第 3 章

X 结构 Steiner 最小树算法

3.1 引言

作为超大规模集成电路物理设计中一个关键的环节，总体布线面临诸多复杂的难题。Steiner 最小树(Steiner Minimal Tree，SMT)问题是在给定引脚集合的基础上通过引入一些额外的点(Steiner 点)以寻找一棵连接给定引脚集合的最小代价布线树。因为 Steiner 最小树模型是 VLSI 总体布线中多端线网的最佳连接模型，所以 Steiner 最小树构建是 VLSI 布线中一个关键环节。

总体布线的布线算法主要基于曼哈顿结构以及非曼哈顿结构两种。其中，非曼哈顿结构因其更强的布线优化能力受到越来越多研究人员的关注。为了更好地研究非曼哈顿结构布线，首要工作是研究非曼哈顿结构 Steiner 最小树的构建问题。本章 3.2 节～3.6 节分别介绍了 5 种有效的 X 结构 Steiner 最小树(X-architecture Steiner Minimum Tree，XSMT)算法，并取得了不错的优化效果。

本章涉及的专有名词符号见表 3.1。

表 3.1 专用名词符号表

缩 写	英 文 全 称	中 文 全 称
VLSI	Very Large Scale Integration circuit	超大规模集成电路
EDA	Electronic Design Automation	电子设计自动化
SMT	Steiner Minimal Tree	Steiner 最小树
RSMT	Rectilinear Steiner Minimal Tree	矩形 Steiner 最小树
XSMT	X-architecture Steiner Minimal Tree	X 结构 Steiner 最小树
WL-XSMT	WireLength-driven X-architecture Steiner Minimum Tree	线长驱动 X 结构 Steiner 最小树
PSO	Particle Swarm Optimization	粒子群优化
SLPSO	Social Learning Particle Swarm Optimization	社会学习粒子群优化

续表

缩　写	英 文 全 称	中 文 全 称
SLDPSO	Social Learning Discrete Particle Swarm Optimization	社会学习离散粒子群优化
DE	Differential Evolution	差分进化
DDE	Discrete Differential Evolution	离散差分进化
MDDE	Multi-strategy optimization Discrete Differential Evolution	多策略优化离散差分进化
MA	Memetic Algorithm	文化基因算法

XSMT 通过引入 Steiner 点以最小的代价连接给定的引脚集合，这里连接线可走线方向除了包括传统的水平方向和垂直方向，还允许 45°和 135°的绕线方向。

XSMT 问题中给定 $P=\{P_1,P_2,P_3,\cdots,P_n\}$作为待布线网的 n 个引脚集合，且每个 P_i 对应一个坐标对(x_i,y_i)。例如有一个待布线网有 8 个引脚，表 3.2 给出了引脚的输入信息，相应的引脚版图分布如图 3.1 所示，例如，其中编号为 1 的引脚对应的坐标对信息为表 3.2 第二列所示的(33,33)。

表 3.2　待布线网的引脚输入信息

编　号	1	2	3	4	5	6	7	8
X 坐标	33	2	42	47	34	38	37	20
Y 坐标	33	9	35	2	1	2	5	40

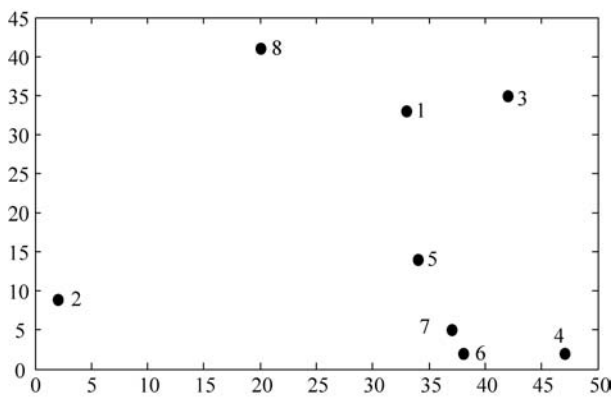


图 3.1　待布线网的引脚分布情况

对 X 结构的 4 种 pseduo-Steiner 点选择方式的具体定义已在前面给出，详见 2.5.2 节。

3.2 基于离散 PSO 的 X 结构 Steiner 最小树算法

由于 Steiner 最小树问题是一个 NP 难问题，所以一些在求解 NP 难问题中展现出良好应用前景的进化算法[包括粒子群优化算法（Particle Swarm Optimization，PSO）和蚁群算法（Ant Colony Optimization，ACO）]被用于求解 RSMT 和 XSMT 问题。作为一类基于种群进化的方法，粒子群优化算法于 1995 年由 Eberhart 和 Kennedy 共同提出。PSO 算法相对其他优化算法而言，操作简单且能快速收敛至一个合理、优秀的解方案。

基于以上相关研究工作的分析，本节设计并实现一种基于离散粒子群优化的有效算法，即 XSMT_PSO 算法，用来求解 X 结构 Steiner 最小树的构建问题。首先，为了解决 PSO 算法在高维问题中收敛慢的现象。其次，根据 X 结构 Steiner 最小树的特点，受遗传算法中的变异算子和交叉算子的启发，提出了离散化的 PSO 更新操作策略。然后，设计了一种适合于 X 结构 Steiner 最小树问题的有效编码策略。最后，设计相应的仿真实验，通过实验结果证明本节算法的可行性和有效性。

XSMT_PSO 算法的流程图如图 3.2 所示。

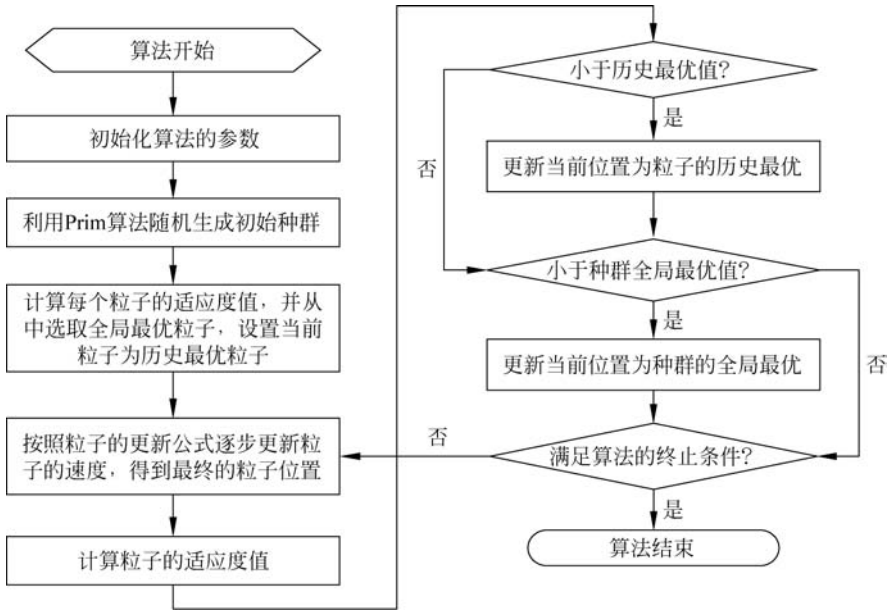


图 3.2 本节算法的流程图

3.2.1　XSMT_PSO 算法

对于基本 PSO 算法的具体介绍已在前面给出，详见 2.3.2 节。

基本 PSO 算法主要运用于解决连续优化问题，而 XSMT 问题的解空间离散，因此需要构造一种离散形式的 PSO 算法模型。现有文献应用 PSO 解决离散问题，主要有将速度作为位置变化的概率、直接将连续 PSO 用于离散问题的求解以及重新定义 PSO 操作算子 3 种策略。为此，本节设计了适合 XSMT 问题的有效离散 PSO 算法，即 XSMT_PSO 算法。

1. 粒子的编码策略

XSMT_PSO 算法采用边点对编码。在该编码方式中，候选 XSMT 的一条边由边的两个端点与边对应的 pseudo-Steiner 点选择方式。pseudo-Steiner 点选择方式是将生成树的边转化成 X 结构 Steiner 树(X-architecture Steiner Tree，XST)的 X 结构边。对于一个含 n 个引脚的线网，其每棵候选的 XST 都由 $n-1$ 条边构成，每条边有两个引脚以及一个对应的 pseudo-Steiner 点选择方式，同时用一位数字表示该 XST 的适应度值，所以每个粒子编码的总长度为 $3\times(n-1)+1$。例如，一个候选 XST($n=8$)可用 XSMT_PSO 算法中一个粒子的编码表示，其编码采用如下所示的数字串表示：

7 6 *0* 6 4 *1* 7 5 *1* 5 1 *2* 1 3 *0* 1 8 *1* 5 2 *2* 10.0100

其中，最后一个数“10.0100”为该粒子的适应度值，各个斜体数字为对应边的 pseudo-Steiner 点选择方式。对第二个数字子串(6 4 *1*)而言，编号 6 和 4 的引脚以及 pseudo-Steiner 点选择方式 1 选择构成了该边。

2. 粒子的适应度函数

定义 3.1　一棵 X 结构 Steiner 最小树的长度是该布线树中所有边片段的长度总和，其计算方式如下：

$$L(T_X)=\sum_{e_i\in T_X} l(e_i) \tag{3.1}$$

其中，$l(e_i)$表示在布线树 T_X 中每个边片段 e_i 的长度。

根据 X 结构 Steiner 树的 4 种互连线选择方式，XST 所有边片段可分为以下 4 种类型：水平边片段、垂直边片段、45°斜边片段及 135°斜边片段。为计算方便，将 45°边片段顺时针方向旋转为水平边；同理，将 135°边片段顺时针方向旋转为垂直边，使得所有边片段都可分为水平和垂直两种。最后将水平边根据其左引脚的大小按从下往上、从左到右的方式排列，垂直边则根据其下引脚的大小按从左到右、从下到上的方式排列。依照上述方式，求解 XST 的长度只需要计算所有不重复的边片段即可。

本节算法中的粒子的适应度函数设计如下：

$$\text{fitness}=\frac{1}{L(T_X)+1} \tag{3.2}$$

其中，分母做布线树的长度加 1 的操作是防止出现布线树的长度为 0 的情况，即两端线网中两个引脚的位置重合。可以看出，布线树长度越小，即粒子适应度值越大，则粒子越优。

3. 粒子的更新公式

XSMT_PSO 算法结合遗传算法的变异和交叉操作改变了基本 PSO 粒子的更新方式以实现构建 X 结构 Steiner 最小树这一离散问题。粒子的更新公式如下：

$$X_i^t = N_3(N_2(N_1(X_i^{t-1}, w), c_1), c_2) \tag{3.3}$$

其中，w 是惯性权重因子，c_1 和 c_2 是加速因子，N_1 表示变异操作，N_2 和 N_3 表示交叉操作，r_1、r_2、r_3 都是在区间[0,1)的随机数。

1) 粒子的速度更新

粒子的速度更新公式如下：

$$W_i^t = N_1(X_i^{t-1}, w) = \begin{cases} M(X_i^{t-1}), & r_1 < w \\ X_i^{t-1}, & \text{否则} \end{cases} \tag{3.4}$$

其中，w 表示粒子进行变异操作的概率。

在式(3.4)中，变异操作的具体过程为：以 3.1 节中 $n=8$ 的候选布线树为例，如图 3.3 所示，取其中的 $n-1$ 位 pseudo-Steiner 点选择方式作为变异序列，即长度为 7 的序列，序列中每个位置只能取 0～3 的整数。此时，随机产生一个 0～1 的数 r_1，若小于变异概率 w，则随机选择序列中一个位置(mp1)，将 mp1 上的值更新为 0～3 的整数，但需与 mp1 位置的原值不一样。如图 3.3 所示，mp1 位置上值由 2 更新为 1，此时变异操作结束。

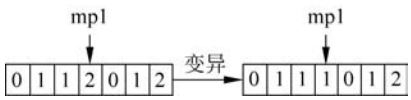


图 3.3 本节算法的变异操作

2) 粒子的个体经验学习

粒子的个体经验学习公式如下：

$$S_i^t = N_2(W_i^t, c_1) = \begin{cases} C_p(W_i^t), & r_2 < c_1 \\ W_i^t, & \text{否则} \end{cases} \tag{3.5}$$

其中，c_1 表示粒子与其历史最优进行交叉操作的概率。

3) 粒子与种群其他粒子进行合作学习

粒子与种群其他粒子进行合作学习的公式如下：

$$X_i^t = N_3(S_i^t, c_2) = \begin{cases} C_p(S_i^t), & r_3 < c_2 \\ S_i^t, & \text{否则} \end{cases} \tag{3.6}$$

其中，c_2 表示粒子与全局最优方案进行交叉操作的概率。

在式(3.5)和式(3.6)中，粒子的交叉操作具体可描述为：在变异操作后，随机产生一个 0～1 的数 $r_2(r_3)$，若其小于交叉概率 $c_1(c_2)$，粒子的序列需与其历史最优方案(种群全局最优方案)进行交叉操作。如图 3.4 所示，随机产生两个交叉位置(cp1 和 cp2)，将本粒子中 cp1 和 cp2 的值替换为其历史最优方案(种群全局最优方案)在该区间的值，得到如图 3.4 右半部分所示交叉后的粒子编码情况。

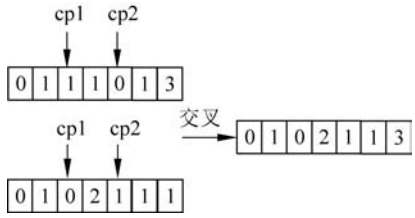


图 3.4　本节算法的交叉操作

4. XSMT_PSO 算法的参数设置

在粒子群优化算法中，参数的设置对于算法性能有很大的影响。其中，惯性权重 w 决定了粒子受前一时刻自身速度的影响程度，加速因子 c_1 和 c_2 控制粒子对于最优位置的学习步长。本节算法的 3 个参数均采用 Shi 和 Eberhart 所提的线性递减策略进行设置。其中，采用该策略的惯性权重使种群在迭代早期保持较好的全局搜索能力，在迭代后期保证一定的局部搜索能力；而线性递减的 c_1 及线性递增的 c_2 可保障算法的早期可在局部范围内进行详细搜索，使其不会在早期直接移动至下一个局部最优位置，同时在后期加快算法的收敛速度。

本节算法具体参数设置为惯性权重因子 w 从 0.9 线性递减至 0.1，加速因子 c_1 从 0.9 线性递减至 0.2，加速因子 c_2 从 0.4 线性递增至 0.9，每次迭代中相应的参数设置具体按照式(3.7)～式(3.9)进行更新。

$$w = \mathrm{w\_start} - \frac{\mathrm{w\_start} - \mathrm{w\_end}}{\mathrm{evaluations}} \times \mathrm{eval} \tag{3.7}$$

$$c_1 = \mathrm{c_1\_start} - \frac{\mathrm{c_1\_start} - \mathrm{c_1\_end}}{\mathrm{evaluations}} \times \mathrm{eval} \tag{3.8}$$

$$c_2 = \mathrm{c_2\_start} - \frac{\mathrm{c_2\_start} - \mathrm{c_2\_end}}{\mathrm{evaluations}} \times \mathrm{eval} \tag{3.9}$$

其中，w_start、$\mathrm{c_1}$_start、$\mathrm{c_2}$_start 表示参数 w、c_1、c_2 迭代开始的初始值，w_end、$\mathrm{c_1}$_end、$\mathrm{c_2}$_end 表示参数 w、c_1、c_2 迭代的最终值，eval 表示当前迭代次数，evaluations 则表示算法的最大迭代次数。

5. XSMT_PSO 算法的复杂度分析

引理 3.1　假设种群大小为 p，迭代次数为 iters，引脚的个数为 n，则 XSMT_PSO 算法的复杂度为 $O(\mathrm{iters} \times p \times n\log n)$。

证明：在 XSMT_PSO 算法的内部循环包含变异操作以及对适应度值的计算。其中，变异和交叉操作只变换 pseudo-Steiner 点选择方式，时间为常数时间。而在计算适应度(即布线树线长)时，主要由排序方法的复杂度决定。因此，算法内部循环的复杂度为 $O(n\log n)$。而外部循环的复杂度主要取决于种群规模以及算法的迭代次数，因此 XSMT_PSO 算法的复杂度为 $O(\text{iters}\times p\times n\log n)$。

3.2.2 实验仿真与结果分析

本节算法的参数设置如下：种群大小为 50，最大迭代次数为 1000。为了验证本节算法的有效性，进行两组实验对比，其中本节算法在每个测试实例中各执行 10 次并取最优值。在两组实验的实验结果中，RSMT 代表采用文献[3]的算法构造矩形 Steiner 最小树的长度，而 XSMT 则是采用本节算法构建的 X 结构 Steiner 最小树的长度。在第一组实验中，采用 OR-Library 测试数据用以对比文献[3]的算法和本节算法构造 Steiner 最小树的长度。如表 3.3 所示，给出了在 46 个测试实例中本节算法所构造的 XSMT 的长度值和文献[3]的算法所构造的 RSMT 的长度值以及它们的对比改进情况。如表 3.3 所示，可发现本节算法构造的 XSMT 的长度值相对于文献[3]的算法所构造的 RSMT 的长度值取得了平均 9.84%的优化率。而文献[6]设计了精确算法构建 XSMT，并指出了 XSMT 的长度一般比 RSMT 的长度优化 10%左右，同时文献[6]中算法的测试实例也是来源于 OR-Library 测试数据，并得到 XSMT 长度比 RSMT 长度的优化率范围一般(9.75±2.29)%。而本节算法的平均优化率 9.84%处于该优化率范围内。

表 3.3 在 46 个测试实例中 XSMT 和 RSMT 的长度对比情况

编　　号	RSMT	XSMT	优化率/%
1	1.87	1.7646	5.64
2	1.68	1.5664	6.76
3	2.36	2.1855	7.39
4	2.54	2.2382	11.88
5	2.29	2.1385	6.62
6	2.48	2.3285	3.79
7	2.54	2.3475	7.58
8	2.42	2.2685	6.26
9	1.72	1.6543	3.82
10	1.85	1.7057	7.80
11	1.44	1.3511	6.17
12	1.80	1.6828	6.51
13	1.50	1.3243	11.71
14	2.60	2.3657	9.01
15	1.48	1.2895	12.87

续表

编　　号	RSMT	XSMT	优化率/%
16	1.60	1.2485	21.97
17	2.01	1.6920	15.82
18	4.06	4.0566	0.08
19	1.93	1.8128	6.07
20	1.12	1.0685	4.60
21	2.16	1.9188	11.17
22	0.63	0.5363	14.87
23	0.65	0.5277	18.81
24	0.30	0.2680	10.67
25	0.23	0.2097	8.83
26	0.15	0.1290	14.00
27	1.33	1.2070	9.25
28	0.28	0.2156	23.00
29	2.00	1.4728	26.36
30	1.10	1.1000	0.00
31	2.66	2.4870	6.50
32	3.30	3.0381	7.94
33	2.69	2.3317	13.32
34	2.54	2.2682	10.70
35	1.57	1.4243	9.28
36	0.90	0.9000	0.00
37	0.90	0.8070	10.33
38	1.66	1.4808	10.80
39	1.66	1.4808	10.80
40	1.62	1.5204	6.15
41	2.24	2.0634	7.89
42	1.53	1.3833	9.59
43	2.68	2.5443	5.76
44	2.61	2.2910	12.22
45	2.26	2.0881	7.61
46	1.50	1.5000	0.00
均值			9.84

在第二组实验中，本节使用文献[5]提及的网站 rand_points 产生 10 个随机测试实例，并分别构造相应的 RSMT 和 XSMT。每个测试实例的引脚规模和实验结果在表 3.4 中给出，从中可看到本节算法构造的 XSMT 取得了 9.28%的长度优化率。该组实验的平均线长优化率也处于文献[6]给定的(9.75±2.29)%的优化率范围内。

表 3.4　在随机生成的 10 个测试实例中 XSMT 和 RSMT 的长度对比情况

实　例	引　脚　数	RSMT	XSMT	优化率/%
1	8	17 931	17 040	5.0
2	9	20 503	18 163	11.4
3	10	21 910	19 818	9.5
4	20	35 723	32 199	9.9
5	50	53 383	47 960	10.2
6	70	61 987	55 980	9.7
7	100	76 016	68 743	9.6
8	410	156 520	142 880	8.7
9	500	170 273	154 290	9.4
10	1000	245 201	222 050	9.4
均值				9.28

图 3.5 给出了一个测试实例，其引脚数为 10 个，通过多次运行本节算法可以得到 5 种不同拓扑结构（总共可得到拓扑的种类不止 5 种，在图 3.5 中简单给出 5 种拓扑），而且它们的长度值都是一样的。可见当最优值相同或近似时，XSMT_PSO 可以获得更多的拓扑结构，为总体布线进行拥挤度优化工作提供不同的拓扑选择方案。

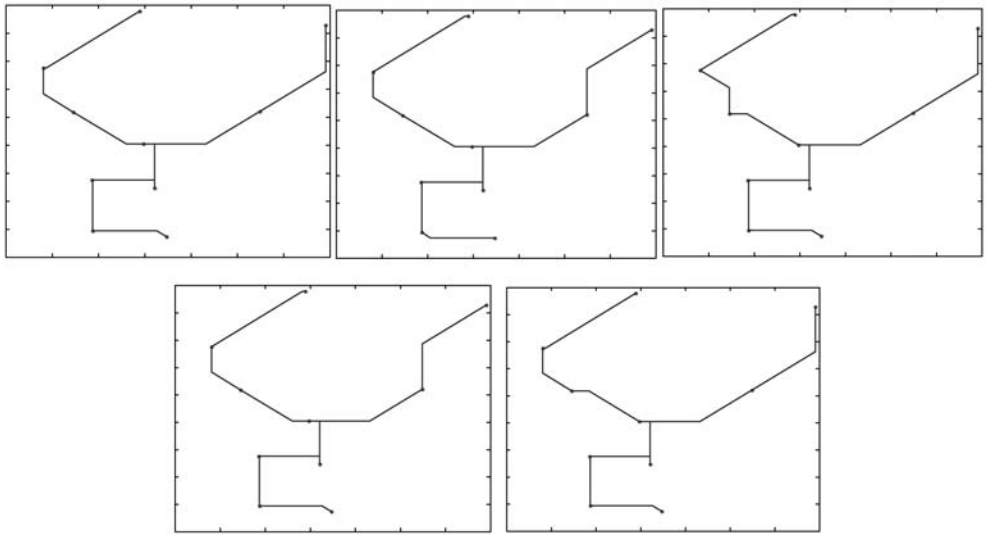


图 3.5　引脚数为 10 的实例可取得线长一样的 5 种不同拓扑结构

综上所述，两组实验均可取得跟文献[10]的算法相似的线长优化率，而且本节算法还可得到多种不同拓扑的 RSMT，这些不同的拓扑对 VLSI 总体布线的拥挤度优化工作可带来一定的帮助和指导。

3.2.3　小结

本节基于 PSO 提出了一种有效的 XSMT 构造算法，相对矩形 Steiner 最小树可更有效地优化线长，并且可获得多种拓扑结构，从而有利于对 VLSI 总体布线阶段的拥挤度进行优化。

3.3　基于离散差分进化的 X 结构 Steiner 最小树算法

3.3.1　传统差分进化算法

差分进化(Differential Evolution，DE)算法由 Storn 和 Price 于 1995 年首次提出，是一种基于种群的全局进化算法，其本质是一种高效的多目标全局优化算法，用于求解多维空间中的整体最优解，具有操作简单的特点和良好的优化能力。原始的差分进化算法通过变异、交叉、选择 3 个操作，不断迭代进化，使种群向全局最优解移动。

1. 传统差分进化算法粒子更新策略

1) 种群初始化

在解空间随机生成 M 个个体，每个个体由 D 维向量组成，第 i 个个体的第 j 维的取值方式如下：

$$X_i^j = L_{\text{j\_min}} + \text{rand}(0,1) \times (L_{\text{j\_max}} - L_{\text{j\_min}}) \tag{3.10}$$

其中，$L_{\text{j\_min}}$ 和 $L_{\text{j\_max}}$ 分别为第 j 维值的最小值及最大值范围。

2) 变异算子

在第 g 次迭代过程中，变异策略为

$$V_i(g) = X_{p1}(g) + F \times (X_{p2}(g) - X_{p3}(g)) \tag{3.11}$$

其中，$p_1 \neq p_2 \neq p_3 \neq i$，$F$ 为缩放因子在[0，2]区间选择。

其他变异算子包括：

$$V_i(g) = X_{\text{best}}(g) + F \times (X_{p1}(g) - X_{p2}(g)) \tag{3.12}$$

$$V_i(g) = X_i(g) + F \times (X_{\text{best}}(g) - X_i(g)) + F \times (X_{p1}(g) - X_{p2}(g)) \tag{3.13}$$

$$V_i(g) = X_{\text{best}}(g) + F \times (X_{p1}(g) - X_{p2}(g)) + F \times (X_{p3}(g) - X_{p4}(g)) \tag{3.14}$$

$$V_i(g) = X_{p1}(g) + F \times (X_{p2}(g) - X_{p3}(g)) + F \times (X_{p4}(g) - X_{p5}(g)) \tag{3.15}$$

3) 交叉算子

差分进化算法在交叉过程的交叉策略为

$$v_i^j=\begin{cases}h_i^j(g), & \text{rand}(0,1)\leqslant \text{cr}\\ x_i^j(g), & \text{否则}\end{cases} \tag{3.16}$$

其中，cr 为交叉概率，且 cr∈[0,1]。

4）选择算子

选择过程的策略即为贪心策略，即

$$X_i(g+1)=\begin{cases}V_i(g), & f(V_i(g))<f(X_i(g))\\ X_i(g), & \text{否则}\end{cases} \tag{3.17}$$

传统差分进化算法流程如图 3.6 所示。

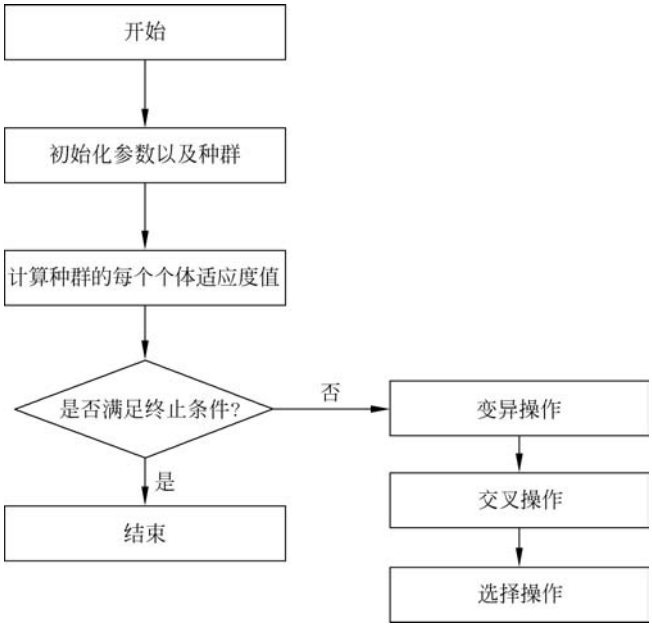


图 3.6 传统差分进化算法流程

2. 传统差分进化算法伪代码

如算法 3.1 所示，传统差分算法主要分为两个步骤：第一，初始化相关参数并初始化种群；第二，进入迭代过程，在每次迭代中，对每个个体做变异操作和交叉操作，获得一个新的试验个体，根据贪心选择，选择试验个体和父代中最优的个体进入下一代，如此循环，直至迭代结束或者满足算法结束的条件，从而获得一个全局最优解。

3.3.2 算法设计

1. 编码策略及适应度值函数

本节算法同样采用边点对编码策略，详细介绍见 3.2.1 节。

一棵 XSMT 的长度是该树所有边集长度的总和，详见式(3.1)。算法的适应度值求解函数进行预处理，将一棵 XSMT 的边分为 45°、135°、水平和垂直 4 种类型，再将 45°斜边与 135°斜边分别以顺时针和逆时针的方式旋转，转换成水平边和垂直边。记录所有的水平边和垂直边并进行排序，扣除共享的边之后，将所有边加起来即为该 XSMT 的线长值，因此本节算法中衡量一棵 XSMT 的适应度值函数设计如下：

$$f = L(T_X) \tag{3.18}$$

算法 3.1　DE 算法

输入：种群大小 M；维度 D；迭代次数 T；

输出：最优解向量 $\boldsymbol{\Delta}$

```
1. (initialization);
2. for i=1 to M do
3.    for j=1 to D do
4.       x^j_(i,t) = x^j_min + rand(0,1) · (x^j_max − x^j_min);
5.    end
6. end
7. while(|f(Δ)|≥ε) or(t≤T) do
8.    for i=1 to M do
9.       for j=1 to D do
10.         v^j_(i,t) = Mutation(x^j_(i,t));
11.         u^j_(i,t) = Crossover(x^j_(i,t), v^j_(i,t));
12.      end
13.      if f(u_(i,t)) < f(x_(i,t)) then
14.         x_(i,t) ← u_(i,t);
15.         if f(x_(i,t)) < f(Δ) then
16.            Δ ← x_(i,t);
17.            end
18.         else
19.            x_(i,t) ← x_(i,t);
20.         end
21.   end
22.   t ← t+1;
23. end
24. return the best vector;
```

2. 初始化策略

传统差分进化算法直接采用式(3.10)进行随机初始化。针对 XSMT 构建问题，如果采用随机初始化去生成种群的每个个体，将导致算法求解空间过大而无法

收敛的情形，故本节采用最小生成树生成法对种群个体进行初始化，具体采用Prim算法，伪代码如算法3.2所示，算法中部分参数意义如下。

T：最小生成树集合，S：起始引脚点集合，V：引脚点集合(1～N)。

算法3.2 Prim算法

输入：待布线引脚点集合 N

1. 随机选取起点 s
2. $T=\varnothing$;
3. $S=\{s\}$;
4. **while**(S !$=V$) **do**
5. $(i,j)=i\in S$ **and** $j\in(V-S)$的最小权边;
6. $T=T\cup\{(i,j)\}$;
7. $S=S\cup\{j\}$;
8. **end**

3. 个体更新公式

针对XSMT构造问题，设计如下适合离散问题的新颖变异算子及交叉算子。

定义3.2 定义以下集合运算。

$A\odot B$：表示求 A 与 B 的差集结果；

$A\oplus B$：表示将 A 中的元素以$\oplus$规则加入 B 中，直至 B 满足运算结束的条件结束；

$A\otimes B$：表示 A 与 B 以$\otimes$规则作交叉操作。

针对本节所求解的问题，A 与 B 都为Steiner树的边集，提出如下$\oplus$规则和$\otimes$规则。

$\oplus$规则即为：

(1) 若 B 为空集，则对 A 采取两点变异。

(2) 若 B 不为空，采用并查集的方法，将 A 中元素作为待选边加入B 中，若 A 中可加入 B 的边，加完后 B 仍不是一棵不合法的Steiner树，则随机连接未连接的点构成新边并初始化其连接方式，将新边加入 B 中直至 B 为一棵合法的Steiner树结束。

$\otimes$规则即为：

起始边集为 A 和B 的公共边，候选边集为剩余的边，采用并查集的方法，不断地从候选边集中选取边加入起始边集，直到起始边集构成一棵合法的Steiner树，若候选边集为空依旧无法构成一棵合法的Steiner树，则随机连接未连接的点构成新边并初始化其连接方式，将新边加入起始边集中直到构成一棵合法的Steiner树。

1) 变异算子

结合定义3.2，给出如下变异算子：

$$V_i(g)=X_{p1}(g)\oplus(X_{p2}(g)\odot X_{p3}(g)) \tag{3.19}$$

结合式(3.19)的变异策略，对变异操作会出现的两种情况进行如下讨论。

(1) 如图 3.7 所示，将随机选出的 3 个个体 p_1、p_2、p_3 代入式(3.19)后，$p_2\odot p_3$ 的结果为空集，则随机删除 p_1 个体的一条边 m_1，则 p_1 变为两棵子树，结合并查集的方法，从两棵子树各随机选取一个引脚点连接并随机初始化该边的连接方式，则得到变异后的新个体。

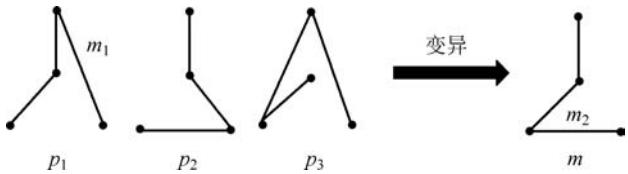


图 3.7　变异算子 1

(2) 如图 3.8 所示，根据式(3.19)，$p_2\odot p_3$ 的结果不为空集，结合并查集策略，将 $p_2\odot p_3$ 的结果作为构建树的起始边集，p_1 个体作为待选边集，不断从待选边集选择符合的边加入起始边集中，若待选边集所有元素都被选择完还未构成一棵合法的 Steiner 树，则随机选择尚未连接的点和已连接的点连接，并初始化走线方式直至起始边集构成一棵合法的 Steiner 树，将这棵构建完的 Steiner 树作为变异产生的个体。

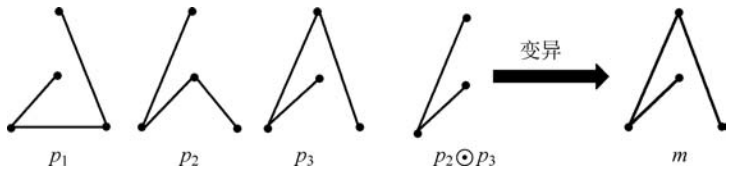


图 3.8　变异算子 2

2) 交叉算子

结合定义 3.2，给出如下交叉策略：

$$\mathrm{XM}_i(g)=V_i(g)\otimes X_i(g)\ \ \mathrm{rand}(0,1)\leqslant \mathrm{cr} \tag{3.20}$$

如图 3.9 所示，根据式(3.20)，对变异后的个体 m 和当前个体 i 进行交叉操作，将 m 与 i 的公共边作为起始边集，则剩余的边作为候选边集。直到构成一棵合法的 Steiner 树，交叉过程将不断从候选边集中选取边加入起始边集。若候选边集为空时还未获得合法的 Steiner 树，则算法将随机连接未连接的点形成新的边，并为该边定义布线方式。这样得到的 Steiner 树作为交叉后产生的交叉个体，代码实现与变异操作类似，此处不再赘述。

3) 选择算子

差分进化的选择操作根据式(3.17)，采用贪心策略，比较交叉后的个体 xm 与当前个体 i 的适应度值，选择适应度值最优的个体进入下一代。

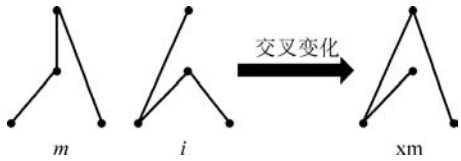


图 3.9 交叉算子

4. 精炼策略

针对算法最终的求解结果，种群中的每个个体可能存在着局部的优化空间，对此提出如下的精炼策略：

(1) 对于种群中的每个个体中每个 p_i 点，计算其度数，度数即为与该点所连的边的数目；

(2) 枚举每个点 p_i 的所有拓扑结构，假定点 p_i 的度为 d，由于每条边有 4 种走线方式，故该点的子结构总数量为 4^d，以子结构的线长为衡量依据，寻找每个点的最小子结构；

(3) 对于策略(2)所求得 n 个子结构，以边的共享程度为衡量依据，对其进行从大到小排序，结合并查集策略，不断地取每个子结构，直至构成一棵合法的树结束。算法的伪代码见算法 3.3。

算法 3.3 精炼策略

输入：迭代结束后的种群

```
1. for i=1 to popsize do
2.    for j=1 to vertice do
3.       CalculateDegree(p_i);
4.    end
5.    for j=1 to vertice do
6.       for k=1 to 4^d do
7.          FindBestSubstructure();
8.          GetSharelength();
9.       end
10.   end
11.   sort();
12.   for j=1 to vertice do
13.      tree=make_tree();
14.      if legal(tree) then
15.         break;
16.      end
17.   end
18. end
```

5. 算法相关参数设置

XSMT-DDE 算法的主要参数包括种群大小 NP、迭代次数 evaluations、阈值 threshold、缩放变量 F 和交叉概率 cr。

设 NP=50，evaluations=500，threshold=0.25，对于 F 和 cr，在算法的第一阶段，由于变异过程引进集合的运算规则，故算法第一阶段 $F=1$，第二阶段 F 采用自适应的策略，具体如下：

$$F_i = F_l + (F_u - F_l)\frac{f_m - f_b}{f_w - f_b} \tag{3.21}$$

其中，$F_l=0.1$，$F_u=0.9$，f_b、f_m、f_w 分别为变异过程中选择出的 3 个个体适应度值的从大到小排序后的结果。交叉概率同样采用自适应策略，具体如下：

$$\mathrm{cr}_i = \begin{cases} \mathrm{cr}_l + (\mathrm{cr}_u - \mathrm{cr}_l)\dfrac{f_i - f_{\min}}{f_{\max} - f_{\min}}, & f_i > \bar{f} \\ \mathrm{cr}_l, & \text{否则} \end{cases} \tag{3.22}$$

其中，$\mathrm{cr}_l=0.1$，$\mathrm{cr}_u=0.6$，f_i、$f_{\min}$、$f_{\max}$、$\bar{f}$ 分别为当前个体的适应度值、种群最小适应度值、种群最大适应度值及种群平均适应度值。

6. XSMT-DDE 算法流程

XSMT-DDE 算法流程图如图 3.10 所示。

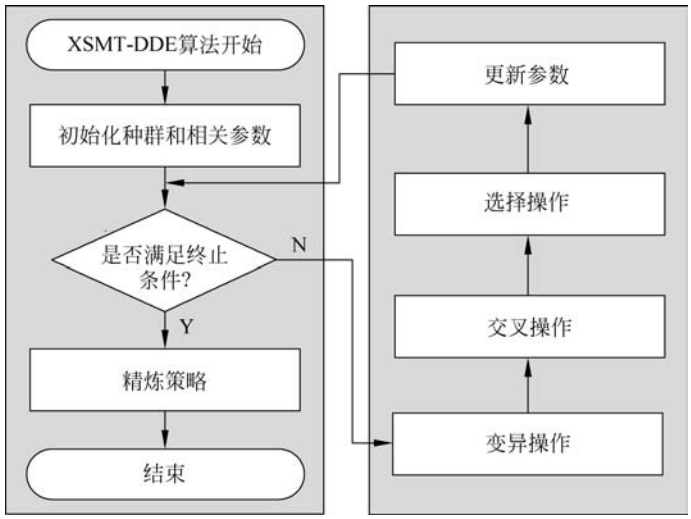


图 3.10　XSMT-DDE 算法流程图

当进行迭代操作时，在第一阶段的每次迭代中，依次以本节重新设计的变异算子（见式(3.19)）、交叉算子（见式(3.20)）进行种群的变异交叉；在第二阶段，由于只变换边的走线方式，故直接采用式(3.11)及式(3.16)对种群个体进行变异交叉，最后通过贪心选择策略进行种群的个体更新优化并根据式(3.18)重新计算种群每

个个体的适应度值。

7. XSMT-DDE 算法伪代码

XSMT-DDE 算法伪代码如算法 3.4 所示，同样分为两个阶段。第一阶段，以引脚点的曼哈顿距离作为边的权值，采用 Prim 算法构造最小生成树。第二阶段分为两个时期：第二阶段前期，采用改进的变异算子及交叉算子对种群进行变异和交叉，再进行贪心选择；第二阶段后期，采用传统的变异算子和交叉算子，再进行贪心选择，直至达到最大迭代次数结束，最后再对得到的种群采取精炼操作。

算法 3.4 XSMT-DDE 算法

输入：XSMT-DDE 参数；阈值：threshold；引脚点集合

输出：最优解向量 $\boldsymbol{\Delta}$

```
1. t←1(initialization);
2. for i=1 to M do
3.       Prim();
4. end
5. while |f(Δ)|≥ε or (t≤T) do
6.       for i=1 to M do
7.          if t≤T×threshold then
8.             v_{i,t}^j=ImproveMutation(x_{i,t}^j);
9.             u_{i,t}^j=ImproveCrossover(x_{i,t}^j,v_{i,t}^j);
10.         else
11.            v_{i,t}^j=TraditionalMutation(x_{i,t}^j);
12.            u_{i,t}^j=TraditionalCrossover(x_{i,t}^j,v_{i,t}^j);
13.      end
14.      if f(u_{i,t})<f(x_{i,t}) then
15.         x_{i,t}←u_{i,t};
16.         if f(x_{i,t})<f(Δ) then
17.            Δ←x_{i,t};
18.         end
19.      else
20.            x_{i,t}←x_{i,t};
21.      end
22.   end
23.   t←t+1;
24. end
25. Refinement();
26. return the best vector Δ;
```

3.3.3　算法仿真与实验结果

1. 开发环境及数据来源

本节算法采用 Visual Studio 开发工具，采用 C/C++语言编写，在 Windows 10 的平台上实现，用于验证算法有效性的测试电路数据来源于文献[3]、文献[4]、文献[8]和文献[14]。为验证本节算法能够较好地解决 XSMT 求解问题，设计 3 组对比实验：第一组实验采用 OB-Library 测试数据，将本节算法与文献[3]和文献[4]的算法进行对比；第二组实验采用随机生成的 10 个测试用实例将本节算法与文献[8]和文献[14]的算法作对比；第三组实验采用 ibm 测试数据集将本节算法与文献[8]的算法进行对比。

2. 实验结果分析

1）验证最小生成树法的有效性

在本节算法中，对种群的初始化采用最小生成树法的方式进行种群的初始化，避免了因随机初始化导致算法解空间过大，从而使算法无法收敛的情况，具体实验结果如表 3.5 所示。其中最后一列表示优化率，具体计算方式如下：

$$\text{优化率}=\frac{\text{RIS}-\text{MST}}{\text{RIS}}\times 100\% \tag{3.23}$$

表 3.5　最小生成树策略带来的优化率

电　路	引　脚　数	随机初始解(RIS)	最小生成树法(MST)	优化率/%
1	8	16 900	16 900	0.00
2	9	18 023	18 023	0.00
3	10	19 397	19 397	0.00
4	20	32 219	32 039	0.56
5	50	55 751	47 908	14.07
6	70	80 640	56 166	30.35
7	100	123 767	68 509	44.65
8	410	1 042 081	141 666	86.41
9	500	1 378 707	154 697	88.78
10	1000	3 388 682	221 739	93.46
均值				35.83

从表 3.5 的最后一列可以看出，随着当前测试电路的引脚点数增多，最小生成树法能够有效地优化由于随机初始化种群导致算法解空间过大而无法收敛的情形。最小生成树法在解决此问题上取得了平均 35.83%的优化率。

2）验证精炼策略的有效性

在本节算法中，对 XSMT-DDE 迭代结束后最终的种群结果采取了精炼的策略，对种群中的每个个体进行精炼，优化布线树的局部结构，实验结果如表 3.6 所示，由实验结果可知，在加入精炼策略后，能够对算法最终结果起到优化作用，取得

了平均0.55%的优化率。

表3.6 精炼策略带来的优化率

电　路	引 脚 数	XSMT-DDE	精　炼	优化率/%
1	8	16 900	16 900	0.00
2	9	18 023	18 023	0.00
3	10	19 397	19 397	0.00
4	20	32 039	32 021	0.06
5	50	47 908	47 772	0.28
6	70	56 166	56 120	0.08
7	100	68 509	68 328	0.26
8	410	141 666	139 993	1.18
9	500	154 697	152 456	1.45
10	1000	221 739	216 958	2.16
均值				0.55

3）XSMT-DDE实验对比

第一组实验采用OB-Library测试数据，将本节算法与文献[3]提出的RSMT算法进行对比，具体实验结果如表3.7所示，由实验结果可知，本节算法所构造的XSMT长度较文献[3]取得了平均9.77%的优化率。第二组实验采用GEO测试数据，将本节算法与文献[3]提出的RSMT算法以及文献[4]提出的OSMT算法进行对比，实验结果如表3.8所示。由实验结果可知，XSMT-DDE在解决Steiner最小树的构造问题中，与传统策略曼哈顿结构RSMT和非曼哈顿结构OSMT对比，分别取得了10.23%和1.05%的平均优化率。第三组实验将本节算法与文献[14]的SAT算法和文献[8]的KNN算法进行对比，具体实验结果如表3.9所示。由实验结果可知，本节算法相较于SAT和KNN算法分别取得了9.86%和8.57%的优化率。

表3.7 XSMT-DDE与RSMT的线长对比

编　号	RSMT	XSMT-DDE	优化率/%
1	1.87	1.7474	6.56
2	1.68	1.5664	6.76
3	2.36	2.1855	7.39
4	2.54	2.2382	11.88
5	2.29	2.1385	6.61
6	2.48	2.3285	6.11
7	2.54	2.3475	7.58
8	2.42	2.2685	6.26
9	1.72	1.6285	5.32
10	1.85	1.6974	8.25

续表

编　　号	RSMT	XSMT-DDE	优化率/%
11	1.44	1.3346	7.32
12	1.80	1.6828	6.51
13	1.50	1.3243	11.72
14	2.60	2.3657	9.01
15	1.48	1.2895	12.87
16	1.60	1.2485	21.97
17	2.01	1.6920	15.82
18	4.06	4.0328	0.67
19	1.93	1.7970	6.89
20	1.12	1.0685	4.60
21	2.16	1.9188	11.17
22	0.63	0.5363	14.88
23	0.65	0.5277	18.82
24	0.30	0.2680	10.67
25	0.23	0.2097	8.82
26	0.15	0.1290	14.00
27	1.33	1.2070	9.25
28	0.28	0.2156	23.01
29	2.00	1.4728	26.36
30	1.10	1.0743	2.34
31	2.66	2.4722	7.06
32	3.30	3.0188	8.52
33	2.69	2.2992	14.53
34	2.54	2.2317	12.14
35	1.57	1.4033	10.62
36	0.90	0.9000	0.00
37	0.90	0.8070	10.34
38	1.66	1.4808	10.79
39	1.66	1.4808	10.79
40	1.62	1.4940	7.78
41	2.24	2.0634	7.89
42	1.53	1.3750	10.13
43	2.68	2.4695	7.85
44	2.61	2.2909	12.22
45	2.26	2.0505	9.27
46	1.50	1.5000	0.00
均值			9.77

以上 3 组实验结果表明，本节算法在 XSMT 的构造问题中，能够较好地解决

线长优化的问题，从而构造一棵质量较佳的 XSMT。

表 3.8 XSMT-DDE 与 RSMT、OSMT 对比

电　路	引脚数	RSMT	OSMT	XSMT-DDE	优化率/%	
					XSMT-DDE：RSMT	XSMT-DDE：OSMT
1	8	17 931	17 040	16 900	5.75	0.82
2	9	20 503	18 163	18 023	12.10	0.77
3	10	21 910	19 818	19 397	11.47	2.12
4	20	35 723	32 199	32 021	10.36	0.55
5	50	53 383	47 960	47 772	10.51	0.39
6	70	61 987	55 980	56 120	9.46	−0.25
7	100	76 016	68 743	68 328	10.11	0.60
8	410	156 520	142 880	139 993	10.56	2.02
9	500	170 273	154 290	152 456	10.46	1.19
10	1000	245 201	222 050	216 958	11.52	2.29
均值					10.23	1.05

表 3.9 XSMT-DDE 与 SAT、KNN 对比

测试用例	线网数目	引脚数目	SAT	KNN	XSMT-DDE	优化率/%	
						XSMT-DDE：SAT	XSMT-DDE：KNN
ibm01	11 507	44 266	61 005	61 071	56 186	7.90	8.00
ibm02	18 429	78 171	172 518	167 359	155 239	10.02	7.24
ibm03	21 621	75 710	150 138	147 982	134 290	10.56	9.25
ibm04	26 263	89 591	164 998	164 828	149 946	9.12	9.03
ibm06	33 354	124 299	289 705	280 998	257 193	11.22	8.47
ibm07	44 394	164 369	368 015	368 015	336 085	8.68	8.68
ibm08	47 944	198 180	431 879	413 201	373 350	13.55	9.64
ibm09	53 039	187 872	418 382	417 543	382 966	8.46	8.28
ibm10	64 227	269 000	588 079	583 102	533 617	9.26	8.49
均值						9.86	8.57

3.3.4 小结

本节算法将改进的差分进化算法应用于 VLSI 总体布线问题。通过改进相关变异交叉策略，使得改进后的差分进化算法能够有效地应用于离散型的问题，从而可以应用到本节所研究的内容，即 XSMT 的求解构造问题。

3.4　基于多策略优化离散差分进化的X结构Steiner最小树算法

3.3节所述的XSMT-DDE算法展现出良好的线长优化能力。本节设计并实现了基于多策略优化离散差分进化的X结构Steiner最小树算法(X-architecture Steiner Minimal Tree algorithm based on Multi-strategy optimization Discrete Differential Evolution,XSMT-MDDE),其中采用了与3.3节相同的模型的编码策略、适应度函数设计策略、初始化种群策略,在DDE算法中加入精英克隆选择策略、多变异策略和自适应学习因子策略优化离散差分进化算法的搜索过程,以提高最终布线树的质量。

3.4.1　算法设计

1. 精英克隆选择策略

性质3.1　该策略使用了两种基于集合的个体自身变异策略,精英个体在短时间内进行变异。对精英个体进行克隆和变异,基于贪婪策略选择最优个体,在较小的时间复杂度内构建出高质量的精英区域。

1) 基于DE算法的精英克隆选择策略

精英克隆选择策略包括4个步骤:选择、克隆、变异和消亡。选取种群中的部分个体作为精英个体,然后对其进行克隆,形成克隆种群。克隆个体采用两种变异策略突变成变异个体。根据消亡策略选择变异个体进入精英区。精英区与种群的规模相当,精英区个体参与随后的差分进化过程。

精英选择和克隆策略可以有效地扩大DDE的搜索范围,提高算法的全局搜索能力,在一定程度上避免陷入局部峰值,防止算法过早收敛。

2) 精英克隆选择策略的算法流程

在解决基于离散差分进化算法的XSMT构造问题中,精英克隆选择策略的算法具体步骤如下。

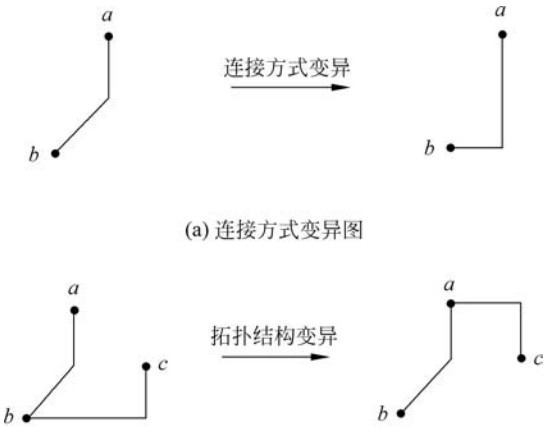
(1) 选择。在差分进化算法进行一次变异、交叉、选择后,选择其中适应度值排名前n的个体组成精英种群,$n=k\times N$。设置精英比例系数k,本算法经实验验证选取k为0.2时能够获得最好的效果。

(2) 克隆。对临时种群的个体进行扩增,扩增数量N_i依据式(3.24)进行计算,生成克隆群体C。

$$N_i=\mathrm{round}\left(\frac{N}{i}\right) \tag{3.24}$$

其中，N 为精英种群大小，round()为取整函数。

(3) 变异。对于克隆群体的个体逐个进行变异，变异策略采用连接方式变异或拓扑结构变异，两种变异方式如图 3.11 所示，每个个体随机选择一种方式变异。对于采用连接方式变异的个体随机选取 e 条边，e 的大小根据边的数量决定，如式(3.25)所示，对选中的边更改连接方式。对于采用拓扑结构变异的个体，从个体中随机选择一条边断开，从而形成两个集合，再从两个集合中各自随机选择一个点进行连接，变异过程采用并查集的思想以确保得到合法树。变异后形成变异种群 M。



(a) 连接方式变异图

(b) 拓扑结构变异图

图 3.11 两种变异方式

$$e = \max\left\{1, \text{round}\left(t\left(\frac{n-1}{10}\right)\right)\right\} \tag{3.25}$$

其中，n 为引脚数量，t 为比例系数。

(4) 消亡。开辟一个规模与种群大小相同的精英区，在集合 M 中选择适应度值最优的个体 m_{best}，若 fitness(m_{best})优于 fitness(g_{best})，则将个体 m_{best} 加入精英区中，集合 M 中其余个体全部消亡；否则，变异集合 M 中的所有个体全部消亡。若精英区已满，则选择适应度值最劣的个体淘汰，再加入新个体。算法流程的伪代码如算法 3.5 所示。

2. 多变异策略

性质 3.2 本策略提出的 3 种新的变异策略引入了集合运算的思想。在合理计算时间的前提下，通过调整当前个体的边集和其他个体的边集，改变 XSMT 中的一些子结构，寻找更好的子结构组合。

算法 3.5　精英克隆选择策略算法

输入 当前种群个体集合 P

输出 精英克隆选择得到的个体

```
1. p_i = get(p);//获取精英个体
2. N = k × NP;
3. for i = 1 to N do
4.     N_i = round(N/i);
5.     M = ∅; // M 为变异种群集合
6.     for j = 1 to N_i do
8.         m_j = Mutation(p_i);//将 p_i 变异
9.         M ∪ {m_j};
10.    end for
11.    m_best = get(M); //获取适应度值最优的变异个体
12.    if Fitness(m_best) < Fitness(g_best) then
13.        Q ∪ {m_best};
14.    end if
15. end for
```

1）多变异策略概述

在差分进化算法中，常用的变异策略有 6 种，每种策略的表示方式为 DE/***a***/*b*”，其中 DE 代表差分进化，***a*** 代表基向量，*b* 代表差分向量的数量，6 种策略如下公式所示。

（1）DE/rand/1：

$$V_i^g = X_{r1}^g + F(X_{r2}^g - X_{r3}^g) \tag{3.26}$$

（2）DE/rand/2：

$$V_i^g = X_{r1}^g + F(X_{r2}^g - X_{r3}^g) + F(X_{r4}^g - X_{r5}^g) \tag{3.27}$$

（3）DE/best/1：

$$V_i^g = X_{\text{best}}^g + F(X_{r1}^g - X_{r2}^g) \tag{3.28}$$

（4）DE/best/2：

$$V_i^g = X_{\text{best}}^g + F(X_{r1}^g - X_{r2}^g) + F(X_{r3}^g - X_{r4}^g) \tag{3.29}$$

（5）DE/current-to-best/1：

$$V_i^g = X_i^g + F(X_{\text{best}}^g - X_i^g) \tag{3.30}$$

（6）DE/rand-to-best/1：

$$V_i^g = X_{r0}^g + F(X_{\text{best}}^g - X_{r0}^g) + F(X_{r1}^g - X_{r2}^g) \tag{3.31}$$

其中，$r_0 \neq r_1 \neq r_2 \neq r_3 \neq r_4 \neq r_5 \neq i \in \{1,2,\cdots,\text{NP}\}$，$X_r^g$ 表示第 g 次迭代种群中的随机个体，X_{best}^g 表示第 g 次迭代时的全局最优解，学习因子 $F \in [0,2]$。

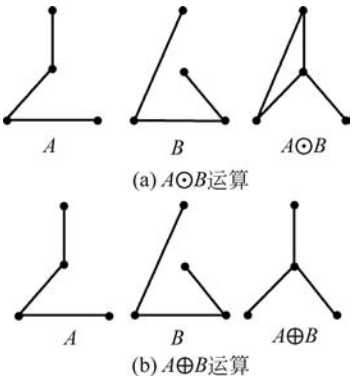
2）基于 XSMT 构建问题的多变异策略

这里使用了两种基于集合运算的符号，对集合运算的定义如下。

A 为个体 X_1 的边集，B 为个体 X_2 的边集，全集为 $A \cup B$，定义以下两种运算符。

定义 3.3 $A \odot B$ 表示求 A 与 B 的对称差集，即为 $(A \cup B) - (A \cap B)$，如图 3.12(a)所示。

定义 3.4 $A \oplus B$ 首先计算集合 $C = A - B$，再将集合 B 中边加入集合 C 中，直至集合 C 的边集能够构成一棵合法的树为止，如图 3.12(b)所示。



(a) $A \odot B$运算

(b) $A \oplus B$运算

图 3.12 两种新型运算符的运算过程

给出以上两种运算的定义后，接下来讨论基于 XSMT 问题提出的 3 种新型变异策略，由此构成变异策略池。

变异策略 1：DE/current-to-best(gbest+pbest)/1。在该变异策略中，第一部分的基向量为当前个体，差分向量由当前个体与当前个体的局部最优个体运算产生，经式(3.32)运算得到个体 $\boldsymbol{T}$。第二部分的基向量为个体 $\boldsymbol{T}$，差分向量由个体 $\boldsymbol{T}$ 与全局最优个体运算产生，经式(3.33)运算得到目标变异个体 $\boldsymbol{V}_i^g$，变异公式如下：

$$\boldsymbol{T} = \boldsymbol{X}_i^g \oplus \boldsymbol{F} * (\boldsymbol{X}_{\text{pbest}}^g \odot \boldsymbol{X}_i^g) \tag{3.32}$$

$$\boldsymbol{V}_i^g = \boldsymbol{T} \oplus \boldsymbol{F} * (\boldsymbol{X}_{\text{gbest}}^g \odot \boldsymbol{T}) \tag{3.33}$$

变异策略 1 保留个体 $\boldsymbol{X}_i^g$ 与最优个体的局部共同边，剩余的边以相同的概率从个体 $\boldsymbol{X}_i^g$ 和最优个体中获取，使得当前种群个体得到最优个体的部分拓扑结构。

变异策略 2：DE/rand-to-best(gbest+pbest)/1。其中，第一步的基向量选取为个体 $\boldsymbol{X}_i^g$，差分向量由种群中的随机个体与其对应的局部最优个体运算产生，经式(3.34)运算得到个体 $\boldsymbol{T}$。第二步的基向量为个体 $\boldsymbol{T}$，差分向量由另一随机个体(两次选取的随机个体不同)与全局最优个体运算产生，经式(3.35)运算得到目标个体 $\boldsymbol{V}_i^g$，变异公式如下：

$$\boldsymbol{T} = \boldsymbol{X}_i^g \oplus \boldsymbol{F} * (\boldsymbol{X}_{\text{pbest}}^g \odot \boldsymbol{X}_{r1}^g) \tag{3.34}$$

$$\boldsymbol{V}_i^g = \boldsymbol{T} \oplus \boldsymbol{F} * (\boldsymbol{X}_{\text{gbest}}^g \odot \boldsymbol{X}_{r2}^g) \tag{3.35}$$

其中，$r_1 \neq r_2 \neq i$，变异策略 2 有较大的概率获取不同于当前个体与最优个体的结构，使得变异范围不局限在最优个体的边集。

变异策略 3：DE/current-to-rand/1。其中，个体 $\boldsymbol{X}_i^g$ 依旧为基向量，而差分向量的对象则分别是个体 $\boldsymbol{X}_i^g$ 与种群中的随机个体 $\boldsymbol{X}_r^g$，目标个体 $\boldsymbol{V}_i^g$ 通过式(3.36)生成。

$$\boldsymbol{V}_i^g = \boldsymbol{X}_i^g \oplus \boldsymbol{F} * (\boldsymbol{X}_i^g \odot \boldsymbol{X}_r^g) \tag{3.36}$$

变异策略 3 的变异范围广，在前期的搜索过程中能够扩大搜索空间。

在多变异策略中，设置阈值 threshold 将整个迭代过程分为两个阶段，在前期采用多变异策略，每种策略从策略池中等概率选取，根据变异式得到变异个体。在迭代后期种群具有较高的拓扑结构多样性，为了搜索每种结构下更优的适应度值，更改变异方式，采用自身变异策略，在不改变个体拓扑结构的原则上，变异部分边的连接方式。多变异策略伪代码如算法 3.6 所示。

算法 3.6　多变异策略算法

输入 当前种群 P，迭代次数 evaluations，阈值 threshold
输出 变异后种群 V

```
Begin:
1. for i=1 to N do
2.     if i≤evaluation * threshold then
3.         s=(rand()/(max size+1)) * 3+1;
4.     else
5.         s=0;
6.     end if
7.     if s=1 then
8.         V∪Mutation1(p_i); //采用变异策略 1
9.     else if s=2 then
10.        V∪Mutation2(p_i); //采用变异策略 2
11.    else if s=3 then
12.        V∪Mutation3(p_i); //采用变异策略 3
13.    else
14.        V∪Mutation4(p_i); //处于后期，采用自身变异
15.    end if
16. end for
End
```

3. 自适应学习因子策略

性质 3.3　学习因子是决定 DDE 算法性能的关键参数，对算法的开发和探索

能力有着决定性的影响。本节首次提出了一种基于集合运算的自适应学习因子，有效地平衡了 XSMT-MDDE 算法的搜索和开发能力。

1）自适应学习因子的定义

对于变异算子 $V_i^g = X_i^g \oplus F_i * (X_{\text{best}}^g \odot X_i^g)$，$F_i \in [0,2]$，现有如下定义来定义运算符 $*$：

定义 3.5 若 $F_i < 1$，从边集合 $X_{\text{best}}^g \odot X_i^g$ 中随机淘汰 n 条边 $\{e_1, e_2, \cdots, e_n\}$，其中 $e_i \in X_{\text{best}}^g$ 且 $e_i \notin X_i^g$，n 的值由式(3.37)计算得到。

$$n = \text{round}((1 - F_i) \times |X_{\text{best}}^g|) \tag{3.37}$$

其中，$|X_{\text{best}}^g|$ 为集合 X_{best}^g 的大小。

定义 3.6 若 $F_i > 1$，从边集合 $X_{\text{best}}^g \odot X_i^g$ 中随机淘汰 n 条边 $\{e_1, e_2, \cdots, e_n\}$，其中 $e_i \in X_i^g$ 且 $e_i \notin X_{\text{best}}^g$，$n$ 的值由式(3.38)计算得到。

$$n = \text{round}((F_i - 1) \times |X_i^g|) \tag{3.38}$$

定义 3.7 若 $F_i = 1$，则不改动任何边集合的保留比例，按原先的运算策略进行。

2）参数 F 更新过程

每个个体 X_i^g 设置一个自适应参数 F_i，初始化均为 1，在迭代过程中，每次选择操作结束后对参数 F_i 进行更新。自适应参数 F 更新过程的伪代码如算法 3.7 所示。

算法 3.7 自适应学习因子更新

输入 种群 P，种群大小 NP，迭代次数 evaluations

输出 自适应参数 F

```
Begin:
1. F_i ← 1; //初始化
2. r = max{10, 0.001 × f_best};
3. for j = 1 to evaluations do
4.     for i = 1 to NP do
5.         Δ = f_i − f_best;
6.         if Δ < r then
7.             F_i = F_i − 0.05;
8.         else
9.             F_i = F_i + 0.05;
10.        end if
11.    end for
12. end for
End
```

(1) 设置参考常数 r，本算法中常数 r 的选取方式如式(3.39)所示。

$$r = k \times f_{\text{best}} \tag{3.39}$$

其中，k 取 0.001，f_{best} 为 X_{best}^{g} 的适应度值；

(2) 计算当前个体 X_i^g 的适应度值 f_i 与 X_{best}^{g} 的适应度值 f_{best} 的差值 Δ；

(3) 更新 F_i，更新公式如下：

$$F_i = \begin{cases} F_i + 0.05, & \Delta > r \\ F_i - 0.05, & \Delta \leqslant r \end{cases} \tag{3.40}$$

适应度值 f_i 足够接近 X_{best}^{g} 的适应度值 f_{best} 时，此时减小 F_i 的值，更大程度地保留原有结构；反之增大 F_i 值，以扩大变异范围。

3.4.2　算法仿真与实验结果

1. 开发环境与数据来源

本节算法的开发工具为 Visual Studio 2019，使用 C/C++语言，操作系统为 Windows 10。为验证本节所述算法能较好地解决 XSMT 问题，设计如下 3 组对比实验，其中前两组的测试数据来源为 GEO，第三组的测试数据来源为 IBM：第一组实验为加入多策略优化前后的结果进行比较，验证多策略优化后的 DE 算法在解决 XSMT 问题中的有效性；第二组实验将本节算法的实验结果与 DDE、ABC 和 GA 算法进行对比；第三组实验采用 IBM 数据集，实验结果与 SAT 和 KNN 进行对比。在本节所述实验的所有算法初始种群大小均为 50，迭代次数均为 500。

2. 实验结果分析

1) 多策略优化的有效性验证

第一组实验：为了验证多策略优化 DE 算法在解决 XSMT 问题中的有效性，将加入多策略优化前后的实验结果进行对比，实验对比结果分别如表 3.10 和表 3.11 所示，其中表 3.10 为平均线长优化结果，表 3.11 为标准差优化结果。可以看出，加入多策略优化后能够取得 2.40%的平均线长优化率及 95.65%的标准差优化率。

表 3.10　多策略优化的平均线长优化结果

电　路	引脚数	XSMT-DDE	XSMT-MDDE	平均线长优化率/%
1	8	16 956	16 900	0.33
2	9	18 083	18 023	0.33
3	10	19 430	19 397	0.17
4	15	25 728	25 624	0.40
5	20	32 434	32 091	1.06
6	50	49 103	48 090	2.06
7	70	57 386	56 105	2.23
8	100	70 407	68 457	2.77
9	400	145 183	138 512	4.59

续表

电　路	引脚数	XSMT-DDE	XSMT-MDDE	平均线长优化率/%
10	410	146 680	140 359	4.31
11	500	160 031	152 649	4.61
12	1000	232 057	217 060	5.90
均值				2.40

表 3.11　多策略优化的标准差优化结果

电　路	引脚数	XSMT-DDE	XSMT-MDDE	标准差优化率/%
1	8	56	0	100.00
2	9	58	0	100.00
3	10	42	0	100.00
4	15	198	8	95.96
5	20	343	22	93.59
6	50	1036	119	88.51
7	70	1082	136	87.43
8	100	1905	187	90.18
9	400	3221	57	98.23
10	410	3222	56	98.26
11	500	3193	50	98.43
12	1000	3977	113	97.16
均值				95.65

2）XSMT-MDDE 实验对比

第二组实验：为了测试多策略优化的离散差分进化算法（MDDE）在解决 XSMT 问题上的能力，将本节算法与传统离散差分进化算法（DDE）、人工蜂群算法（ABC）和遗传算法（GA）进行对比，表 3.12～表 3.14 分别展示了 4 种算法对于平均线长、最优线长和标准差的优化对比。与 DDE、ABC 和 GA 相比，本算法分别得到 2.40%、1.74%和 1.77%的平均线长优化率，1.26%、1.55%和 1.77%的最优线长优化率，95.65%、33.52%和 28.61%的标准差优化率。

表 3.12　各算法平均线长优化对比

电　路	引脚数	平均线长				平均线长优化率/%		
		DDE	ABC	GA	MDDE	DDE	ABC	GA
1	8	16 956	16 918	16 918	16 900	0.33	0.00	0.00
2	9	18 083	18 041	18 041	18 023	0.33	0.10	0.10
3	10	19 430	19 696	19 696	19 397	0.17	1.52	1.52
4	15	25 728	25 919	25 989	25 624	0.40	1.14	1.40
5	20	32 434	32 488	32 767	32 091	1.06	1.22	2.06

续表

电　路	引脚数	平均线长				平均线长优化率/%		
		DDE	ABC	GA	MDDE	DDE	ABC	GA
6	50	49 103	48 940	48 997	48 090	2.06	1.74	1.85
7	70	57 386	57 620	57 476	56 105	2.23	2.63	2.39
8	100	70 407	70 532	70 277	68 457	2.77	2.94	2.59
9	400	145 183	141 835	141 823	138 512	4.59	2.40	2.40
10	410	146 680	143 642	143 445	140 359	4.31	2.29	2.15
11	500	160 031	156 457	156 394	152 649	4.61	2.43	2.39
12	1000	232 057	222 547	222 487	217 060	5.90	2.47	2.44
均值						2.40	1.74	1.77

表 3.13　各算法最优线长优化对比

电　路	引脚数	最优线长				最优线长优化率/%		
		DDE	ABC	GA	MDDE	DDE	ABC	GA
1	8	16 918	16 918	16 918	16 900	0.11	0.11	0.11
2	9	18 041	18 041	18 041	18 023	0.10	0.10	0.10
3	10	19 415	19 696	19 696	19 397	0.09	1.52	1.52
4	15	25 627	25 627	25 897	25 605	0.09	0.09	1.13
5	20	32 209	32 344	32 767	32 091	0.37	0.78	2.06
6	50	47 987	48 637	48 783	47 975	0.03	1.36	1.66
7	70	56 408	57 227	57 445	55 919	0.87	2.29	2.66
8	100	68 829	70 382	70 092	68 039	1.15	3.33	2.93
9	400	141 967	141 490	141 467	138 382	2.53	2.20	2.18
10	410	144 033	143 310	143 282	140 179	2.68	2.18	2.17
11	500	156 950	156 034	156 110	152 591	2.78	2.21	2.25
12	1000	226 654	222 262	222 285	216 824	4.34	2.45	2.46
均值						1.26	1.55	1.77

表 3.14　各算法标准差优化对比

电　路	引脚数	标　准　差				标准差优化率/%		
		DDE	ABC	GA	MDDE	DDE	ABC	GA
1	8	56	0	0	0	100.00	—	—
2	9	58	0	0	0	100.00	—	—
3	10	42	0	0	0	100.00	—	—
4	15	198	148	46	8	95.96	94.59	82.61
5	20	343	118	45	22	93.59	81.36	51.11
6	50	1036	242	133	119	88.51	50.83	10.53
7	70	1082	195	140	136	87.43	30.26	2.86

续表

电　路	引脚数	标　准　差				标准差优化率/%		
		DDE	ABC	GA	MDDE	DDE	ABC	GA
8	100	1905	69	112	187	90.18	−171.01	−66.96
9	400	3221	200	170	57	98.23	71.50	66.47
10	410	3222	146	122	56	98.26	61.64	54.10
11	500	3193	160	133	50	98.43	68.75	62.41
12	1000	3977	131	107	113	97.16	13.74	−5.61
均值						95.65	33.52	28.61

第三组实验：为了验证本节算法在构造多线网 XSMT 的能力，本实验采用 IBM 数据集，与文献[14]的 SAT 算法和文献[8]的 KNN 算法进行对比，实验结果如表 3.15 所示，分别得到 10.05%和 8.86%的线长优化率。

表 3.15　IBM 数据集中各算法构造 XSMT 的效果对比

电　路	线网数	引脚数	线长			平均线长优化率/%	
			SAT	KNN	MDDE	SAT	KNN
ibm01	11 507	44 266	61 005	61 071	56 080	8.07	8.17
ibm02	18 429	78 171	172 518	167 359	154 868	10.23	7.46
ibm03	21 621	75 710	150 138	147 982	133 999	10.75	9.45
ibm04	26 263	89 591	164 998	164 828	149 727	9.26	9.16
ibm06	33 354	124 299	289 705	280 998	256 674	11.40	8.66
ibm07	44 394	164 369	368 015	368 015	335 556	8.82	8.82
ibm08	47 944	198 180	431 879	413 201	371 948	13.88	9.98
ibm09	53 039	187 872	418 382	417 543	382 282	8.63	8.44
ibm10	64 227	269 000	588 079	589 102	532 644	9.43	9.58
均值						10.05	8.86

3.4.3　小结

本节设计了 3 种优化策略用于改进离散差分进化算法，以求解 XSMT 问题。通过精英克隆选择策略扩大搜索空间，多变异策略扩大变异范围，更好地平衡全局与局部的搜索能力，自适应学习因子策略动态调整保留当前个体边集与全局最优个体边集之间的比例。本节提出的基于多策略优化离散差分进化的 XSMT 算法，以线长优化和标准差优化为评价指标，与其他相关算法相比均取得了不错的优化结果。同时，XSMT-MDDE 在引脚数量越大的线网中所体现出的优化能力越强，因此应用在 VLSI 布线中能够得到优秀的布线结果。

3.5 基于文化基因的 X 结构 Steiner 最小树算法

文化基因算法(Memetic Algorithm,MA)于 1989 年由 Pablo Moscato 首次提出,并逐渐引起各国研究人员的兴趣和关注。在一般遗传算法中,操作的对象是局部搜索得到的优秀种群,避免迂回的无用搜索,以更好地搜索全局最优。而文化基因则是结合全局搜索和局部搜索,在某些问题领域提高了算法的搜索效率,适用于求解 NP 难问题。近年来,文化基因算法已被成功地应用到了自动化控制、工程优化、机器学习等领域并得到了令人满意的结果。

本节主要介绍一种基于文化基因算法的 X 结构 Steiner 最小树构建算法,即 MA_XMST 算法。首先,使用 Prim 算法预处理种群,产生初始种群,解决了因引脚数过多而无法收敛的情况。然后,通过修改文化基因的个体更新公式并设计了一种新的文化基因中的操作,使其能应用于离散 X 结构 Steiner 树,用于解决 VLSI 中的布线问题。最后,调整文化基因中的权重因子,能在一定迭代次数下,快速得到一个最优或者较优的拓扑结果。

3.5.1 MA_XMST 算法

1. MA_XMST 算法描述

MA_XMST 算法是基于文化基因算法,通过使用 Prim 算法进行预处理,产生初始种群,并修改文化基因算法中的个体编码方式,同时修改相关操作来实现 X 结构 Steiner 最小树构建这个问题的解决。个体的更新公式如式(3.41)所示。

$$P_i^t = S(M_2(M_1(C(P_i^{t-1}, w), m_1), m_2)) \tag{3.41}$$

其中,P_i^t 表示第 t 次迭代以后种群中第 i 个个体。C 表示基于最大并集的交叉操作,M_1、M_2 表示双重变异操作,S 表示局部搜索操作。m_1、m_2、w 是权重因子。

MA_XMST 算法伪代码如算法 3.8 所示。

算法 3.8 MA_XMST

输入:所有的线网

Step1. 选取一个线网

Step2. 用 Prim 算法产生种群

Step3. 遍历种群中每个个体并计算其适应度函数值

Step4. 算法终止条件内:

1 基于最大并集的交叉操作

2 双重变异操作

3 局部搜索

4 计算适应度函数值

5. 选择下一次迭代的个体
6. 迭代次数加一
Step5 算法终止
Step6 输出结果

2. 初始化种群

初始化种群一般使用随机的方法，也可以根据已有信息人为选取比较优质的种群个体。在 MA_XMST 算法中，使用 Prim 算法初始化种群。这种预处理策略，能加速收敛，并能辅助算法优化拓扑结构。其中，Prim 算法的伪代码如 3.3.2 节中的算法 3.2 所示。

3. 适应度函数

在 X 结构 Steiner 最小树问题中，优化 Steiner 树的拓扑结构，使其总线长最短是首要目标。因此，在 MA_XMST 算法中，个体的目标函数值是所有边的长度之和，计算公式如 3.2.1 节式(3.1)，个体的适应度函数是一个非零常数与目标函数值之和的倒数，其计算公式如式(3.42)所示。

$$F(N_i)=\frac{1}{C+L(T_X)} \tag{3.42}$$

其中，C 是一个非零常数，代表的是线网中的单位长度。这样做的目的是防止式(3.42)中分母为 0 的情况出现，即引脚重合。

4. 混合优化的遗传算子

遗传操作将局部的启发式搜索与全局寻优搜索结合得到混合搜索机制：首先选出适应度值较优的个体且产生一些交叉作用后的新个体。可能位于新区域的新个体，在下一代局部搜索中被优秀个体取代，然后再进行进一步的全局进化。这种混合优化的策略在进化效率上显然要比单纯的个体间搜索高得多。本节使用是混合优化遗传算子，分别使用基于最大并集的交叉操作和双重变异操作。

1) 基于最大并集的交叉操作

为了加快算法的收敛速度，MA_XMST 算法采用最大并集的策略来实现交叉操作，选择当前最优的个体和迭代过程中最优的个体进行交叉。初始阶段是得到两个个体的交集，第二个阶段是将剩下未连接到 Steiner 树的引脚，随机选择前两者中的一种连接方式连接起来，最终得到交叉操作的结果。例如，有一个 5 个引脚的线网，图 3.13(a)是个体 A，图 3.13(b)是个体 B，基于最大并集的交叉操作后得到的结果如图 3.13(c)、图 3.13(d)所示。

2) 双重变异操作

为了降低造成局部收敛的可能性，避免种群选择的局限性。本算法采用双重变异操作，即两种变异方式来拓展文化基因算法的搜索：第一种是拐点的变异；第二种是引脚选取方式的变异。

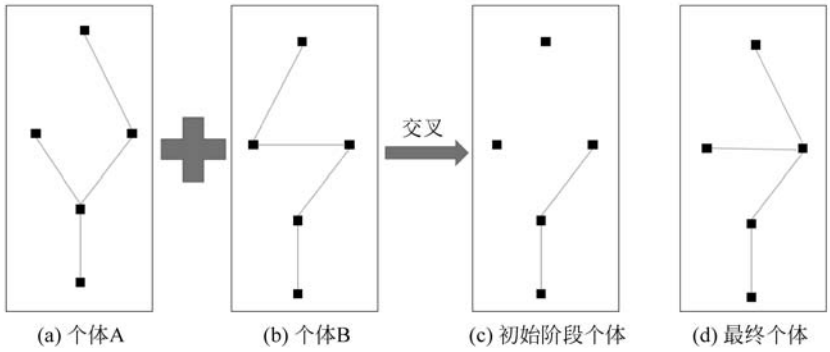


图 3.13　基于最大并集的交叉操作

拐点的变异是改变边的选择方式，从 4 种选择方式中随机选择一种边选择方式。图 3.14 是拐点变异操作，线网 1 的边选择方式从选择 2(见图 3.14(a))变异成为选择 0(见图 3.14(b))，从而得到了更优的个体。

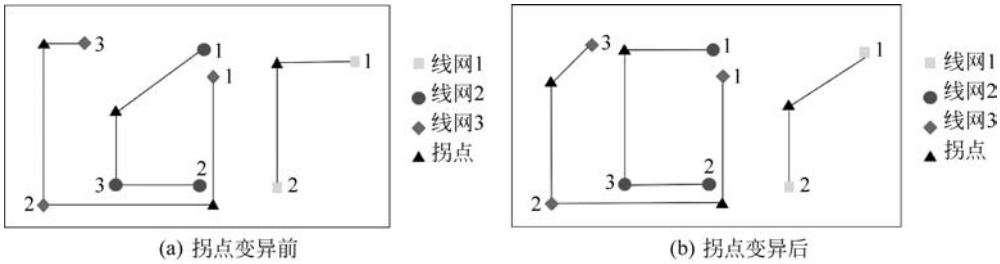


图 3.14　拐点变异操作

引脚的变异是多点变异，改变了 X 结构 Steiner 树的拓扑结构。通过不同的拓扑结构来寻优，让 MA_XMST 算法有较强的搜索能力，使其得到更好的结果。例如，将线网 1 上的引脚 1 和引脚 2 的连接变异成为引脚 1 和引脚 3 的连接，得到了不同的拓扑结构，如图 3.15 所示。

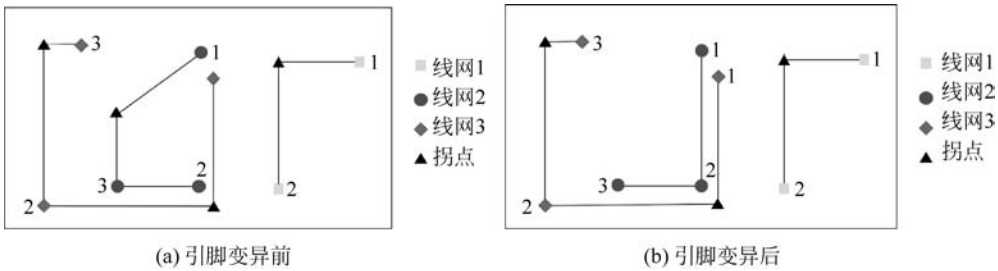


图 3.15　引脚变异操作

5. 局部搜索

局部搜索的目的是得到局部区域内最优秀的个体。若不采用局部搜索算法，则可能导致算法的局部搜索能力变差，无法得到最优或者较优的结果。在 MA_XMST 算法中，局部搜索的领域是个体所有边的 4 种边选择方式，从选择 0 到选择 3。最终得到局部搜索领域中最优的个体，更新种群。局部搜索算法的伪代码如算法 3.9 所示。

算法 3.9 局部搜索算法

输入：个体的边集

```
1. E={e}
2. while E!=NULL do
3.   for i=0 to choice[i]<4 do
4.     NewFitness=calculateFitness(choice[i]);
5.     if NewFitness>MaxFitness do
6.       MaxFitness=NewFitness;
7.       Update(choice[i]);
8.     end
9.   end
10. End
```

6. 选择种群

合并父代和子代个体，计算所有个体的适应度值，按照适应度函数值大小排序，选择与种群规模大小相同的优秀个体组成新种群。

3.5.2 实验仿真与结果分析

1. 开发环境及数据来源

本节算法采用 C++实现，用 Visual Studio 2015 编写与开发，在 Windows 10 操作平台上运行，分别对单个线网和多个线网两种测试电路数据进行实验。为了验证预处理策略有效性的实验数据来自文献[5]，并与未采用预处理策略的实验结果进行对比；为了验证 MA_XMST 算法的优越性，实验数据采用 ISPD98 测试数据集，并与文献[8]提出的 K 近邻算法（K-Nearest Neighbour，KNN）和文献[14]提出的伪布尔 SAT 的算法（A pseudo boolean SAT）进行对比。

2. 实验结果分析

1）预处理策略的有效性证明

为了验证 MA_XMST 算法所使用的预处理策略的有效性，本节算法设置种群规模为 50，最大的迭代次数为 200，验证预处理有效性的 10 组测试用例是单个线网数据集，其引脚数量最少的为 8，最多的是 1000，每个测试用例均运行 10 次的本节算法并取线长的平均值。

表 3.16 给出的是在算法中使用 Prim 算法和未使用 Prim 算法预处理种群在相同的运行环境下，通过这 10 组测试用例，验证预处理策略的有效性。通过使用 Prim 算法预处理种群生成一个最小生成树，避免了因为引脚数过多造成的解空间过大而不能得到一个收敛结果的情况。

实验结果如表 3.16 所示，第三列是未使用预处理策略得到的线网线长，第四列是使用预处理策略得到的线网线长，对于任何一个测试用例，预处理策略都产生了优化效果，并且随着引脚数量的增加，使用 Prim 算法预处理得到的结果优化率显著提高。

表 3.16　验证预处理策略的有效性

测 试 用 例	引　脚　数	未使用预处理	使用预处理	减少率/%
1	8	17 107	16 918	1.10
2	9	19 273	18 040	6.40
3	10	20 955	19 695	6.01
4	20	48 060	32 257	32.88
5	50	127 884	48 580	62.01
6	70	180 752	57 329	68.28
7	100	215 120	70 086	67.42
8	410	926 264	143 318	84.53
9	500	1 244 181	156 331	87.44
10	1000	2 572 420	222 160	91.36
均值				50.74

对于这 10 组数据，使用了预处理方案的结果比未使用预处理的结果平均优化了 50.74%的布线线长。

2）与其他算法的实验对比

为了验证 MA_XMST 算法的优越性，本节算法设置种群规模为 50，最大的迭代次数为 200，验证算法优越性的测试用例是 ISPD98 的 9 组多线网数据集。其中线网数目从 11 507 个增长到 64 227 个，引脚总数从 44 266 个增长到 269 000 个。本节中每个测试用例均运行 10 次的本节算法并取线长的平均值，最后与 KNN 和 SAT 进行比较。

表 3.17 是本节提出的基于文化基因的 X 结构 Steiner 树算法与 KNN 算法和 SAT 算法的实验结果对比。本算法对于每个测试用例在线长方面都有较大的优化，几乎每个测试用例都优化了 10%以上。最终，MA_XMST 算法分别比 SAT 和 KNN 平均减少了 11.52%和 10.24%的布线线长，充分说明基于文化基因的 X 结构 Steiner 树算法在处理现实布线中能有较大的优越性。

表 3.17 与 SAT 和 *K*NN 算法的对比

测试用例	线网数目	引脚数目	SAT	*K*NN	MA_XMST	减少率/(%)	
						SAT	*K*NN
ibm01	11 507	44 266	61 005	61 071	54 823	10.13	10.23
ibm02	18 429	78 171	172 518	167 359	152 604	11.54	8.82
ibm03	21 621	75 710	150 138	147 982	131 848	12.18	10.90
ibm04	26 163	89 591	164 998	164 828	146 277	11.35	11.25
ibm06	33 354	124 299	289 705	280 998	253 548	12.48	9.77
ibm07	44 394	164 369	368 015	368 015	329 647	10.43	10.43
ibm08	47 944	198 180	431 879	413 201	368 694	14.63	10.77
ibm09	53 093	187 872	418 382	417 543	376 609	9.98	9.80
ibm10	64 227	269 000	588 079	583 102	523 730	10.94	10.18
均值						11.52	10.24

3.5.3 小结

本节提出了基于文化基因的 X 结构 Steiner 树算法，并成功应用于 VLSI 的布线问题。首先，在预处理阶段使用 Prim 算法处理种群，克服了因为引脚数量过多造成的无法收敛的问题，得到了将近 50.74%的优化率。其次，根据 X 结构 Steiner 树结构的特点，结合文化基因算法，设计了离散个体更新公式并修改了相应操作，使算法快速收敛。最后，通过实验证明基于文化基因的 X 结构 Steiner 树算法的有效性和优越性，在与最近几年提出解决布线问题的 SAT 和 *K*NN 算法相比，分别取得了 11.52%和 10.24%的优化率。

3.6 线长驱动的 X 结构 Steiner 最小树算法

3.6.1 引言

早期解决 Steiner 最小树问题的方法大多为精确算法和传统启发式算法，面临复杂度极高和极易陷入局部最优解的难题，而群智能(Swarm Intelligence，SI)技术的高效搜索能力和自组织能力，能够更好地完成 Steiner 最小树的构建。尤其是粒子群优化算法，其控制参数少、实现简单和优秀的寻优能力在提高 Steiner 树问题的解的质量方面具有明显优势。然而，由于传统的粒子群优化方法仅通过向种群最优粒子学习来实现粒子的社交过程，使得算法开发强度过大，一旦陷入局部极值，就很难跳出。因此，本节提出了基于社会学习离散 PSO 的线长驱动 X 结构 Steiner 最小树算法(Wirelength-Driven X-architecture Steiner Minimum Tree algorithm based on Social Learning Discrete Particle Swarm Optimization，SLDPSO-WL-XSMT)，使得算法能够平衡好开发和探索的强度，从而更好地解决

WL-XSMT 问题。该算法由两个阶段构成。

(1) 社会学习粒子群搜索阶段。该阶段通过 PSO 技术搜索到一棵令人满意的具有较短线长的 XST。首先,使用混沌惯性权重以加强 PSO 的全局搜索;其次,提出了一个新的社会学习策略,改变粒子在每次迭代中只向个体历史最优粒子(pbest)或种群最优粒子(gbest)位置靠近的单一学习方式,通过维护样例池不断改变粒子在每次迭代中的学习对象,增强了种群进化的多样性;最后,变异和交叉算子被融合进粒子的更新公式中以实现 PSO 的离散化,从而构建一棵理想的 X 结构 Steiner 树。

(2) 线长减少阶段。该阶段设计了一个有效的基于局部拓扑优化的策略以减少 XST 的线长。通过不断调整每个引脚所在的局部最优结构,进一步优化布线树的长度,得到最终的 XSMT。

3.6.2　算法设计

本节提出的 SLDPSO-WL-XSMT 算法首先通过 SLDPSO 搜索找到一棵令人满意的具有较短线长的 XST。在这个阶段,混沌下降变异策略和结合交叉算子的社会学习策略被加入到 PSO 中,用来增强种群进化的多样性,同时能够平衡算法的开发和探索能力。接着再通过线长减少阶段调整 Steiner 树的局部最优结构得到最终 XSMT。该算法的整体架构如图 3.16 所示。

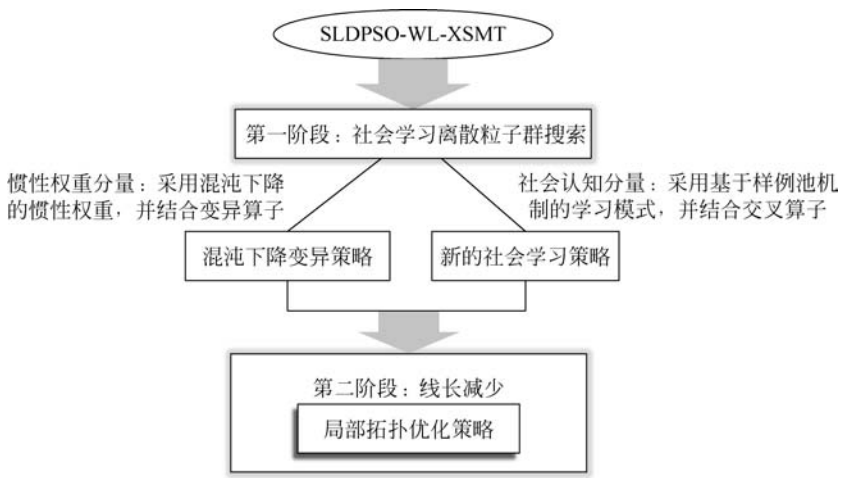


图 3.16　SLDPSO-WL-XSMT 算法的整体架构图

1. SLDPSO 搜索阶段

由于 XST 问题是一个离散问题,SPSO 的粒子更新过程不再适用于解决 XSMT 问题。为此,需要对 XSMT 问题进行编码,设计针对线长优化的适应度函数,同时变异和交叉算子被融合进粒子的更新公式中以完成 PSO 的离散化。本节

基于混沌搜索和新的社会学习模式，设计了针对 WL-XSMT 问题的社会学习离散 PSO 算法，并给出了算法实现的细节，即从混沌下降变异策略，基于样例池机制的社会学习策略、粒子编码、适应度函数、粒子更新公式以及该阶段的具体步骤 5 个方面详细展开。

1）混沌下降变异策略

惯性权重是影响 PSO 收敛精度的重要参数，而常数的惯性权重会不利于 PSO 算法的全局探索。为此，Shi 和 Eberhart 提出了一种有效的参数策略，即线性递减策略。该策略能有效提高解的质量，并被广泛应用在 VLSI 布线的相关问题中。而文献[17]的工作对惯性权重进行了更为深入的研究，在 15 种惯性权重的实验对比中，发现混沌惯性权重能够获得最佳效果。

性质 3.4 混沌搜索是一种具有伪随机性、遍历性和规律性的随机运动，具有不对初始值敏感、计算精度高的特点。

定义 3.8 Logistic 映射 利用 Logistic 映射方程生成一组具有遍历性的随机序列。

$$z_{n+1}=\mu\cdot z_n\cdot(1-z_n),\quad n=0,1,2,\cdots \tag{3.43}$$

其中，$z\in(0,1)$为混沌变量，z 的初始值 $z_0\neq\{0.25,0.5,0.75\}$，否则最终的随机序列会是周期性的。$\mu\in[0,4]$为控制参数，当 $\mu=4$ 时，Logistic 映射将表现出完全的混沌动力学，混沌变量的轨迹在整个搜索空间上分布密集，即其混沌结果分布在区间[0,1]。

SLDPSO-WL-XSMT 采用混沌下降的惯性权重，使得 PSO 算法在迭代前期进行全局探索，尽可能扩大搜索解的范围，而在迭代后期更倾向于局部搜索使算法快速收敛，同时保持粒子在整个迭代过程中变异的随机性。具体公式如下：

$$\omega=(\omega_{\text{init}}-\omega_{\text{end}})\cdot\frac{\text{Maxiter}-\text{iter}}{\text{iter}}+z\cdot\omega_{\text{end}} \tag{3.44}$$

其中，ω_{init} 和 ω_{end} 分别是 ω 的起始值和结束值，z 是混沌变量，遵循式(3.44)，Maxiter 和 iter 分别是算法的最大迭代次数和当前的迭代次数。混沌下降的惯性权重能够在保持原有变化趋势的同时又具有混沌特性。

2）基于样例池机制的社会学习策略

社会学习在群体智能的学习行为中扮演着重要角色，有利于种群中的个体在不会增加自身试验和错误代价的前提下，从其他个体上学习行为。社会学习粒子群优化(Social Learning Particle Swarm Optimization，SLPSO)是一种改进 PSO 算法，其中提出的社会学习机制极大地提升了粒子群的全局搜索能力并改善算法性能。在该算法的基础上，文献[21]的研究工作设计了一种单样例学习和均值样例学习来分别代替 SLPSO 的模仿分量和社会认知分量。但是这些更新公式都不能直接应用在离散问题上。因此，受 SLPSO 算法的启发，本节设计了一个适用于离散问题的新的社会学习策略以融入粒子的更新公式中，从而提升粒子群优化算

法的性能。

定义 3.9　学习样例池　种群 $S=\{X_i \mid 1\leqslant i\leqslant M\}$ 中所有粒子按适应度值大小升序排列：$S=\{X_1,\cdots,X_{i-1},X_i,X_{i+1},\cdots,X_M\}$，则粒子群 $\mathrm{EP}=\{X_1,\cdots,X_{i-1}\}$ 构成了粒子 X_i 的学习样例池。

性质 3.5　不同的粒子，其学习的样例池不同；同一个粒子，在每次迭代中，其学习的样例池也不一定相同。

性质 3.6　粒子在每一次的社会学习中，从自身当前的样例池中随机选择一个粒子作为学习对象。特别地，当粒子选择的学习对象是样例池中的第一个粒子 X_1，则粒子此时学习的是全局最优解 X_G。

图 3.17 显示了粒子学习的样例池。五角星是未知的待探索的最优解，粒子 X_G 是目前种群找到的最优解。对粒子 X_i 来说，圆形粒子是相对落后的粒子，而三角形粒子比 X_i 更接近全局最优解，具有比 X_i 更好的适应度值，这些粒子（包括 X_G）构成了 X_i 的样例池，其中每一个粒子都有可能成为其社会学习的对象。粒子 X_i 在每一次更新时会随机选择样例池中的任意一个粒子 X_k（$1\leqslant k\leqslant i-1$）并学习其经验（即该粒子的历史最优方案 X_k^P），来完成自身的社会学习过程。这样的社会学习方式，一方面允许粒子在进化过程中通过不断学习不同的优秀个体来提升自己（根据性质 3.5 可得），有利于种群的多样化发展；另一方面，粒子在更换学习对象的过程中也有一定的概率学习到全局最优粒子（根据性质 3.6 可得），从而使得算法能够较好地平衡种群的探索与开发。

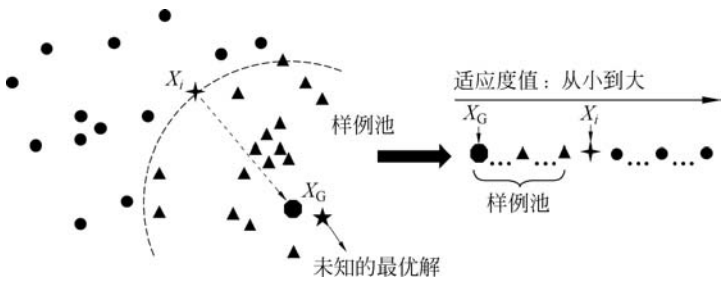


图 3.17　粒子的学习样例池示意图

3）粒子编码

算法采用的边点对的粒子编码策略，详细介绍见 3.2.1 节。

4）适应度函数

一个粒子就代表一棵候选 Steiner 树，粒子的适应度值就反映了 Steiner 树的质量。由于本节工作考虑的是线长驱动的 XSMT 问题。因此，设定粒子的适应度值为对应布线树的长度。一棵 X 结构 Steiner 树的长度等于这棵树中所有边线段的长度总和，具体计算公式如下：

$$\mathrm{fitFunc}=L(T_X)=\sum_{e_i\in T_X} l(e_i) \tag{3.45}$$

其中，$l(e_i)$表示布线树 T_X 中边线段 e_i 的长度。

5）粒子更新公式

由于 SPSO 算法不能直接应用在 XSMT 这个离散问题上，因此，本节借助交叉和变异操作来实现 SLPSO 的离散化，从而更好地解决 XSMT 问题。在 SLDPSO-WL-XSMT 算法中，粒子遵循以下的更新公式：

$$X_i^t = \mathrm{SF}_3(\mathrm{SF}_2(\mathrm{SF}_1(X_i^{t-1},\omega),c_1),c_2) \tag{3.46}$$

其中，ω 为惯性权重因子，c_1 和 c_2 为加速因子，SF_1 为惯性分量，SF_2 和 SF_3 分别为个体认知分量和社会认知分量。在 SLDPSO-WL-XSMT 中，SF_1 通过变异操作实现，变异程度决定了多大程度上保留上一次迭代的状态；SF_2 和 SF_3 通过交叉操作实现，即分别与个体历史最优方案(pbest)和样例池中任意优秀粒子的历史最优方案(kbest)进行部分基因交叉。

SPSO 的社会认知分量，是通过与种群最优粒子(gbest)学习来实现的。gbest 是种群中所有粒子 pbest 中最优的一个，这是综合了种群(社会)中所有粒子的学习经验而得出的最佳学习对象。值得注意的是，每一次迭代粒子都以某种概率向 gbest 学习。这意味着，一旦算法搜寻到一个局部极值，这个局部极值会立即成为这一代中的 gbest，是其他所有粒子进行社会学习的对象。而所有粒子同时只向同一个对象学习，很容易陷入局部寻优的僵局，使得在后面的迭代中粒子会反复在局部极值附近寻找最优值。因此，本节将基于样例池的社会学习策略融入离散的 PSO 中，同时使用混沌下降的惯性权重，以更好地解决 WL-XSMT 问题。

(1) 惯性分量。算法使用 SF_1 表示粒子的惯性分量，通过变异操作实现，表示如下：

$$W_i^t = \mathrm{SF}_1(X_i^{t-1},\omega) = \begin{cases} M_p(X_i^{t-1}), & r_1 < \omega \\ X_i^{t-1}, & \text{否则} \end{cases} \tag{3.47}$$

其中，W_i^t 是变异后的粒子，惯性权重 ω 决定了粒子进行变异操作的概率，遵循式(3.44)的混沌下降公式，$M_p()$为针对 PSP 变换的变异操作，r_1 是[0,1)区间的随机数。

SLDPSO-WL-XSMT 算法采用两点变异。如果产生的随机数 $r_1<\omega$，算法将随机选出两条边，更换这两条边的选择方式(选择 0～3)，从而达到随机改变布线树拓扑结构的目的；如果产生的随机数 $r_1\geqslant\omega$，则保持布线树不变。图 3.18 给出了一棵含有 6 个引脚的布线树的变异示意图，布线树下方是粒子的编码(图中未给出粒子适应度值)。从图 3.18 可以看到，算法随机选择了粒子 X_i：*321 122 253* 42 *3 260 673* 的两条边 $m_1(3,2,1)$和 $m_2(2,5,3)$，经过 SF_1 操作后，m_1 和 m_2 的布线方式分别由选择 1 和选择 3 变成选择 2 和选择 0。

(2) 个体认知分量。算法使用 SF_2 表示粒子的个体认知分量，通过交叉操作实现，表示如下：

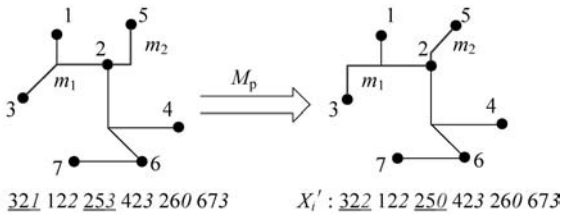


图 3.18　SLDPSO-WL-XSMT 算法的变异操作

$$S_i^t = \mathrm{SF}_2(W_i^t, c_1) = \begin{cases} C_p(W_i^t, X_i^P), & r_2 < c_1 \\ W_i^t, & \text{否则} \end{cases} \tag{3.48}$$

其中，S_i^t 是交叉后的粒子，c_1 决定了粒子 W_i^t 与个体历史最优 X_i^P 交叉的概率，$C_p()$ 为交叉操作，r_2 是[0,1)内的随机数。这个阶段粒子的学习对象是 X_i^P。

(3) 社会认知分量。算法使用 SF_3 表示粒子的社会认知分量，通过交叉操作实现，表示如下：

$$X_i^t = \mathrm{SF}_3(S_i^t, c_2) = \begin{cases} C_p(S_i^t, X_k^P), & r_3 < c_2 \\ S_i^t, & \text{否则} \end{cases} \tag{3.49}$$

其中，c_2 决定了粒子 X_i 与学习样例池中的任意一个粒子的历史最优解 ($X_k^P(1 \leqslant k \leqslant i-1)$)交叉的概率，$r_3$ 是[0,1)内的随机数。这个阶段粒子的学习对象是样例池中的优秀粒子的历史最优方案 X_k^P。

式(3.48)和式(3.49)中的交叉操作如下：当产生的随机数 $r_2 < c_1$(或 $r_3 < c_2$)时，执行交叉操作，对一棵含 n 个引脚的 Steiner 树：算法首先随机产生需要交叉的连续区间$[C_{\text{start}}, C_{\text{end}}]$，其中 C_{start} 和 C_{end} 是[1,$n-1$]区间的随机整数；然后，找到粒子 X_i 的学习对象(X_i^P 或 X_k^P)在区间$[3\times(C_{\text{start}}-1), 3\times C_{\text{end}}]$上的编码；最后，将粒子 X_i 该区间上的 pseudo-Steiner 编码值替换其学习对象该区间上的编码。如图 3.19 所示，粒子 X_i(321 122 253 423 260 673)是待交叉的粒子，粒子(321 121 250 423 262 673)是其学习对象，在式(3.48)中呈现为个体历史最优粒子 X_i^P，在式(3.49)中呈现为样例池中任意粒子的历史最优 X_k^P。算法首先产生一个连续区间$[C_{\text{start}}, C_{\text{end}}]=[2,4]$，对应需要交叉的边为 e_1、e_2 和 e_3；接着，找到 X_i 的学习对象在区间$[3\times(2-1), 3\times 4]=[3,12]$上的编码：*121 250 423*，对应图中蓝色实线段；最后，将粒子 X_i 该区间上的 pseudo-Steiner 编码值替换为上述编码串中的值。交叉操作结束后 X_i 中的边 e_1、e_2 和 e_3 的布线方式分别由选择 2、选择 3 和选择 3 转变成选择 1、选择 0 和选择 3，同时其余边的拓扑结构不变。

经过上述交叉操作，粒子 X_i 能够从更优秀的粒子上学习到部分基因，按照这样，反复的迭代学习就能使得粒子 X_i 逐渐向全局最优位置靠拢。

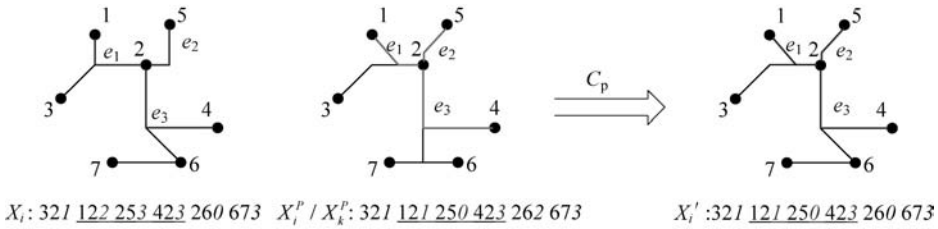


图 3.19 SLDPSO-WL-XSMT 的交叉操作

6) SLDPSO 搜索的具体步骤

SLDPSO 搜索的算法步骤可以总结如下。

步骤 1,通过最小生成树方法构建初始布线树,并随机初始化每条边的连接方式为选择 0 或选择 1。

步骤 2,根据边点对编码方式进行粒子编码,初始化种群及每个粒子的历史最优解 pbest 和样例池 EP。

步骤 3,根据式(3.46)～式(3.49)更新每个粒子的速度和位置。其中,ω 采用混沌下降策略(式(3.44)),c_1 和 c_2 分别采用线性递减和线性递增策略。

步骤 4,计算每个粒子的适应度值。

步骤 5,更新每个粒子的 pbest 和 EP,更新粒子的 kbest。

步骤 6,如果达到设定的最大迭代次数,输出所有 pbest 中的最优解 gbest,结束算法;否则,返回步骤 3。

性质 3.7 SLDPSO-WL-XSMT 算法的 SLDPSO 搜索阶段中,初始化边的布线选择时,相比随机初始化为 4 种选择方式(选择 0～3),初始化为选择 0 或选择 1 的初始边选择策略,能够有效提高解的质量。

由于曼哈顿结构的走线方式(选择 2 和选择 3)限制了线长的优化,而 45°和 135°的走线方式,更能充分利用有限的布线资源。因此,在某种程度上,将布线树的边初始化为选择 0 和选择 1,能够产生较为优质的初始粒子,加快算法找到最优方案。

性质 3.8 本节提出的基于样例池机制的 SLDPSO 方法能较好地平衡算法的全局探索和局部开发能力,从而有效解决 XSMT 问题。

值得注意的是,在粒子群优化算法中,粒子是基于历史信息来更新的,包括由每个粒子自身找到的最优解(X_i^P)和整个群体找到的最优解(X_G)。而在本节算法中,粒子不仅会根据自身的历史经验(X_i^P)来学习,也会学习当前群体中任意一个更好的粒子的历史最优(X_k^P)(有概率学到全局最优解 X_G)。一方面,混沌下降的惯性权重增大了种群进化过程中的多样性,有利于增强算法的探索能力,从而突破局部最优解;另一方面,粒子 X_k 具有比 X_i 更好的适应度值(在第 t 次迭代时,总有 $f(X_k^{P,t})\leqslant f(X_k^t)\leqslant f(X_i^t)$),粒子没有学习 X_k,而是直接学习 X_k 的历史最优

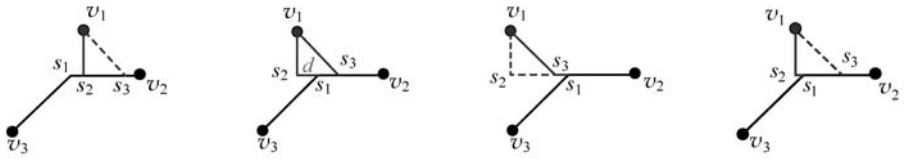
经验，这样有利于加强算法的开发能力，加速收敛。

2. 线长减少阶段

1) 局部最优拓扑的分析

粒子飞行的随机性是 PSO 算法具有如此强大搜索能力的主要原因之一，针对大规模离散问题，PSO 能在有限的时间内找到令人满意的解，但是通常离未知的最优解还会有一定差距。另外，X 结构的引入在一定程度上减少了 Steiner 树的线长，但并不是在所有情况下，X 结构的走线方式都优于直角结构。因此，本节设计了一个局部拓扑优化策略，对 SLDPSO 搜索阶段找到的 Steiner 树进行更精确的调整。

如图 3.20 所示，v_2 是一个 2-度节点，分别与 v_1 和 v_3 连接。在已有的拓扑结构(v_2,v_3,s_1)的基础上(其中，s_1 是连接点 v_2 和 v_3 的 PS)，考虑 v_2 和 v_1 之间的连接方式，以使得这个 2-度节点 v_2 具有最优的拓扑结构。图 3.20 列出了几种典型的情况，其中，s_2 和 s_3 分别是 v_2 通过直角结构和 X 结构与 v_1 相连形成的 PS。



(a) 拓扑结构一　(b) 拓扑结构二　(c) 拓扑结构三　(d) 拓扑结构四

图 3.20　X 结构下 2-度节点的局部最优拓扑

(1) 当 s_2 位于 s_1 右侧时(见图 3.20(a))：v_2 通过选择 3 连接到 v_1，需要新增线长 $e(s_2,v_1)$；v_2 通过选择 0 连接到 v_1，需要新增线长 $e(s_3,v_1)$。很明显，(v_2,v_1,s_2)是最优的结构，因为直角边 $e(s_2,v_1)$的线长始终小于斜边 $e(s_3,v_1)$的线长。

(2) 当 s_2 位于 s_1 左侧时(见图 3.20(b)、(c)和(d))：在图 3.20(b)中，有 $e(s_1,s_2)=d$，使得 $e(s_1,s_2)+e(s_2,v_1)=e(s_3,v_1)$。在这种情况下，针对已给的拓扑结构，$(v_2,v_1,s_2)$和$(v_2,v_1,s_3)$两种连接方式得到的总线长是一样的；而当 $e(s_1,s_2)>d$ 时(见图 3.20(c))，由于始终有 $e(s_1,s_3)+(e(s_1,s_2)+e(s_2,v_1))>e(s_1,s_3)+e(s_3,v_1)$，所以 v_2 通过选择 0 连接到 v_1，即(v_2,v_1,s_3)是最优的结构；当 $e(s_1,s_2)<d$ 时(见图 3.20(d))时，很明显，相比 X 结构，直角结构的布线方式能获得更短的线长，即(v_2,v_1,s_2)是最优结构。

上述分析一方面说明了仅仅依靠 X 结构或直角结构都不能获得最佳的拓扑，二者的有效结合才能得到更短的线长。另一方面，基于这种思考模式，本节设计了相应的局部拓扑优化策略来进一步优化 SLDPSO 算法得到的 XSMT。

2) 线长优化的具体步骤

性质 3.9　SLDPSO-WL-XSMT 算法提出的局部拓扑优化策略通过逐一调整线网中所有引脚的局部拓扑结构，能有效减少 XST 的线长。局部拓扑优化策略的

伪代码如算法 3.10 所示。

假定线网中度数最多的引脚度数为 Q，q 是用户自定义参数，且满足：$1 \leqslant q \leqslant Q$。考虑到实际线网结构的复杂性，不建议对引脚的所有邻接边进行调整，尤其是 Q 很大，并且线网中存在较多的度接近 Q 的引脚的时候，此时会付出较大的时间代价。因此，局部拓扑优化策略设定最多遍历每个引脚的 q 条邻接边并进行适当的调整。同时，算法可以通过调整参数 q 的大小，在优化效果和时间之间做一个折中。q 的值越接近 Q，优化效果就越好。本节设定 $q=\lceil 0.8\times Q \rceil$。

算法 3.10 局部拓扑优化策略

```
输入：XST
输出：XSMT
Begin
1.    // 1.记录 XST 中每个点的度数及其邻接点列表
2.    for each terminal v_i in P of XST do
3.      Initialize v_i.degree=0, v_i.adj_list[]=∅;
4.    end for
5.    for each edge(v_i, v_j) of XST do
6.      if v_i.degree < q then
7.        v_i.degree +=1;
8.        add v_i to v_j.adj_list[];
9.      end if
10.     if v_j.degree < q then
11.       v_j.degree +=1;
12.       add v_j to v_i.adj_list[];
13.     end if
14.   end for
15.   // 2. 所有点按度降序排列，每个点的邻接点列表中的元素也按照点的度数降序排列
16.   Sort(P) in decreasing order according to v_i.degree;
17.   for each terminal v_i in P of XST do
18.             Sort(v_i.adj_list) in decreasing order according to v_i.adj_list[k].degree;
//1≤k≤v_i.degree
19.   end for
20.   // 3.依次优化各个点的局部拓扑结构
21.   for each terminal v_i in P do
22.     for each edge(v_i, v_i.adj_list[k]) connected to v_i do
23.       curChoice=getChoice(v_i, v_i.adj_list[k]); //获取边的连接方式
24.         if choice==bestChoice then //如果已经是最优结构，则不处理
```

```
25.              Continue;
26.            else //否则更新当前局部拓扑为最优结构
27.              ResetChoice(v_i,v_i.adj_list[k],bestchoice);
28.          end if
29.        end for
30.      end for
     End
```

布线树中的每个节点都有两个属性：一个是 degree，存储的是这个节点的度，即其邻接点个数；另一个是邻接点列表 adj_list[]，用来存储各个邻接点。该策略的具体实现步骤如下。

步骤 1，记录 XST 中每个点的度数及其邻接点列表。在这一步骤中，算法通过遍历 XST 的每一条边来记录每个点的度数，并将其邻接点加入对应列表。特别地，对于度大于 q 的点来说，算法仅仅记录该点的 q 个邻接点。

步骤 2，设置局部拓扑优化的顺序为：从度数大的点往度数小的点优化。通过将所有点按度数从大到小排列，同时将每个点的邻接点也按度数从大到小排列，来实现先优化密集区域的拓扑结构，再优化稀疏区域的拓扑结构。

步骤 3，优化局部拓扑结构。按照步骤 2 的顺序，依次优化各个点的局部拓扑结构。对于 P 中的每个点 v_i，遍历该点的每一条邻接边(v_i，v_i.adj_list[k])，其中 $1 \leqslant k \leqslant v_i$.degree，并通过调整该条边的选择方式(选择 0～3)，用局部最优拓扑替换当前的拓扑结构，若已经是最优结构，则不进行处理。

局部拓扑优化策略通过不断调整每个点的每条邻接边的选择方式，以获得在当前 XST 拓扑下的局部最优结构，从而优化 XST 的线长。

3. 算法的流程和复杂度分析

SLDPSO-WL-XSMT 算法由社会学习离散粒子群搜索阶段和线长减少阶段构成。首先，通过基于样例池机制的社会学习粒子群的迭代搜索以寻找一棵较高质量的 X 结构 Steiner 树，再利用局部拓扑优化策略对上一阶段的解方案进行线长优化，以得到最终的 XSMT。算法的整体流程如图 3.21 所示。

引理 3.2　假设种群大小为 M，迭代次数为 T，引脚个数为 n，则 SLDPSO-WL-XSMT 的时间复杂度为 $O(MT\times(M\log_2 M+n\log_2 n))$。

证明：

(1) SLDPSO 搜索阶段。在 SLDPSO 搜索阶段的步骤 3～步骤 5 中，包含变异与交叉操作、计算适应度和更新样例池。变异和交叉操作的时间复杂度均为常数时间 $O(1)$。布线树线长的计算时间主要由排序时间决定，为 $O(n\log_2 n)$。由于每一次迭代结束，粒子会进行样例池的更新，消耗的时间为排序所花的时间，即 $O(M\log_2 M)$。故算法内部循环的复杂度为 $O(M\log_2 M+n\log_2 n)$。加上外部循环的条件 M 和 T，这一阶段的时间复杂度为 $O(MT\times(M\log_2 M+n\log_2 n))$。

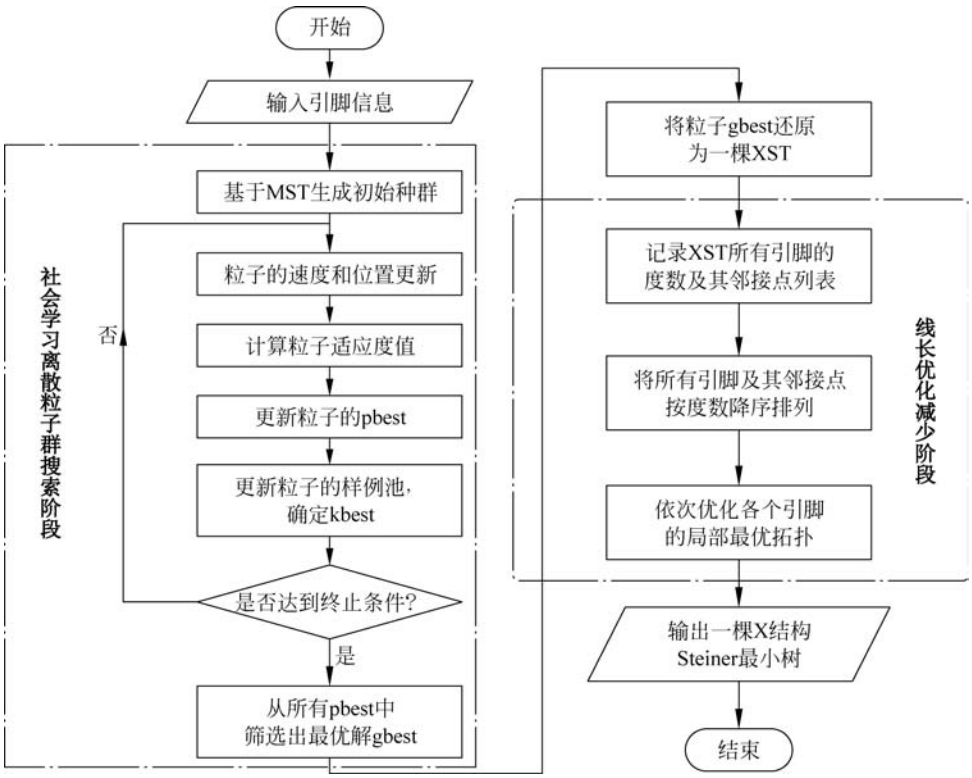


图 3.21 SLDPSO-WL-XSMT 算法的整体流程图

(2) 线长减少阶段。该阶段分为 3 个步骤。步骤 1 中记录 XST 中所有点的度数及其邻接点列表所花时间为 $O(n)$。步骤 2 需要对所有引脚按度排序，所需时间为 $O(n\log_2 n)$，并且需要对每个引脚的邻接表元素按度排序。由于在本节工作中每个引脚的邻接表元素个数最大不超过 4，可忽略这部分的排序时间，则步骤 2 的总时间复杂度为 $O(n\log_2 n)$。步骤 3 中优化所有点的局部拓扑结构时需要遍历每个引脚，时间复杂度为 $O(n)$。因此，这一阶段的时间复杂度为 $O(n\log_2 n+2n)$，即其近似复杂度为 $O(n\log_2 n)$。

综上所述，SLDPSO-WL-XSMT 算法的近似时间复杂度为 $O(MT\times(M\log_2 M+n\log_2 n))$。

3.6.3 实验仿真与结果分析

为了验证本节算法和相关策略的有效性，本节在基准测试电路 GEO 上开展实验，并给出了详细的实验对比数据。该测试电路的规模包括总引脚数为 8～1000 的线网。本节算法设置种群大小为 100，粒子迭代次数为 1000，其余参数与文献[22]设置的参数一致。考虑到粒子群算法的随机性，本节所有实验中的平均值均通过独立运行 20 次得来。

1. 初始边选择策略的有效性验证

为验证初始化边选择策略的有效性，本节对比了初始边选择为选择0～3（记为4-init）和本节算法所采用的初始边选择为选择0或选择1（记为2-init）所得到的布线树的线长，如表3.18所示。从表3.18中可以看到，对于所有的测试用例，2-init的初始边选择方式相比4-init能够取得更好的平均线长，平均优化线长6.762%；在最优粒子方面，除了测试电路1以外，2-init能够获得比4-init具有更短线长的最优粒子。很明显，“平均线长”代表了初始种群的质量，而“最优线长”代表了种群中最有潜力的粒子。因此，2-init的初始边选择方式能够为算法带来更多优质的初始粒子，从而加快算法找到最优的解方案。

表3.18　2-init初始边选择策略的有效性验证

电　路	引脚数	最优线长			平均线长		
		4-init	2-init	优化率/%	4-init	2-init	优化率/%
1	8	17 434	17 506	−0.411	18 939	17 751	6.269
2	9	18 377	18 041	1.831	20 211	18 283	9.540
3	10	19 823	19 703	0.603	21 710	19 944	8.134
4	20	33 874	32 810	3.142	35 624	33 072	7.163
5	50	51 022	49 138	3.692	52 884	49 340	6.702
6	70	59 494	57 547	3.272	61 093	57 721	5.519
7	100	73 271	70 365	3.966	75 104	70 569	6.038
8	410	151 451	143 302	5.381	153 109	143 683	6.156
9	500	164 033	155 888	4.965	166 340	156 481	5.927
10	1000	235 507	222 235	5.636	237 172	222 542	6.168
均值				**3.208**			**6.762**

2. SLDPSO方法的有效性验证

为验证SLDPSO-WL-XSMT算法第一阶段，即基于样例池机制的社会学习DPSO（记为SLDPSO-XSMT）方法的有效性，本节将SLDPSO-XSMT与文献[4]提出的DPSO-XSMT算法进行比较，实验结果如表3.19所示。为了实验的公平，DPSO-XSMT算法的数据均是在本节设定的参数下运行得来的，其中惯性权重设置0.95～0.4线性递减。本节比较了20次独立实验中，SLDPSO方法和DPSO方法找到的最佳XSMT的线长、平均线长以及标准差。

表3.19　SLDPSO-XSMT和DPSO-XSMT算法的性能比较

电路	引脚数	最优线长			平均线长			标准差		
		DPSO	SLD PSO	优化率/%	DPSO	SLD PSO	优化率/%	DPSO	SLD PSO	优化率/%
1	8	16 918	16 918	0.000	16 918	16 918	0.000	0	0	0.00
2	9	18 041	18 041	0.000	18 041	18 041	0.000	0	0	0.00

续表

电路	引脚数	最优线长			平均线长			标准差		
		DPSO	SLD PSO	优化率/%	DPSO	SLD PSO	优化率/%	DPSO	SLD PSO	优化率/%
3	10	19 696	19 696	0.000	19 696	19 696	0.000	0	0	0.00
4	20	32 193	32 193	0.000	32 213	32 207	0.019	11	10	11.29
5	50	47 960	47 953	0.014	48 038	47 982	0.118	83	32	61.45
6	70	56 279	56 278	0.002	56 448	56 341	0.191	98	35	64.73
7	100	68 578	68 486	0.133	68 911	68 697	0.311	182	104	42.69
8	410	141 427	140 902	**0.371**	141 852	141 143	**0.499**	238	108	**54.50**
9	500	154 365	153 889	0.308	154 742	154 056	0.443	204	97	52.70
10	1000	220 795	219 955	0.380	221 142	220 240	0.408	282	167	40.68
均值				**0.121**			**0.199**			**32.80**

从表 3.19 可以看出，相对 DPSO 方法，本节提出的 SLDPSO 方法在测试电路 4～测试电路 10 上(引脚数为 20～1000)取得了线长优化。其中，对电路 8(引脚数为 410)的优化效果最好，得到的最佳 XSMT 的线长优化了 0.371%、平均线长减少了 0.499%以及标准差优化了 54.50%。表 3.19 中的实验数据证明了本节提出的社会学习策略的有效性。

值得注意的是，SLDPSO 相对早期的 DPSO 方法，其标准差大大减少，说明算法的稳定性得到了极大的提升。其一是由于本节算法采用了混沌下降的惯性权重。因为粒子群的惯性保持部分是通过变异操作实现的，而惯性权重的大小决定了变异的概率。单纯的线性递减方式，使得粒子群变异的频率在很大程度上依赖于设定的初始值和结束值，因此会出现在迭代初期变异的频率很高，而在迭代后期变异的频率又很低的情况。由于混沌搜索的伪随机性、遍历性和对不依赖初始值的特点，使得惯性权重的值能较均匀地分布在线性递减的区域中，使得变异的概率相对稳定，同时又能随迭代次数的增长而降低。其二是因为粒子的社会学习通过样例池实现，扩大了学习对象的范围，而 DPSO 方法只通过学习种群最优粒子(gbest)来进行社会学习。对于种群中落后的粒子来说，由于与 gbest 的差距较大，它们的一次更新可能产生很大的位置改动，从而导致算法稳定性较差。而通过样例池随机选择学习对象，从一定程度缓解了这种跳跃的情况，从而提高了算法的稳定性。

3. 局部拓扑优化策略的有效性验证

SLDPSO-WL-XSMT 算法提出的局部拓扑优化策略通过逐一调整线网中所有引脚的局部最优拓扑，能有效减少 XST 的线长。为验证该策略的有效性，本节分别对比了使用该策略前后算法获得的最佳 Steiner 树的线长、最差 Steiner 树的线长以及平均线长，如表 3.20 所示。可以看到，局部拓扑优化策略对 50 个引脚以上的测试用例有明显优化效果，并且引脚数越多，该策略的优化效果越强。对于

1000 个引脚的线网来说，该策略能减少 2.381%的平均线长，其中针对最不理想的 Steiner 树，其线长为 220 629，经过局部拓扑优化之后，线长减少到 214 950，减少了 2.574%。

表 3.20　局部拓扑优化策略使用前后的实验结果对比

电路	引脚数	最佳线长			最差线长			平均线长		
		之前	之后	优化率/%	之前	之后	优化率/%	之前	之后	优化率/%
1	8	16 918	16 918	0.000	16 918	16 918	0.000	16 918	16 918	0.000
2	9	18 041	18 041	0.000	18 041	18 041	0.000	18 041	18 041	0.000
3	10	19 696	19 696	0.000	19 696	19 696	0.000	19 696	19 696	0.000
4	20	32 193	32 193	0.000	32 214	32 214	0.000	32 205	32 205	0.001
5	50	47 960	47 953	0.014	48 032	47 953	0.165	47 977	47 953	0.051
6	70	56 281	56 271	0.018	56 489	56 307	0.323	56 351	56 280	0.125
7	100	68 486	68 335	0.221	69 019	68 335	0.991	68 697	68 347	0.510
8	410	140 902	139 082	1.291	141 332	139 122	1.564	141 143	139 074	1.466
9	500	**153 889**	**151 455**	1.582	**154 196**	**151 441**	1.786	154 056	151 408	1.719
10	1000	219 970	214 991	2.263	**220 629**	**214 950**	**2.574**	220 233	214 990	**2.381**
均值				0.539			0.740			0.625

另外，可以很明显看到，对于所有的测试用例，最差 XST 的线长减少率均高于最佳 XST 的线长减少率，并且最差的 XST 经过局部拓扑优化后，有可能得到比最佳 XST 更短的线长。例如，对于测试电路 9 来说，由 SLDPSO 搜索得到的最佳 XST 线长为 153 889，最差的为 154 196，而在经过该优化策略后，最差的 XST 线长为 151 441，比最好的 XST 优化后的线长 151 455 还要短。表 3.20 中的实验数据证明了局部拓扑优化策略在减少线长上的有效性。

4. 与现有 SMT 算法的对比

为验证本节提出的 SLDPSO-WL-XSMT 算法的有效性，本节先后对比了 SLDPSO-WL-XSMT 算法与现有的 SMT 算法的线长优化能力，包括 RSMT 和 XSMT 算法。表 3.21 给出了本节算法与文献[3]提出的 DPSO(R)和文献[22]提出的 HTS-PSO(R)两种 RSMT 算法的线长优化能力的对比。表中数据显示，在测试电路 1～电路 10 上，DPSO(R)、HTS-PSO(R)和本节提出的 SLDPSO-WL-XSMT 算法得到的 SMT 的线长平均比为 1.114∶1.089∶1。

其中，针对引脚数量最小的测试电路 1，DPSO(R)和 HTS-PSO(R)算法求得的 SMT 线长分别比本节算法长 6.0%和 4.6%；针对规模最大的测试电路 10，DPSO(R)算法得到的 SMT 线长是本节算法的 1.141 倍，HTS-PSO(R)算法得到的 SMT 线长比本节算法长 11.6%。并且观察表 3.21 中的数据可以看到，针对引脚数越多的线网，本节算法得到的 Steiner 最小树线长优化越明显。

表 3.21 SLDPSO-WL-XSMT 和两种 RSMT 算法的线长优化能力对比

电路	引脚数	平均线长			归一化值		
		DPSO(R)	HTS-PSO(R)	本节算法	DPSO(R)	HTS-PSO(R)	本节算法
1	8	17 931	17 693	16 918	**1.060**	**1.046**	1.000
2	9	20 503	19 797	18 041	1.136	1.097	1.000
3	10	21 910	21 226	19 696	1.112	1.078	1.000
4	20	35 723	35 072	32 205	1.109	1.089	1.000
5	50	53 383	52 025	47 953	1.113	1.085	1.000
6	70	61 987	61 129	56 280	1.101	1.086	1.000
7	100	76 016	74 416	68 347	1.112	1.089	1.000
8	410	156 520	153 672	139 074	1.125	1.105	1.000
9	500	170 273	166 592	151 408	1.125	1.100	1.000
10	1000	245 201	239 824	214 990	**1.141**	**1.116**	1.000
均值					**1.114**	**1.089**	**1.000**

同时，本节也将提出的算法与 3 种性能较好的 XSMT 算法进行线长优化能力对比，包括文献[23]的 DDE(X)算法、文献[4]的 DPSO(X)算法和文献[22]的 HTS-PSO(X)算法。表 3.22 和表 3.23 分别对比了上述算法所求 XSMT 的平均线长和标准差，这两个评价指标分别反映了算法的线长优化能力和稳定性。

表 3.22 SLDPSO-WL-XSMT 和 3 种 XSMT 算法的平均线长优化能力对比

电路	引脚数	平均线长				归一化值			
		DDE(X)	DPSO(X)	HTS-PSO(X)	本节算法	DDE(X)	DPSO(X)	HTS-PSO(X)	本节算法
1	8	16 911	16 918	16 921	16 918	1.000	1.000	1.000	1.000
2	9	18 039	18 041	18 023	18 041	1.000	1.000	0.999	1.000
3	10	19 469	19 696	19 397	19 696	0.988	1.000	0.985	1.000
4	20	32 342	32 213	32 063	32 202	1.004	1.000	0.996	1.000
5	50	48 668	48 038	48 027	47 953	1.015	1.002	1.002	1.000
6	70	57 255	56 448	56 350	56 278	1.017	1.003	1.001	1.000
7	100	70 686	68 911	68 625	68 347	1.034	1.008	1.004	1.000
8	410	147 115	1 441 852	140 898	139 074	1.058	1.020	1.013	1.000
9	500	159 672	154 742	153 708	151 408	1.055	1.022	1.015	1.000
10	1000	**232 359**	221 142	219 954	**214 990**	**1.081**	**1.029**	**1.023**	1.000
均值						**1.025**	**1.008**	**1.004**	**1.000**

表 3.23 SLDPSO-WL-XSMT 和 3 种 XSMT 算法的标准差对比

电路	引脚数	标准差				归一化值			
		DDE(X)	DPSO(X)	HTS-PSO(X)	本节算法	DDE(X)	DPSO(X)	HTS-PSO(X)	本节算法
1	8	34	0	16	0	—	—	—	—
2	9	41	0	0	0	—	—	—	—
3	10	133	0	0	0	—	—	—	—
4	20	165	11	31	10	16.01	1.10	3.01	1.000
5	50	827	83	94	0	—	—	—	—
6	70	1135	98	145	14	79.63	6.89	10.16	1.000
7	100	1802	182	150	14	**130.94**	**13.22**	**10.86**	1.000
8	410	3615	238	188	36	100.80	6.64	5.23	1.000
9	500	3688	204	181	46	80.21	4.44	3.93	1.000
10	1000	4089	282	184	34	119.95	8.28	5.40	1.000
均值						**87.92**	**6.76**	**6.43**	1.000

如表 3.22 所示，3 种 XSMT 算法与本节算法产生的 XSMT 线长平均比为 1.025∶1.008∶1.004∶1，说明本节算法总体上优于这 3 种 XSMT 算法。与 DDE(X)算法相比，在引脚数为 20 及以上的线网测试集中，本节算法具有明显的线长优化。其中，针对电路 10，SLDPSO-WL-XSMT 算法产生的 Steiner 树线长仅为 214 990，而 DDE(X)产生的线长为 232 359，比本节算法得到线长多出 8.1%。与 DPSO(X)算法相比，本节算法能够在所有测试数据上取得相同或者更短的线长。而 HTS-PSO(X)算法针对引脚数为 20 及以下的线网测试集来说，比本节算法略胜一筹，但在大规模线网中，SLDPSO-WL-XSMT 算法能够获得更短的线长。综合分析，本节算法较上述 3 种 XSMT 算法，在电路 5～电路 10 上有明显的线长优化，并且线网规模越大，SLDPSO-WL-XSMT 算法的线长优化能力越强。特别地，对于测试电路 10，DDE(X)、DPSO(X)和 HTSPSO(X)算法得到的线长比本节算法长 8.1%、2.9%和 2.3%。可见，SLDPSO-WL-XSMT 算法具有较强的线长优化能力，尤其针对大规模线网，效果显著。

在稳定性方面，SLDPSO-WL-XSMT 算法表现出明显优势。如表 3.23 所示，在电路 1～电路 3 和电路 5 上，本节算法能够将标准差降到 0，而对于其他测试用例来说，DDE(X)、DPSO(X)、HTSPSO(X)算法与本节算法产生的 XSMT 线长标准差平均比为 87.92∶6.76∶6.43∶1。特别地，SLDPSO-WL-XSMT 算法在测试电路 7(100 个引脚)上表现出最强的稳定性，而上述 3 种算法的实验结果波动性较大，其标准差分别是本节算法的 130.94 倍、13.22 倍和 10.86 倍。

3.6.4 小结

本节针对 VLSI 布线中线长驱动的 Steiner 最小树问题，提出了一个基于

SLDPSO 的 XSMT 算法。算法首先采用 X 结构作为 Steiner 树边的连接方式以充分利用布线资源。其次，为了克服 SPSO 算法容易陷入局部极值的不足，在粒子群搜索阶段使用混沌下降的惯性权重并结合变异算子实现 PSO 的惯性部分，以保持粒子在整个进化过程中变异的随机性，增强了算法的全局搜索能力；设计了一个基于样例池机制的社会学习策略并结合交叉算子实现 PSO 的社会认知部分以拓宽粒子的学习范围，从而保持种群进化的多样性。最后，在线长减少阶段，设计了一个局部拓扑优化策略，通过逐一调整线网中引脚的局部最优拓扑来进一步优化 Steiner 树的线长。实验证明，本节提出的 SLDPSO 方法具有较强的全局搜索能力和算法稳定性，从而更好地解决 Steiner 树问题。

3.7 本章总结

本章主要介绍了 X 结构 Steiner 最小树算法的基本原理以及多种的研究方法并提出了 5 种有效的 X 结构 Steiner 最小树算法：基于离散 PSO 的 X 结构 Steiner 最小树构建算法、基于离散差分进化的 X 结构 Steiner 最小树算法、基于多策略优化离散差分进化的 X 结构 Steiner 最小树算法、基于文化基因的 X 结构 Steiner 最小树算法、线长驱动的 XSMT 算法，详细描述了算法原理后在每节实验部分详细列出了与多个先进算法的实验对比结果，证实了本章提出算法的有效性。

参 考 文 献

[1] Garey M R, Johnson D S. The rectilinear Steiner tree problem is NP-complete[J]. *SIAM Journal on Applied Mathematics*, 1977, 32(4): 826-834.

[2] Jacob T P, Pradeep K. A multi-objective optimal task scheduling in cloud environment using cuckoo particle swarm optimization[J]. *Wireless Personal Communications*, 2019, 109(1): 315-331.

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

[4] 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.

[5] Zachariasen M. GeoSteiner Homepage, 2003. [Online]. Available: http://www.diku.dk/geosteiner.

[6] Coulston C S. Constructing exact octagonal Steiner minimal tree[C]//Proceeding of the 13th ACM Great Lakes Symposium on VLSI. New York, NY, USA: ACM Press, 2003, 28-29.

[7] Beasley, J. E. OR-Library: Distributing Test Problems by Electronic Mail[J]. *Journal of the Operational Research Society*, 1990, 41(11): 1069-1072.

[8] Kundu S, Roy S, Mukherjee S. K-nearest neighbour (KNN) approach using SAT based

technique for rectilinear steiner tree construction [C]//2017 Seventh International Symposium on Embedded Computing and System Design(ISED), Durgapur, India: IEEE, 2017: 1-5.

[9] Epitropakis M G, Tasoulis D K, Pavlidis N G, et al. Enhancing differential evolution utilizing proximity-based mutation operators [J]. *IEEE Transactions on Evolutionary Computation*, 2011, 15(1): 99-119.

[10] 刘漫丹. 文化基因算法(Memetic Algorithm)研究进展[J]. 自动化技术与应用, 2007, 26(11): 1-4.

[11] 谭立状, 负国潇, 张家华. 采用基于遗传算法的文化基因算法求解 TSP 问题[J]. 科技视界, 2016(05): 62-64.

[12] 许晶, 陈建利. 求解边坡最小安全系数的混合文化基因算法[J]. 福州大学学报(自版), 2017, 45(04): 559-565.

[13] 张亚娟, 刘寒冰, 靳宗信. 一种解决 VLSI 布局问题的文化基因算法[J]. 科技通报, 2013, 29(12): 154-156.

[14] Kundu S, Roy S, Mukherjee S. Sat based rectilinear steiner tree construction[C]//2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology(iCATccT), Bengaluru, India: IEEE, 2016: 623-627.

[15] Chen X, Liu G, Xiong N, et al. A survey of swarm intelligence techniques in VLSI routing problems[J]. *IEEE Access*, 2020, 8: 26266-26292.

[16] 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*, 2015, 20(2): 1-28.

[17] Bansal J C, Singh P K, Saraswat M, et al. Inertia weight strategies in particle swarm optimization [C]//2011 Third World Congress on Nature and Biologically Inspired Computing. IEEE, 2011: 633-640.

[18] Xu X, Rong H, Trovati M, et al. CS-PSO: chaotic particle swarm optimization algorithm for solving combinatorial optimization problems [J]. *Soft Computing*, 2018, 22(3): 783-795.

[19] Feng Y, Teng G F, Wang A X, et al. Chaotic inertia weight in particle swarm optimization [C]//The Second International Conference on Innovative Computing, Informatio and Control. IEEE, 2007: 475-475.

[20] Cheng R, JIN Y. A social learning particle swarm optimization algorithm for scalable optimization[J]. *Information Sciences*, 2015, 291: 43-60.

[21] Zhang X, Wang X, Kang Q, et al. Differential mutation and novel social learning particle swarm optimization algorithm[J]. *Information Sciences*, 2019, 480: 109-29.

[22] Liu G, Chen Z, Zhuang Z, et al. A unified algorithm based on HTS and self-adapting PSO for the construction of octagonal and rectilinear SMT[J]. *Soft Computing*, 2020, 24(6): 3943-3961.

[23] Wu H, Xu S, Zhuang Z, et al. X-architecture Steiner minimal tree construction based on discrete differential evolution[C]//The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery. Springer, Cham, 2019: 433-442.