考虑障碍中布线资源再利用的直角斯坦纳树构造算法

张浩[1,2]，叶东毅[1,2]，郭文忠[2]
1.福州大学离散数学与理论计算机科学研究中心
2.福州大学数学与计算机科学学院
福州，350116

摘要：

直角斯坦纳树问题是 VLSI 物理设计中基本问题之一。为了充分利用可布线资源，本文研究了考虑障碍中布线资源再利用的直角斯坦纳树构造算法。本算法构建扩展直角满斯坦纳树网格作为布线图，并设计了带约束的最短路启发式算法为斯坦纳树构造算法。为了提高求解质量，还引入了两种候选斯坦纳点以及插入关键节点局部搜索策略。实验表明，和相关算法相比，该算法在合理运行时间内，充分利用了障碍内可布线资源，节约了障碍外布线资源，而且有效缩短了总布线长度。

关键词：大规模集成电路，斯坦纳树，NP 完全问题，布线图，Dijkstra's 算法

中图分类号： TP391

Rectilinear Steiner tree construction with reusing routing resource on the top of the obstacles

Zhang Hao,Ye Dong-Yi,Guo Wen-Zhong
Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350116
College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116

Abstract:

Rectilinear Steiner minimal tree is one of the fundamental problems in VLSI physical design. In order to save and make good use of the routing resource, we propose a alogrithm for constructing rectilinear steiner tree with reusing routing resource on the top of the obstacles. We build the routing grap by extending the rectilinear full Steiner trees grid. The shortest path heuristic is employed for constructing the Steiner tree by considering some constraints. Two types of Steiner point candidates and a key-node insertion local search are involved for improving the quality of the solutions. Experimental results show that our algorithm make full use of the routing resource on the obstacles as well as saving the outside-obstacles routing resource compared with the conventional obstacle-avoiding rectilinear Steiner minimal tree problem algorithm. In terms of wirelength, our

algorithm improves the latest length-restricted Steiner minimum tree problem algorithm with about 2% in a reasonable running time.

Key words: Very Large Scale Integration**;** Steiner Tree; NP-complete problem; Routing Graph; Dijkstra's algorithm

1.引言

直角斯坦纳树(rectilinear Steiner minimal tree, RSMT)是大规模集成电路(Very Large Scale Integration,VLSI)物理设计中的一个基本问题[1,2]，且是个 NP 完全问题[3]。RSMT 在布线阶段可用于构造线网的初始拓扑；在布图规划和布局阶段，可以估算线网总线长、拥挤度和时延值。随着预先布线线网、宏单元以及知识产权保护模块(intellectual property block)的出现，使得绕障碍直角斯坦纳树(obstacle-avoiding rectilinear Steiner minimal tree problem ,OARSMT)[4-13]问题成为人们研究的热点。在实践中，通常可布线区域包含多个布线层，障碍往往只占据了设备层(device layer)和某几个较低的金属层(metal layers)，并没有完全阻断绕线，即在障碍内部可以布线。利用障碍内部的可布线资源，可以节约障碍外部的布线资源，有效缩短总线长，节约 buffer 资源、减少时延、降低功耗和减少布线拥挤度。信号在较长线网中传递时会产生信号失真，重现失真信号需要在线网中插入 buffer (也称为中继器)，但是 buffer 不能放置在障碍内部。为了避免信号的失真，斯坦纳树在障碍内部的部分需要满足电压转换速率约束(slew constraints)。电压转换速率约束增加了斯坦纳树构造的复杂程度。文献[14-18]中，分别对此问题进行了不同程度地简化和建模，并提出了相应的求解方法。本文中，将这个问题称为考虑障碍中布线资源再利用的直角斯坦纳树构造问题(rectilinear Steiner tree construction with reusing routing resource on the top of the obstacles, RST-RERR)。显然 RSMT 和 OARSMT 是 RST-RERR 问题在约束值取极值时的两种特例，而且 OARSMT 的解是 RST-RERR 的一个可行解。

2.相关工作

2003 年 Muller-Hannemann 等人在文献[14]中提出了解决 RST-RERR 的简化模型：限制长度的斯坦纳最小树(length-restricted Steiner minimum tree problem, LRSMT)。该模型中，斯坦纳树在障碍内的连通分量长度不超过门限值 L。他们修改 DNH(distance network heuristic) [19]算法，来构造 LRSMT 问题的可行解。该算法是 2 近似算法，时间复杂度为$O(n \log n)$，其中n表示 Hanan 格图[1]的规模。在他们随后的工作中还将这个方法拓展到 X 架构下[15]。

2014 年 Stephan 等人[16]沿用了 LRSMT 模型。他们在可视图(visibility graph) [20]的基础上，

构建范围可视图(reach-aware visibility graph)。引脚和障碍拐点中两点间满足 LRSMT 问题约束的最短路一定包含在范围可视图中。通过修改 DNH 算法来构造斯坦纳树，并证明了所得解是 2 近似解。假设复杂障碍的拐点数不超过一个常数，则范围可视图包含$O(k\log k)$条边和顶点，算法的复杂度是$O(k(\log k)^2)$，k表示引脚和障碍拐点的总个数。在预处理过程，将直径小于 L 的小障碍都忽略，这样可以减小范围可视图的规模和构造时间。在后期处理中，使用 Flute[2]和 Prim's 算法重新连接极大无障碍区域内的引脚以改善求解质量。为了保证范围可视图中包含的解满足约束条件，将 LRSMT 模型进一步简化，使得范围可视图不包含障碍内部的斯坦纳点。因此，该算法所得的斯坦纳树，在障碍内部的部分都是路径，而不会包含任何斯坦纳点。这种简化会增加总线长。

[17,18]中引入了一种更加精确的模型，称为带电压转换速率约束的绕障直角斯坦纳树（OARSMT_SC）。该模型使用 PERI 模型[21]计算具体的电压转换速率值，并保证可行解在障碍内部的部分子树不违反电压转换速率约束。

Zhang 等人在文献[17]中设计了一种启发式算法。他们首先使用 Flute[2]构造出一个初始 RSMT 结构。修复过程中，引入三种降低电压转换速率的基本操作，并建立一个整数线性规划(integer linear programming ,ILP)模型来修正违反约束的电压转换速率。使用增量法逐个的修复违反约束部分，修正过程将初始 RSMT 分割成多个连通分量。最后，使用障碍内限制长度的迷宫算法将这些连通分量互连起来得到最终的可行解。ILP 模型中具体的约束数量取决于障碍的形状、引脚的位置以及求解精度。需要计算 ILP 模型中每种可能候选内部树的电压转换速率值。文献中实验结果表明其迷宫算法消耗时较长。

Huang 等人在文献[18]中提出了一种确定性算法，可以求得嵌入在扩展 Hanan 网格中的最优解。任何最优解的斯坦纳树均可分解成一个外部树集合和一个内部树集合，而且这两种类型树结构遵循简单的形式。该算法由两个阶段构成：生成一个组候选内部树和一个组候选外部树；从这个两组树中选择一些树组合成一个最优解斯坦纳树。根据扩展 Hanan 网格的定义可知，其中包含了原始 Hanan 网格[1]和 Escape 图[6]，也就是说其中包含了 RSMT 和 OARSMT 问题的最优解。即便如此，扩展 Hanan 网格并不一定包含有 LRSMT 问题的最优解[14]，显然也不包含 OARSMT_SC 问题的最优解。此外，由于 ILP 问题的规模较大，耗时较长，该算法达不到物理设计对实时性的要求。

LRSMT 中简化了约束模型，提高求解效率，但对约束的计算不够准确，容易引起绕行或者违反实际约束，增加了后续工作的难度。OARSMT_SC 中可以精确满足约束，但在设计过程中

需要频繁计算较为复杂的电压转换速率值。

直角斯坦纳树相关问题(RSMT, OARSMT 和 LRSMT)经常转换成 GSTP [1,5,14,6-11,16]问题。在 GSTP 中，给定一个带权无向连通图和图中的一组端点，目标是使用图中的边将所有端点都互连起来且权重和最小。这种方案首先构造至少包含一个最优解或者近优解的布线图，即原始问题就转换成为 GSTP 问题；再选用一种确定性或者启发式算法来求解这个 GSTP 问题。实验结果表明，这种方案具有较好的效果和效率。布线图的准确性(即包含最佳近优解的质量)和规模,影响斯坦纳树构造算法的效率和效果。用于解决 RST-RERR 问题的布线图中必须包含有 RSMT 问题的近优解和 OARSMT 问题的近优解。如[14,18]中所提到的扩展 Hanan 网格是将 RSMT 问题的布线图和 OARSMT 问题的布线图组合起来，但是构造扩展 Hanan 网格需要花费很长的运行时间和很大的内存空间。扩展 Hanan 网格平方级数的规模也导致斯坦纳树构造算法消耗较长的运行时间。

通过上述分析可知，构造较小规模、具有较高准确性的布线图，尽可能减少电压转换速率值的计算次数，以及提高 GSTP 问题求解质量，是解决 RST-RERR 问题的关键所在。

本文提出了一种求解 RST-RERR 问题的启发式算法,用 RST-RERR-H 表示。本文的贡献如下：

1. 使用 GeoSteiner 软件[22]构造扩展直角满斯坦纳树网格作为布线图，确保了布线图中至少包含一个 RSMT 问题的最优解和一个 OARSMT 问题的近优解，并将 RST-RERR 问题转换成带约束的图中斯坦纳树问题(Steiner tree problem in graphs,GSTP)。
2. 分别使用最短路集合和泰森图(Voronoi Diagram)在扩展直角满斯坦纳树网格中标记出两种候选斯坦纳点，均用于改善可行解质量。
3. 通过修改最短路算法(Shortest path heuristic, SPH)算法，设计了一种逐步生长的启发式算法解决带约束的 GSTP 问题,在扩展直角满斯坦纳树网格中连通所有的引脚和指定的候选斯坦纳点；并运用预计算策略，避免频繁计算电压转换速率值。
4. 引入插入关键节点局部搜索，将第二种候选斯坦纳点作为可插入关键节点，进一步提高求解质量。

3. 问题的表示和基础知识

3.1 RST-RERR 问题定义

由于 buffer 不能在两障碍共用边界线上插入，为了更精确的计算电压转换速率，本文使用严格复杂直角障碍定义。

定义 1 严格复杂直角障碍：障碍是直角多边形，不同障碍不能相互叠加，也不能共用边界线，

可以共用拐点。

在测试例子中，将存在共享边界线的障碍合并成一个更大的障碍。本文的严格复杂直角障碍定义和[17,16]相同，与[5,18]不同。

本文中，使用两种不同约束条件的模型来构造直角斯坦纳树：OARSMT_SC[17,18]和LRSMT[14,16]。这两种约束统称为 RST-RERR 约束。

定义 2 RST-RERR 问题：在矩形布线区域内，有一组引脚$P = \{p_0, p_1, \dots, p_n\}$，一组严格复杂直角障碍$O = \{O_0, O_1, \dots, O_k\}$，其中引脚$p_0$是信号源，其他引脚为宿点；引脚不允许位于障碍的内部，但是可以位于障碍的边界上；在满足 RST-RERR 约束条件下，构造直角斯坦纳树连接所有引脚，使得总线长度最短。

定义 3 内部树和外部树： T表示 RST-RERR 问题的一个解，障碍边界将T切割分割成两组子树：内部树和外部树，内部树是某障碍内部的连通分量，外部树是所有障碍外部的连通分量。内部树的每个叶子节点都在障碍的边界上。其中一个叶子节点沿着T距离p_0最近，称为驱动节点，其他叶子称为接收节点。

本文中斯坦纳点是指：在斯坦纳树上度数大于 2 且不是引脚的顶点。斯坦纳树上引脚和斯坦纳点都称为关键节点(key-node)。在 GSTP 等通用问题中，引脚也称为端点。

3.2 约束相关知识

为了便于描述，约束相关符号表示见表 1。信号源p_0沿着斯坦纳树T将信号驱动到所有宿点上。内部树中信号从驱动节点传递到所有的接收节点。为了判断T是否违反约束，假定 buffer 插入在T上的最佳可能的位置上，但不能插入在障碍内部也就是内部树上。为了驱动信号穿过内部树，假定在靠近驱动节点之前的位置插入一个 buffer (b_{in})；为了屏蔽下游电容，在靠近每个接收节点之后的位置都插入一个 buffer(b_{out_i})。如图 3.1 表示一个包含 4 个叶节点的内部树。电压转换速率约束指的是：由$\mathrm{b_{in}}$驱动的信号在接收节点上的电压转换速率不超过特定的门限值 MAX_SC。本文中，将一个内部树中最大的接收节点电压转换速率称为内部树电压转换速率；将一个解中最大的内部树电压转换速率称为该解的电压转换速率。

表 1 约束相关的符号表示

Table I Constraints related notations in this paper

符号	符号的含义	符号	符号的含义
R_b	b_{in}的电压转换速率阻抗	S_v	v点处的电压转换速率
K_b	b_{in}的固有电压转换速率	$S_{step}(v,u)$	v点到u点的步进电压转换速率
c_b	Buffer 的输入电容	$Elmore(v,u)$	v点到u点之间的 Elmore 时延[23,24]

r_b	Buffer 的输出电阻	$S_{v_{in}}$	b_{in}上的输出电压转换速率
r_0	单位长度线上电阻	$C_t(v)$	v处的下游负载电容
c_0	单位长度线上电容	c_v	v点处的电容
v_{in}	内部树的驱动节点	$Succ(v)$	v点的直接后继节点集合
v_{out_i}	内部树的第 i 个接收节点	$Path(v_i, v_j)$	v_i 和 v_j之间的路径,,
b_{in}	驱动节点前的 buffer	MAX_SC	内部树电压转换速率门限值
b_{out_i}	第 i 个接收节点后的 buffer	MAX_LR	内部树总线长门限值
len_i	相关线段的长度		

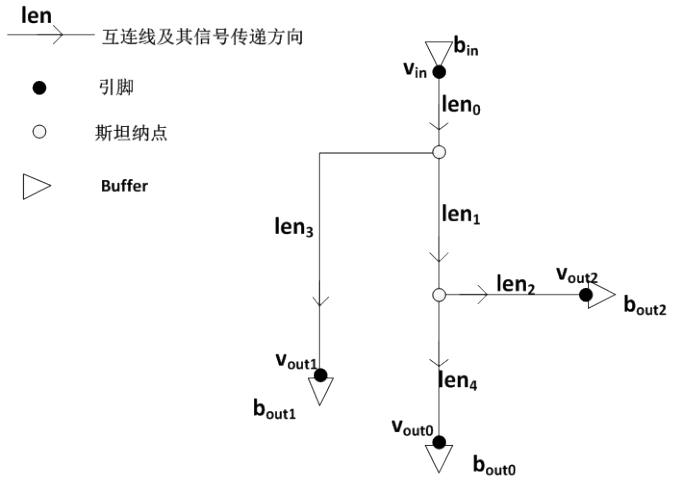


图 3.1 带有 4 个叶节点的内部树

Fig 3.1 a internal tree with four leaves

本文中，使用 PERI 模型[25]计算电压转换速率。如图 3.1 所示，在该内部树中，上游点(信号源)v_{in}之前插入了 buffer b_{in}，下游点(宿点)v_{out_0}之后插入了 buffer b_{out_0}，它们之间的路径上不能插入任何 buffer。v_{out_0}处的电压转换速率由公式(1~5) 计算。其中$S_{step}(v_{in}, v_{out_0})$与$v_{in}$和$v_{out_0}$之间的 Elmore 时延[23,24]相关；$S_{v_{in}}$取决于$b_{in}$的输入电压转换速率和负载电容，[26]中给出一个简化计算方法如公式(3)。

$$S_{v_{out_0}} = \sqrt{{S_{v_{in}}}^2 + S_{step}(v_{in}, v_{out_0})^2} \quad (1)$$

$$S_{step}(v_{in}, v_{out_0}) = ln\,9 * Elmore(v_{in}, v_{out_0}) \quad (2)$$

$$S_{v_{in}} = R_b * C_t(v_{in}) + K_b \quad (3)$$

$$C_t(v) = c_v + \sum_{u \in Succ(v)} C_t(u) \quad (4)$$

$$Elmore(v_i, v_j) = \sum_{v_r \in Path(v_i, v_j)} r * C_t(v_r) \quad (5)$$

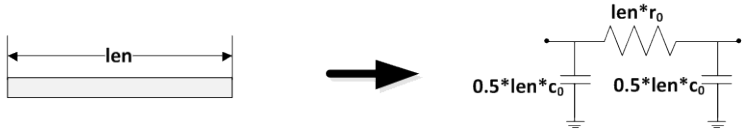
$$Elmore(v_{in}, v_{out0}) = r_{b.in} * \left(\sum_{i=0}^{4} len_i * c_0 + \sum_{j=0}^{2} * c_{b.outj} \right) +$$

$$len_0 * r_0 * \left(0.5 * len_0 * c_0 + \sum_{i=1}^{4} len_i * c_0 + \sum_{j=0}^{2} * c_{b.outj} \right) +$$

$$len_1 * r_0 * (0.5 * len_1 * c_0 + (len_2 + len_4) * c_0 + c_{b.out0} + c_{b.out2}) +$$

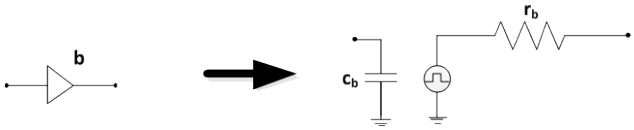
$$len_2 * r_0 * (0.5 * len_2 * c_0 + c_{b.out2}) \quad (6)$$

在 Elmore 时延[23,24]模型中，互连线和 buffer 都建模成电容电阻电路。如图 3.2(a)所示，将互连线看作均匀分布的电阻电容线(uniformly distributed RC line)[27]，并建模成一个 π 型集中电阻电容电路，一半的线上电容(wire capacitance)位于上游节点，剩下的一半线上电容位于下游节点。如图 3.2(b)所示，buffer 可以建模成：一个与上游相连的输入电容(input capacitance) c_b，一个与下游相连的输出电阻(output resistance) r_b。图 3.3 给出了图 3.1 中内部树对应的电阻电容电路树。公式(5)表示节点v_i 和 v_j之间的 Elmore 时延，其中r表示路径$Path(v_i, v_j)$上的一个线上电阻，v_r则表示该线上电阻的位置。公式(6)表示的是v_{in} 到 v_{out_0}之间的 Elmore 时延。



(a) 均匀分布电阻电容线的 π 模型集中电阻电容电路

(a) A π model of a uniformly distributed RC line for a wire segment



(b)Buffer 的电阻电容电路

(b)The model for a buffer b

图 3.2 电阻电容电路模型

Fig 3.2 RC model

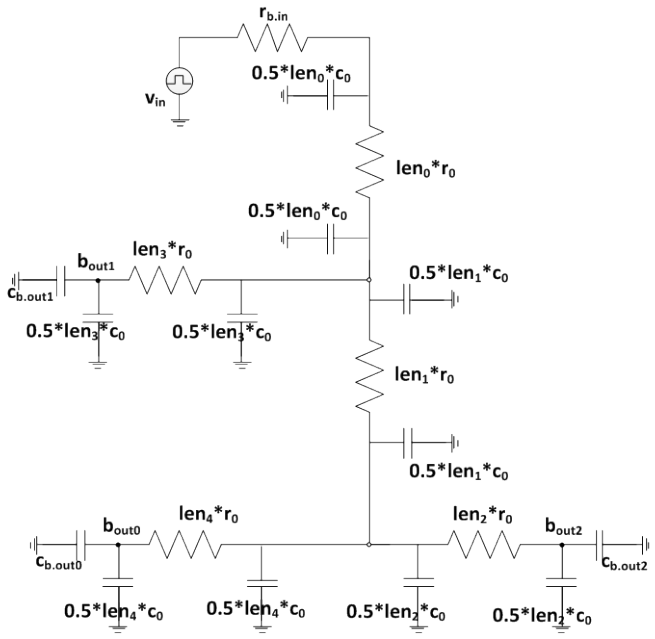


图 3.3 图 3.1 中内部树对应的电阻电容电路树

Fig 3.3 the corresponding RC tree of the internal tree shown in Fig 3.1

内部树的电压转换速率与该树的总线长关系密切，LRSMT 问题是将电压转换速率约束简化为限制内部树的总线长不超过门限值 MAX_LR。在 LRSMT 中，只需要计算内部树的总线长，而不需要计算复杂的内部树电压转换速率。

4. 算法设计

RST-RERR-H 算法将 LRSMT 和 OARSMT_SC 两种模型下的 RST-RERR 问题，均转换成带约束的 GSTP 问题。如图 4.1 所示，本文算法共由四个步骤组成：

1) 生成布线图：本文算法中布线图是通过 GeoSteiner 软件[22]构造出扩展直角满斯坦纳树网格，布线图中包含至少一个 RSMT 问题的最优解和一个 OARSMT 问题的近优解，并在布线图中标记出位于障碍内部的顶点和边。
2) 标记候选斯坦纳点：为了提高所构造斯坦纳树的质量，本文算法中标记了两种候选斯坦纳点，第一种用于 3)步骤中的斯坦纳树构造，第二种用于 4)步骤中的局部搜索。
3) 斯坦纳树构造：在满足 RST-RERR 约束下，使用布线图中的边，将所有引脚和第一种候选斯坦纳点都互连起来，得到一个斯坦纳树初始解。
4) 改善过程：将第二种候选斯坦纳点作为可插入的关键节点，执行插入关键节点局部搜索策略，改善初始解的质量。

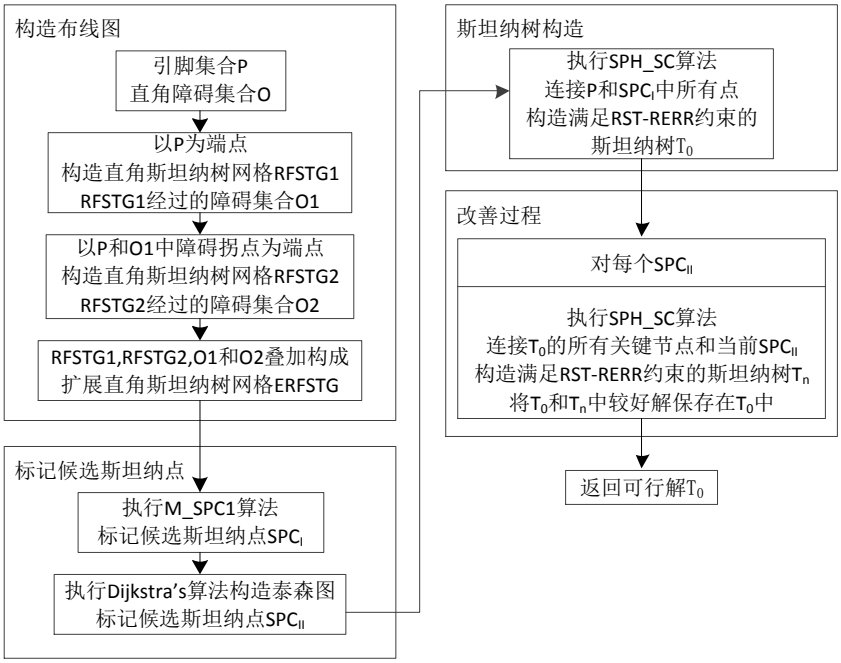


图 4.1 算法 RST-RERR-H 的框图

Fig 4.1 Flow of RST-RERR-H

4.1 布线图的生成

RSMT 问题的最优解可以由一组直角满斯坦纳树(rectilinear full Steiner tree, RFST)的并集构成[28]。直角满斯坦纳树是指：每个叶节点都是端点，且其他顶点都不是端点的斯坦纳树。直角满斯坦纳树网格(rectilinear full Steiner trees grid, RFSTG) 是由足够的 RFST 叠加组成，并保证至少包含一个 RSMT 最优解。实践中，相对于端点个数而言，RFSTG 中度数大于 2 的斯坦纳点的平均个数总是线性的，随机生成测试例子的 RFSTG 中包含的 RFST 个数期望值也近似线性，且生成 RFSTG 的观察时间是平方级的[28]。从实践的角度来看，RFSTG 是解决 RSMT 问题的一个简单有效的布线图。

为了解决 RST-RERR 问题，还需要进一步扩展 RFSTG。为了便于描述，本文中提到的障碍内部不包括障碍边界。首先，将引脚集合P视为端点集合，执行 GeoSteiner 软件[22]，得到一个原始的 RFSTG，用 RFSTG1 表示。将所有被 RFSTG1 经过内部的障碍标记为$O1$。为了满足 RST-RERR 约束，需要将$O1$中障碍的拐点与周围引脚连接起来。将$O1$中障碍的拐点和所有引脚视为端点，再构造一个 RFSTG，用 RFSTG2 表示。将所有被 RFSTG2 穿过内部的障碍标记为$O2$。最后，将 RFSTG1、RFSTG2 和$O1 \cup O2$中所有障碍边界都叠加起来，构成扩展直角满斯坦纳树网格(extended rectilinear full Steiner trees grid, ERFSTG)。显然，ERFSTG 在规模和构造的运行时间上，与 RFSTG 具有相同的属性。

ERFSTG 中，位于障碍内部的顶点称为内部点，位于障碍外部的顶点称为外部点，剩下的顶点都在障碍的边界上，称为边界点。对于 ERFSTG 中的边也有类似的划分，所不同的是，外部边和边界边没有区分都称为外部边。距离某内部点最近的边界点称为该内部点的逃逸点，它们间在 ERFSTG 上的最短距离称为逃逸距离。利用 Dijkstra's 算法，将所有的边界点视作源点，只沿着内部边进行扩展，即可计算出所有内部点的逃逸点和逃逸距离。

$ERFSTG = (V, E, \omega)$是一个带权无向连通图。$V = \{v_1, v_2, \dots, v_l\}$表示顶点集合，$V_{in}$表示内部点集合，$V_{ex}$表示外部点集合，$V_{boun}$表示边界点集合，则$P \subseteq V_{ex} \cup V_{boun}$。$E = \{e_1, e_2, \dots, e_m\}$表示边集合，$E_{in}$表示内部边集合，$E_{ex}$表示外部边集合。集合中成员个数表示为：$l = |V|$，$m = |E|$。$\omega$表示边的权重函数，本文中为边的长度。

4.2 标记候选斯坦纳点

构建关于 GSTP 问题最优解斯坦纳点和端点的 Distance Network[19]，最优解对应 Distance Network 上的一棵生成树，则 Distance Network 最小生成树权重不大于最优解的权重。

性质 1：如果已知 GSTP 问题最优解的所有斯坦纳点，则可以通过最小生成树算法构造出一个最优解。

由性质 1 可知选择合适的候选斯坦纳点，有益于得到质量较高的解。本文标记了两种类型的候选斯坦纳点。在构造斯坦纳树时，尽可能经过这些候选斯坦纳点。

第一种类型的候选斯坦纳点用SPC_I表示，通过基于最短路集合算法(表示为M_SPC_I)进行标记，具体算法如下：初始时，顶点集合VS中包含一个引脚；每次迭代过程中，找到距离VS最近的一个引脚nt，将nt和VS之间所有最短路上的顶点都加入到VS中，将这些最短路不是nt的终端顶点标记为候选斯坦纳点；直到，VS中包含了所有的引脚。每次迭代通过执行一次 Dijkstra's 算法，找到最近的引脚和相应的最短路集合。Dijkstra's 算法耗时$O(m * \lg(l))$，总共需要执行n次 Dijkstra's 算法

引理 1：在ERFSTG上用算法M_SPC_I标记SPC_I，算法运行时间是$O(n * m * \lg(l))$。

在M_SPC_I算法中，除了第一次迭代以外，Dijkstra's 算法不需要从头开始，只需要将新加入的最短路上的顶点标成源点。M_SPC_I算法伪代码见图 4.2。

	Algorithm: M_SPC_I $(ERFSTG, P)$
	Input: $ERFSTG = (V, E, \omega)$ //扩展直角满斯坦纳树网格
	P //引脚集合
	Result: SPC_I //候选斯坦纳点集合
1	**Begin**

```
2       SPC_I = ∅;
3       SPB = ∅;        //最近引脚和VS之间所有最短路上的顶点
4       VHeap = ∅;      //存放顶点的二进制堆
5       for each vertex u ∈ V
6           u.dist = ∞;      // u回溯到源点路径的长度
7       任意选择一个引脚 s;
8       VS = {s};       //初始化顶点集合VS
9       s.dist = 0;
10      VHeap.push_back(s);
11      Repeat
12          u = VHeap.pop_heap();      //弹出具有最小 dist 值的顶点
13          if (u ∈ P)
14              TraceBack(u);// u是最近引脚
15              VS = SPB ∪ VS;
16              SPB = ∅;
17              continue;
18          for 每条和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 P ⊆ VS;
23  return SPC_I;
24  Function TraceBack(u)      //递归过程
25      if u ∈ VS
26          if(u ∉ P)
27              SPC_I = SPC_I ∪ {u};
28          return;
29      for 每条和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};      //更新SPB
34      VHeap.push_back(u);
35  return;
```

图 4.2 算法M_SPC$_I$的伪代码

Fig. 4.2 M_SPC$_I$ Pseudocode

虽然最优解大多数斯坦纳点都位于某些最短路上，但并不是所有的斯坦纳点都是如此，如图 4.3 给出了一个例子。为了标记此类斯坦纳点，本文中用泰森图来标记第二种候选斯坦纳点。将所有引脚都视为种子，用 Dijkstra's 算法构建泰森图。如果某个顶点和其相邻顶点属于三个以上不同泰森单元，则称为交界点。用 DNH 算法将所有交界点和引脚都互连起来，将得到的

斯坦纳树的斯坦纳点标记为第二种候选斯坦纳点SPC_{II}，则SPC_{II}的个数不超过$n-2$[29]。DNH算法耗时$O(m*\lg(l))$[19]。

引理 2：在 ERFSTG上标记SPC_{II}，所需的时间是$O(m*\lg(l))$。

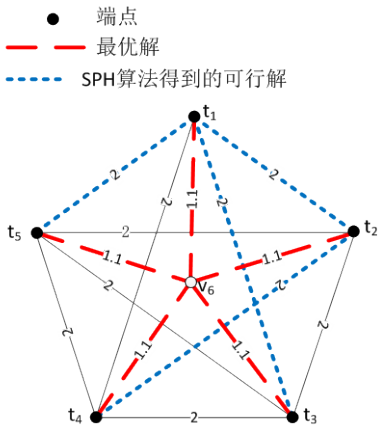


图 4.3 第二类候选斯坦纳点的例子

Fig 4.3 A simple example.

最优解的总线长是 5.5。

用 SPH 算法得到的可行解的总线长是 8。

4.3 斯坦纳树构造

SPH 算法是解决 GSTP 问题的 2 近似算法[30]。在实际的求解质量和运行时间方面，SPH 算法比 DNH 算法具有更佳性能表现[31]。本文将 RST-RERR 约束引入到 SPH 算法中，在 ERFSTG 中，互连一些指定的顶点(记为S，且$P \subset S$)。将这个算法称为带 RST-RERR 约束的 SPH，标记为 SPH_SC。

SPH_SC 是个迭代的过程：初始化时，部分解中只包含有p_0；每次迭代过程中，连接S中部分解以外的一个新顶点到部分解上，，并要求保证不违反 RST-RERR 约束。通过修改 Dijkstra's 算法找到这个新顶点和用于连接的路径。在修改版 Dijkstra's 算法每次扩展中，不仅要考虑当前扩展点回溯到部分解的路径长度，还要检查将扩展关联的路径加入到部分解后的 RST-RERR 约束。

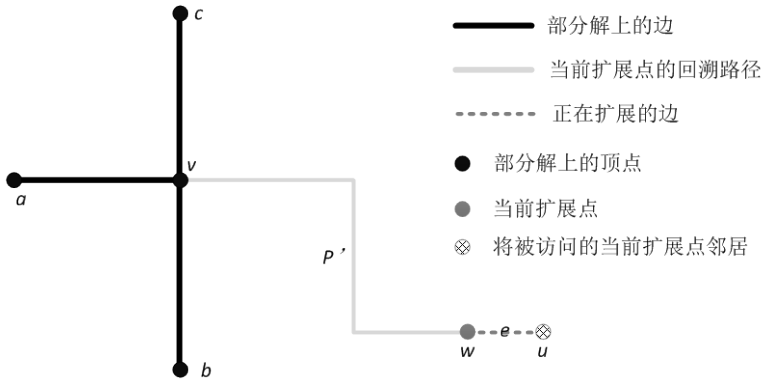


图 4.4 修改版 Dijkstra's 算法在障碍内部扩展的示意图

Fig. 4.4 A propagation of our Dijkstra's algorithm inside a obstacle

在修改版 Dijkstra's 算法中，主要修改了算法的扩展。如图 4.4，是算法扩展的示意图。黑色线表示一个部分解。w表示当前扩展点，u是将被访问的w邻居，它们之间的边是e。灰色线表示的路径P'，是使用扩展信息从w回溯到部分解的路径。P'的另一个终端顶点v是部分解上的一个点。$w.dist$表示w回溯到部分解路径的长度，即P'的长度，初始化时设为无穷大。

首先考虑解决 OARSMT_SC 问题。如果e是外部边，扩展与原算法相同。如果e是内部边，根据P'的组成成分不同，分成两种情形考虑。第一种情形中，v是部分解中某个内部树$v.IT$上的内部点， P'上所有的边都是内部边，而且w也是内部点。因此，e、P'和$v.IT$都位于同一个障碍内部，它们组成新的内部树用IT'表示。将v称为w在$v.IT$上的前驱，表示为$w.prec$。$w.dist2$表示w与$w.prec$之间路径的长度，即图 4.4 中P'的长度。$w.slew$表示w所在内部树(由P'和$v.IT$组成)的电压转换速率。P表示P'和e构成的路径，不妨用变量x表示P的长度。有公式(1~5)可知，可以用一个关于x的一元多项式$f_i^{IT'}(x)$来表示IT'的第i个接收点的电压转换速率平方值。假设有q个接收点，即有$i \in \{1 \dots q\}$。$f_i^{IT'}(x)$的多项式系数都大于 0，则：当$x \geq 0$时，$f_i^{IT'}(x)$是单调递增函数。内部树IT'的电压转换速率是$max_{i\in\{1\dots q\}}(\sqrt{f_i^{IT'}(x)})$。因此，多项式组$f_i^{IT'}(x), (i \in \{1 \dots q\})$中只需要保留可能具有最大值的多项式即可，根据多项式的单调递增特性，可以根据帕累托效率，忽略某些多项式，以减少运算量。保留下来的多项式组$f_i^{IT'}(x), (i \in \{1 \dots q'\})$，称为$v$的关联多项式组。为了确保$IT'$没有违反约束，则要满足$f_i^{IT'}(x) \leq Max_SC^2, \forall i \in \{1 \dots q'\}$。使用牛顿迭代法(Newton–Raphson method)可以求解一元多次方程：$f_i^{IT'}(x) = Max\_SC^2$，即可得到相应的x最大取值Max_x_i。则内部树$v.IT$上的内部点v的障碍内最大可扩展距离$v.Max_ED = min_{i\in\{1\dots q\}}(Max_x_i)$。顶点$u$是否被扩展，即更新$u$关联的信息并压入到堆栈中，

需要根据下列逻辑进行判断：首先，x加上u点的逃逸距离(表示为$u.esc$)不大于$v.Max_ED$，这表明IT'没有违反最大电压转换速率约束；如果，x小于$u.dist$，则u点新的扩展方案比旧的方案离源点更近，顶点u被扩展；如果x等于$u.dist$，并且u点在旧的扩展方案中的$u.slew$大于IT'电压转换速率，则顶点u被扩展。其他情况属于第二种情形，路径P中一定包含一个完全由内部边组成的且以u为终端顶点的极大内部路径P'' (可能只包含e)，P''就是扩展后u所在的内部树。对于仅含有一条路径的内部树，以它的长度为变量，同样得出一个计算电压转换速率的多项式$f(x)$，它的最大长度Max_PL相当于第一种情形中的障碍内最大可扩展距离。顶点u是否被扩展的判断逻辑与第一种情形类似：首先，P''路径长度加上$u.esc$不大于Max_PL，这表明没有违反最大电压转换速率约束；如果，x小于$u.dist$，则u点新的扩展方案比旧的方案离源点更近，顶点u被扩展；如果x等于$u.dist$，并且u点在旧的扩展方案中的$u.slew$大于P''电压转换速率，则顶点u被扩展。$f(x)$和Max_PL在算法中会被频繁使用，在 SPH_SC 算法初始化时预先计算。

每次调用 Dijkstra's 算法后，更新每个新增或有修改内部树上每个内部顶点的下列信息：该内部顶点的关联多项式组和在障碍内最大可扩展距离。这些信息将在下一次调用 Dijkstra's 算法时使用，而不需要在扩展时计算内部树的电压转换速率。

性质 2：修改版 Dijkstra's 算法在障碍内部扩展时，不需要在每步扩展后都计算内部树的电压转换速率，只有在待扩展点信息需要更新时，才使用预先得到的关联多项式来计算待扩展点的电压转换速率。

在解决 LRSMT 时，修改版 Dijkstra's 算法的扩展，用计算内部树的总线长替代计算内部树的电压转换速率即可。

与M_SPC_I算法类似，除了首次迭代，SPH_SC 每次迭代过程中的 Dijkstra's 算法也不需要从头开始。只需要将新加入到部分解中的顶点视为源点即可。此外，SPH_SC 生成的斯坦纳树中，存在一些不必要的路径。这些路径的一个终端顶点是 1 度的非引脚顶点，其他顶点均为度数为 2 的非引脚顶点。使用通过遍历集合$S-P$中的顶点，并执行修剪操作，直接将这些路径从斯坦纳树上删除。SPH_SC 算法伪代码见图 4.5。

SPH_SC 算法中共执行$|S|$次 Dijkstra's 算法，修剪操作在线性时间内完成。

引理 3：在ERFSTG上，SPH_SC 算法运行时间是$O(|S|*m*\lg(l))$，使用 SPH_SC 构造斯坦纳树互连P和SPC_I中顶点，所需的运行时间为$O((n+|SPC_I|)*m*\lg(l))$。

Algorithm: SPH_SC($ERFSTG, P, SPC_I$)

Input: $ERFSTG=(V,E,\omega)$ // 扩展直角满斯坦纳树网格

P //引脚集合

SPC_l // 第一类候选斯坦纳点

Result: T //可行解

Begin

$VHeap = \emptyset$; //存放顶点的二进制堆

for each vertex $u \in V$

$u.dist = \infty$; // u回溯到部分解路径的长度

$u.slew = 0$; // u 的电压转换速率

$T = \{p_0\}$; // 初始化部分可行解为源点

$p_0.dist = 0$;

$p_0.slew = 0$;

$VHeap.push\_back(p_0)$;

Repeat

$w = VHeap.pop\_heap()$; // w是$VHeap$中具有最小 dist 值的所有顶点中 slew 值最小的顶点

if $(w \in (P \cup SPC_l))$

$path = \emptyset$;

TraceBack(w);

$T = T \cup path$

更新被$path$经过的内部树上内部点的关联多项式和障碍内最大可扩展距离信息

continue;

for 每条和w关联的边$e(w,u)$

if $e \in E_{ex}$

if $u.dist > w.dist + \omega(e)$

$u.dist = w.dist + \omega(e)$;

$u.slew = 0$;

$u.parent = w$;

$w.IT = \emptyset$;

$VHeap.push\_back(v)$;

else

if $w \in V_{boun}$

$w.IT = \{w\}$;

if $(w.dist2 + \omega(e) + u.esc \le w.prec.Max\_ED)$

if $(u.dist > w.dist + \omega(e))$

Update(u,w);

if $\left((u.dist == w.dist + \omega(e))\&\&\left(u.slew > \max_{i\in\{1\ldots q\}}(\sqrt{f_i^{w.IT}(w.dist2 + \omega(e))})\right)\right)$

Update(u,w);

Until T 包含了所有引脚;

使用剪枝操作删除多余的路径

return T;

Function **TraceBack**(u) //是一个递归过程

if T中包含u

return;

if $e(u,u.parent) \in E_{in}$

$u.IT = u.IT \cup \{e\}$

```
42          path = path ∪ {e(u, u.parent)};
43          u.dist = 0;
44          VHeap.push_back(u);
45          TraceBack(u.parent);
46      Function Update(u, w)
47          u.dist = w.dist + ω(e);
48          u.dist2 = w.dist2 + ω(e);
49          u.IT = w.IT
50          u.slew = max_{i∈{1...q}}(√(f_i^{u.IT}(u.dist2)));
51          u.parent = w;
52          u.prec = w.prec
53          VHeap.push_back(u);
54      return;
```

图 4.5 解决 OARSMT_SC 问题的 SPH_SC 算法伪代码

Fig. 4.5 SPH_SC Pseudocode for OARSMT_SC

4.4 改善过程

不合适的$\mathrm{SPC_I}$可能会导致 SPH_SC 算法构造的可行解中存在绕行路径。插入关键节点局部搜索可以消除部分绕行路径。此外，以SPC_{II}作为插入的关键点，可以进一步提高可行解质量。插入关键节点局部搜索中采用了贪心策略：对于一个需要插入的SPC_{II}点，利用 SPH_SC 算法将该SPC_{II}点和原可行解的关键节点互连起来，如果所得的可行解质量优于原可行解，则用新的可行解替换原可行解。改善过程消耗了算法主要的运行时间，限制改善过程中 SPH_SC 算法的执行次数为较小的常数值。

可行解斯坦纳树的斯坦纳点个数不超过$n-2$[29]，SPH_SC 算法连接的顶点数是$\mathrm{O}(n)$，则执行一次关键节点局部搜索耗时$O(n*m*\lg(l))$。

引理 4：改善过程所耗时间是$O(n*m*\lg(l))$。

5. 实验结果

本文算法采用 C++编码 gcc 4.4.7 编译，在 CentOS 6.4 系统中完成测试，硬件环境配置为 3.20-GHz Intel Core I5 处理器、16G 内存的计算机，采用单进程、单线程、单处理器运算。测试了传统 OARSMT[5,7-11]文献中常用的 22 个标准测试电路。本文使用相对改进百分比(percentage relative improvement, PRI)来对比本文算法和其他算法求解质量，见公式(7)。

$$\mathrm{PRI}=\frac{\text{其他算法所得解线长}-\text{本文算法所得解线长}}{\text{本文算法所得解线长}}\times 100\% \tag{7}$$

在严格复杂直角障碍定义下，测试电路 IND5 在较小约束值时是无解的[17,16]，详见测试结果。

解决 LRSMT 问题时，约束值 MAX_LR 设置为测试电路的矩形布线区域边界较长边长(表示为 LBB)的不同比例长度。表 2 列出了本文算法解决 LRSMT 问题得到的线长。第 2 列是传统 OARSMT 最新算法[5]所得解的线长，文献[5]中使用的是矩形障碍，即障碍的共享边界上是允许走线的。第 3 列到第 7 列是所得斯坦纳树在障碍外的线长。第 8 列到第 12 列是所得斯坦纳树的总线长。最后一行是本文算法和 OARSMT[5]之间的相对改进百分比的均值。除了 MAX_LR 为 0 时，本文算法所用障碍外线长要比 OARSMT[5]少 3.15 %到 17.44 %，所用总线长少 1.59%到 5.51%。当 MAX_LR 为 0 时，本文算法所用的线长要比 OARSMT[5]多 2.39%。在表 3 中，将本文算法和 LRSMT 最新算法[16]进行了对比。MAX_LR 较大时，[16]求解质量主要依赖于 Flute 算法[2]。在 MAX_LR 为无穷大时，本文算法结果和[16]结果相仿。其他约束值时，本文算法求解质量比[16]提高了 2%左右。表 3 倒数第二行是本文算法测试 22 个例子的总时间，最后一行是引用文献[16]中数据计算得到的测试 22 个例子的总时间。本文算法的运行时间受约束值 MAX_LR 变化影响较小，平均消耗总时间 0.77 秒。 根据文献[16]中报道的数据，平均消耗总时间为 0.6 秒。[16]算法中，约束值 MAX_LR 越大，考虑的障碍就越少，所需的时间也越少。由于文献[16]中使用的测试硬件明显优于本文测试硬件，因此运行时间对比仅作参考。

解决 OARSMT_SC 问题时，测试电路上并没有提供相关的技术参数。本文中，假定测试电路上的单位长度为微米，其他技术参数的设定参考[26,32]文献中所用值，具体参数见表 4。为了设置具体的电压转换速率约束值，首先使用原 SPH 算法为每个测试电路构造一棵斯坦纳树，将该斯坦纳树的电压转换速率值用$slew_{max}$表示。$slew_{min}$表示一棵只包含有 0 长度路径的内部树的电压转换速率值。测试中使用下列 5 个不同的约束值来测试每个电路：

1) 0 slew
2) 20% slew: $slew_{min} + 20\%(slew_{max} - slew_{min})$
3) 50% slew: $slew_{min} + 50\%(slew_{max} - slew_{min})$
4) 80% slew: $slew_{min} + 80\%(slew_{max} - slew_{min})$
5) 无穷大 slew.

表 5 内容与表 2 中内容相似，列出了本文算法解决 OARSMT_SC 问题得到的线长。除了 0 slew 约束，本文算法所用障碍外线长要比 OARSMT[5]少 6.12 %到 17.44 %，所用总线长少 2.81%到 5.51%。当使用 0 slew 约束时，本文算法所用的线长要比 OARSMT[5]长 2.42%。表 6 是本文算法测试 22 个例子的总运行时间。本文算法的运行时间受约束值 MAX_SC 变化影响较小，平均需要总时间 0.88 秒。由于文献[17]中，没有明确所使用的具体技术参数，因此未能与其对求解

质量进行对比。从[17]中所列的运行时间来看，本文算法具有一定的效率优势。

在使用零约束(MAX_SC=0 或者 MAX_LR=0)时，本文算法所得解的线长要比 OARSMT[5]长，这和不同的障碍定义有较大关系。由表 3 和表 6 可知，使用本文算法解决 OARSMT_SC 问题比解决 LRSMT 问题，多消耗的运行时间微不足道。这表明构造斯坦纳树过程中，计算电压转换速率值所消耗时间很少。

实验结果表明，本文算法在合理时间内，有效解决了 OARSMT_SC 问题和 LRSMT 问题。与传统 OARSMT 算法的对比，表明本文算法不仅充分利用障碍上的布线资源，节约了障碍外部的布线资源，而且有效缩短了布线所需的总线长，不仅可以节约 buffer 资源，也可以降低电路功耗。

表 2 本文算法解决 LRSMT 问题得到的线长

Table II The wire length of our algorithm for LRSMT

测试电路	OARSMT[5]	外部斯坦纳树的总线长					斯坦纳树的总线长				
		0	LBB*1%	LBB*5%	LBB*10%	∞	0	LBB*1%	LBB*5%	LBB*10%	∞
IND1	604	614	614	584	584	584	614	614	604	604	604
IND2	9600	10900	10900	7900	7600	7600	10900	10900	9100	9100	9100
IND3	600	678	678	560	524	524	678	678	600	590	590
IND4	1092	1155	1155	976	906	919	1155	1155	1092	1092	1092
IND5	1353	infeas.	infeas.	1160	1075	1046	infeas.	infeas.	1325	1313	1315
RC01	25980	25980	25980	24960	24960	24960	25980	25980	25290	25290	25290
RC02	41350	42590	42590	41620	39700	35070	42590	42590	42030	40690	40060
RC03	54360	54660	54660	53120	51480	49350	54660	54660	53240	53330	52340
RC04	59530	59980	59980	56100	53870	52560	59980	59980	57100	55720	55570
RC05	74720	75320	75320	72200	70750	66080	75320	75320	73150	72730	72170
RC06	81290	81697	79890	65792	64457	64457	81697	80777	77768	77488	77488
RC07	110851	112194	108893	95238	94207	94207	112194	110126	107382	107210	107210
RC08	115516	116176	110129	93005	92510	92510	116176	112992	109344	109104	109104
RC09	113254	116313	110693	88090	87299	87299	116313	113439	107314	107135	107135
RC10	166970	167850	167760	159590	159530	159530	167850	167760	165410	165350	165350
RC11	234875	235652	234478	232741	232741	232741	235652	234961	234531	234531	234531
RC12	758717	762357	754879	754879	754879	754879	762357	759910	759910	759910	759910
RT01	2193	2200	1795	1619	1619	1619	2200	1871	1817	1817	1817
RT02	46965	48035	44976	44521	44521	44521	48035	45718	45690	45690	45690
RT03	8136	8281	7583	7483	7430	7430	8281	7759	7738	7737	7737
RT04	9832	10345	7452	7136	7136	7136	10345	7840	7788	7788	7788
RT05	52318	54456	40842	37558	37333	37333	54456	43885	43477	43465	43465
AVE		-2.39	3.15	13.70	16.06	17.44	-2.39	1.59	4.88	5.31	5.51

LBB 表示测试电路的矩形布线区域边界较长边长。

infeas 表示在当前约束下，该测试电路无解。

表 3 本文算法解决 LRSMT 问题与[16]算法对比

Table III Comparison between our algorithm for LRSMT and [10]

	0	LBB*1%	LBB*5%	LBB*10%	∞
平均相对改进百分比	1.53	1.87	1.94	2.10	-0.04
总运行时间(秒)	0.82	0.78	0.76	0.75	0.76
总运行时间(秒)[16]	1.36	0.68	0.38	0.38	0.18

表 4 本文算法解决 OARSMT_SC 问题，所用到的具体技术参数

Table IV the specific technology parameter for OARSMT_SC

R_b	K_b	c_b	r_b	r_0	c_0
300 fs/fF	60000fs	3.8 fF	450 Ω	0.56 Ω/μm	0.48 fF/μm

表 5 本文算法解决 OARSMT_SC 问题得到的线长

Table V The wire length of our algorithm for OARSMT_SC

测试电路	OARSMT[5]	外部斯坦纳树的总线长					斯坦纳树的总线长				
		0 slew	20% slew	50% slew	80% slew	∞ slew	0 slew	20% slew	50% slew	80% slew	∞ slew
IND1	604	614	614	604	591	584	614	614	614	604	604
IND2	9600	10900	10900	10400	7900	7600	10900	10900	10500	9100	9100
IND3	600	678	678	560	560	524	678	678	600	600	590
IND4	1092	1155	1007	945	906	919	1155	1097	1092	1092	1092
IND5	1353	infeas.	1277	1168	1064	1046	infeas.	1357	1333	1320	1315
RC01	25980	25980	25980	24960	24960	24960	25980	25980	25290	25290	25290
RC02	41350	42590	39700	39700	39700	35070	42590	40690	40690	40690	40060
RC03	54360	54660	53120	53120	51480	49350	54660	53240	53240	53330	52340
RC04	59530	59980	56840	53870	53870	52560	59980	57360	55720	55720	55570
RC05	74720	75320	72200	70750	69210	66080	75320	73150	72730	72680	72170
RC06	81290	81697	74677	67086	64939	64457	81697	78984	77887	77612	77488
RC07	110851	112194	104784	95295	94207	94207	112194	108701	107484	107210	107210
RC08	115516	116176	104768	97529	93005	92510	116176	111100	110123	109344	109104
RC09	113254	116313	101755	89761	87299	87299	116313	109982	107629	107135	107135
RC10	166970	167850	163950	160130	159980	159530	167850	165720	165450	165560	165350
RC11	234875	235652	234478	233584	233132	232741	235652	234961	234609	234632	234531
RC12	758717	762357	760400	755470	755534	754879	762357	761324	760280	760470	759910
RT01	2193	2200	1830	1685	1656	1619	2200	1911	1827	1817	1817
RT02	46965	48035	44976	44976	44796	44521	48035	45718	45718	45690	45690
RT03	8136	8281	7476	7483	7483	7430	8281	7723	7738	7738	7737
RT04	9832	10401	7306	7179	7136	7136	10401	7809	7791	7788	7788
RT05	52318	54540	39214	38127	37283	37333	54540	43641	43535	43465	43465
AVE		-2.42	6.12	11.93	15.33	17.44	-2.42	2.81	4.35	5.19	5.51

表 6 本文算法解决 OARSMT_SC 问题所耗的运行时间(单位：秒)

Table VI Running times in seconds of our algorithm for OARSMT_SC

	0 slew	20% slew	50% slew	80% slew	∞ slew
总运行时间(秒)	0.83	0.89	0.90	0.89	0.87

5.结论

本文提出一种考虑复杂直角障碍中布线资源再利用的斯坦纳树构造方法。该算法以扩展直角满斯坦纳树网格为布线图，设计了考虑电压转换速率和限制长度约束的 SPH 算法，并避免了频繁计算电压转换速率。为了提高求解质量，根据斯坦纳点所处位置的特点，在布线图标记了两种类型的候选斯坦纳点，分别用于构造斯坦纳树和以插入关键节点局部搜索为基础的改善过程。实验结果表明，使用 OARSMT_SC 模型所消耗的运行时间几乎与 LRSMT 模型相同，但是提供了更精确的电压转换速率的计算。和传统 OARSMT 算法相比，本文算法充分利用障碍上的布线资源，节约了障碍外部的布线资源，而且有效缩短了布线所需的总线长。与同类算法相比，本文算法在合理的运行时间内，有效提高了求解质量。

参考文献

1. Hanan M (1966) On Steiner's Problem with Rectilinear Distance. SIAM Journal on Applied Mathematics 14 (2):255-265. doi:10.1137/0114025

2. Chu C, Yiu-Chung W (2008) FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27 (1):70-83. doi:10.1109/TCAD.2007.907068

3. Garey M, Johnson D (1977) The Rectilinear Steiner Tree Problem is NP-Complete. SIAM Journal on Applied Mathematics 32 (4):826-834. doi:doi:10.1137/0132071

4. Tao H, Liang L, Young EFY (2011) On the Construction of Optimal Obstacle-Avoiding Rectilinear Steiner Minimum Trees. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 30 (5):718-731. doi:10.1109/TCAD.2010.2098930

5. Liu C-H, Kuo S-Y, Lee DT, Lin C-S, Weng J-H, Yuan S-Y (2012) Obstacle-Avoiding Rectilinear Steiner Tree Construction: A Steiner-Point-Based Algorithm. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 31 (7):1050-1060. doi:10.1109/TCAD.2012.2185050

6. Ganley JL, Cohoon JP (1994) Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. Circuits and Systems, 1994 ISCAS '94, 1994 IEEE International Symposium on 1:113-116. doi:10.1109/ISCAS.1994.408768

7. Shen Z, Chu CCN, Ying-Meng L (2005) Efficient rectilinear Steiner tree construction with rectilinear blockages. Computer Design: VLSI in Computers and Processors, 2005 ICCD 2005 Proceedings 2005 IEEE International Conference on:38-44. doi:10.1109/ICCD.2005.45

8. Zhang H, Ye D-Y (2014) Key-node-based local search discrete artificial bee colony algorithm for obstacle-avoiding rectilinear Steiner tree construction. Neural Comput & Applic:1-24. doi:10.1007/s00521-014-1760-4

9. Chow W-K, Li L, Young EFY, Sham C-W (2014) Obstacle-avoiding rectilinear Steiner tree construction in sequential and parallel approach. Integration, the VLSI Journal 47 (1):105-114. doi:http://dx.doi.org/10.1016/j.vlsi.2013.08.001

10. Li L, Young EFY (2008) Obstacle-avoiding rectilinear Steiner tree construction. Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design:523-528

11. Lin CW, Chen S-Y, Chi-Feng L, Yao-Wen C, Chia-Lin Y (2008) Obstacle-Avoiding Rectilinear Steiner Tree Construction Based on Spanning Graphs. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27 (4):643-653. doi:10.1109/TCAD.2008.917583
12. 朱祺;王垠; 杨经洪 (2005) 考虑障碍的多端点最小直角 Steiner 树构造算法. 计算机辅助设计与图形学学报 17 (2):223-230
13. 刘耿耿;王小溪;陈国龙;郭文忠;王少铃; (2010) 求解 VLSI 布线问题的离散粒子群优化算法. 计算机科学 37 (10):197-201
14. Müller-Hannemann M, Peyer S (2003) Approximation of Rectilinear Steiner Trees with Length Restrictions on Obstacles. In: Dehne F, Sack J-R, Smid M (eds) Algorithms and Data Structures, vol 2748. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 207-218. doi:10.1007/978-3-540-45078-8_19
15. Müller-Hannemann M, Schulze A (2006) Approximation of Octilinear Steiner Trees Constrained by Hard and Soft Obstacles. In: Arge L, Freivalds R (eds) Algorithm Theory – SWAT 2006, vol 4059. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 242-254. doi:10.1007/11785293_24
16. Held S, Spirkl ST (2014) A fast algorithm for rectilinear steiner trees with length restrictions on obstacles. Proceedings of the 2014 on International symposium on physical design:37-44. doi:10.1145/2560519.2560529
17. Yilin Z, Chakraborty A, Chowdhury S, Pan DZ Reclaiming over-the-IP-block routing resources with buffering-aware rectilinear Steiner minimum tree construction. In: Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on, 5-8 Nov. 2012 2012. pp 137-143
18. Huang T, Young EFY (2012) Construction of rectilinear Steiner minimum trees with slew constraints over obstacles. Paper presented at the Proceedings of the International Conference on Computer-Aided Design, San Jose, California,
19. Mehlhorn K (1988) A faster approximation algorithm for the Steiner problem in graphs. Information Processing Letters 27 (3):125-128. doi:http://dx.doi.org/10.1016/0020-0190(88)90066-X
20. Clarkson K, Kapoor S, Vaidya P (1987) Rectilinear shortest paths through polygonal obstacles in O(n(logn)²) time. Proceedings of the third annual symposium on Computational geometry:251-257. doi:10.1145/41958.41985
21. Kashyap CV, Alpert CJ, Liu F, Devgan A (2002) PERI: a technique for extending delay and slew metrics to ramp inputs. Proceedings of the 8th ACM/IEEE international workshop on Timing issues in the specification and synthesis of digital systems:57-62. doi:10.1145/589411.589424
22. Warme D, Winter P, Zachariasen M (2001) GeoSteiner Software for Computing Steiner Trees. http://www.diku.dk/hjemmesider/ansatte/martinz/geosteiner/.
23. H.B.Bakoglu (1990) Circuits, interconnections, and packaging for VLSI.5
24. Elmore WC (1948) The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. Journal of Applied Physics 19 (1):55-63. doi:doi:http://dx.doi.org/10.1063/1.1697872
25. Kashyap CV, Alpert CJ, Liu F, Devgan A (2004) Closed-form expressions for extending step delay and slew metrics to ramp inputs for RC trees. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 23 (4):509-516. doi:10.1109/TCAD.2004.825861
26. Shiyan H, Alpert CJ, Jiang H, Karandikar SK, Zhuo L, Weiping S, Sze CN (2007) Fast Algorithms for Slew-Constrained Minimum Cost Buffering. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26 (11):2009-2022. doi:10.1109/TCAD.2007.906477
27. TSUNG-YI HO Y-WC, SAO-JIE CHEN (2007) Full-Chip Nanometer Routing Techniques. Springer Netherlands. doi:10.1007/978-1-4020-6195-0_4
28. Zachariasen M (1999) Rectilinear Full Steiner Tree Generation. Networks, 33:125–143
29. LAWLER EL (1976) The Steiner Problem and Other Dilemmas. In: Holt RaW (ed) Combinatorial Optimization: Networks and Matroids. New York, pp 290-296

30. Takahashi H, Matsuyama A (1980) An approximate solution for the Steiner problem in graphs. Math Japonica 6 (24):573-577
31. Aragão M, Werneck R (2002) On the Implementation of MST-Based Heuristics for the Steiner Problem in Graphs. In: Mount D, Stein C (eds) Algorithm Engineering and Experiments, vol 2409. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp 1-15. doi:10.1007/3-540-45643-0_1
32. Zhang Y, Pan DZ (2014) Timing-driven, over-the-block rectilinear steiner tree construction with pre-buffering and slew constraints. Proceedings of the 2014 on International symposium on physical design:29-36. doi:10.1145/2560519.2560533

国家重点基础研究项目特别基金资助的课题(No. 2011CB808000)、国家自然科学基金(No. 11271002)资助
作者简介 张浩，男，1981，博士生，讲师，主要研究方向为 VLSI 布线算法、启发式算法、计算智能算法等，zhanghao@fzu.edu.cn。叶东毅，男，1964，博士生导师，教授，CCF 会员，主要研究方向为数据挖掘、计算智能算法等。郭文忠，男，1979，博士生导师，教授，CCF 会员，主要研究方向为计算智能算法、VLSI 物理设计算法研究。