of Integrated Circuits and Systems*, IEEE Transactions on, 2007, 26(11): 2073-2079.

[19] Liu C H, Yuan S Y, Kuo S Y, et al. High-performance obstacle-avoiding rectilinear steiner tree construction[J]. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2009, 14(3): 45.

[20] Liu C H, Kuo S Y, Lee D T, et al. Obstacle-avoiding rectilinear Steiner tree construction: A Steiner-point-based algorithm[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2012, 31(7): 1050-1060.

[21] Hanan M. On Steiner's problem with rectilinear distance[J]. *SIAM Journal on Applied Mathematics*, 1966, 14(2): 255-265.

[22] Teig S L. The X architecture: Not your father's diagonal wiring[C]//Proceedings of the 2002 international workshop on System-level interconnect prediction. 2002: 33-37.

[23] Huang H H, Chang S P, Lin Y C, et al. Timing-driven non-rectangular obstacles-avoiding routing algorithm for the X-architecture [C]//WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering. World Scientific and Engineering Academy and Society, 2009(8).

[24] Yan J T. Timing-driven octilinear Steiner tree construction based on Steiner-point reassignment and path reconstruction[J]. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2008, 13(2): 1-18.

[25] Fortune S. A sweepline algorithm for Voronoi diagrams[J]. *Algorithmica*, 1987, 2(1):

153-174.

[26] Preparata, F. P. M. I. Shamos. Computational Geometry: an Introduction[M]. Springer Verlag, 1985.

[27] Kruskal J B. On the shortest spanning subtree of a graph and the traveling salesman problem[J]. *Proceedings of the American Mathematical Society*, 1956, 7(1): 48-50.

[28] Huang X, Liu G, Guo W, et al. Obstacle-avoiding algorithm in X-architecture based on discrete particle swarm optimization for VLSI design[J]. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2015, 20(2): 24.

[29] Berg M D. Computational Geometry: Algorithms and Applications [M]. Springer Publishing Company, Incorporated, 2000.

[30] Borah M, Owens R M, Irwin M J. An edge-based heuristic for Steiner routing[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits, and Systems*, 1994, 13(12): 1563-1568.

[31] Ajwani G, Chu C, Mak W K. FOARS: FLUTE based obstacle-avoiding rectilinear Steiner tree construction[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011, 30(2): 194-204.

[32] Chow W K, Li L, Young E F Y, et al. Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach[J]. *Integration the VLSI Journal*, 2014, 47: 105-114.

[33] Huang T, Young E F Y. 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, 32: 882-893.