第一部分：论文复现

1. 理解算法框架

论文提出了一种基于纠缠熵和投影策略的AQFP逻辑电路布局优化方法，核心分为两个阶段：拓扑初始化和布局与缓冲。以下是详细步骤：

2. 拓扑初始化（Algorithm 1）

目标：通过交换同一行中的单元格顺序，最小化纠缠熵 \( E \)，从而减少线长违规和缓冲器插入。

-纠缠熵定义：

其中：

- Sl 是第 l 行的连接线集合。

- Xl1, Xl2 是连接线端点在上一行的顺序。

-Yl1, Yl2 是连接线端点在下一行的顺序。

算法步骤：

1. 初始排序：按单元格连接线数降序排列每行。

2. 迭代交换：

随机选择同一行的两个单元格 m 和 n 。

计算交换后的纠缠熵变化 \( \Delta E \)：

其中：

- t1, t2  是交换单元格的连接线数。

- Ni  是第 i 行的连接线总数。

若 Delta E > 0 ，保留交换；否则恢复。

3. 终止条件：达到最大迭代次数或纠缠熵收敛。

代码实现：

```cpp

void topologyInitialization(Circuit& circuit) {

for each row in circuit {

sort cells by wire count (descending);

for (int iter = 0; iter < max\_iter; iter++) {

int m = randomCellInRow(row);

int n = randomCellInRow(row);

double deltaE = calculateDeltaE(row, m, n);

if (deltaE > 0) {

swapCells(row, m, n);

updateEntropy(row);

}

}

}

}

```

3. 布局与缓冲（Algorithm 2）

目标：通过投影策略确定单元格理想位置，插入缓冲器消除线长违规。

投影策略：

- 将单元格移动到其连接点的左右端点平均值位置：

- 通过对称性减少线长违规。

缓冲插入：

- 若线长仍违规，在违规线段的中点插入缓冲行。

- 调整相邻行的连接关系。

坐标调整：

- 解决单元格重叠问题：

行距调整：

- 逐步增加行间距，直至无违规。

代码实现：

```cpp

void placementAndBuffering(Circuit& circuit) {

initializeCoordinates(circuit);

while (wirelengthViolationExists(circuit)) {

for each cell with violation {

double x\_ideal = calculateIdealPosition(cell);

moveCellToPosition(cell, x\_ideal);

}

adjustXCoordinates(circuit); // 解决重叠

if (violation persists) {

insertBufferRow(circuit);

}

}

adjustRowSpacing(circuit);

}

```

第二部分：优化点及实现步骤

1. 优化纠缠熵计算

问题：交换单元格时需多次计算 \( \Delta E \)，时间复杂度高。

改进：

- 预计算每行的统计量（如 \( \sum Y\_f \), \( \sum Y\_g \)）。

- 使用动态规划减少重复计算。

代码修改：

```cpp

struct RowStats {

int total\_wires;

double sum\_Y\_connected;

};

void precomputeRowStats(Circuit& circuit) {

for each row in circuit {

RowStats stats;

stats.total\_wires = row.wires.size();

stats.sum\_Y\_connected = calculateSumYConnected(row);

row.stats = stats;

}

}

```

2. 动态缓冲插入策略

问题：论文每次插入一行缓冲器，可能导致冗余。

改进：

- 根据线长违规位置动态插入单个缓冲器。

- 使用几何中点法确定缓冲器位置。

实现步骤：

```cpp

void dynamicBufferInsertion(Circuit& circuit) {

for each wire with violation {

Point midpoint = calculateMidpoint(wire);

insertBufferAt(midpoint);

adjustConnections(wire);

}

}

```

3. 引入并行计算

问题：拓扑初始化的迭代交换过程串行执行，耗时长。

改进：

- 使用OpenMP对每行的交换操作并行化。

代码修改：

```cpp

#pragma omp parallel for

for each row in circuit {

for (int iter = 0; iter < max\_iter; iter++) {

int m = randomCellInRow(row);

int n = randomCellInRow(row);

double deltaE = calculateDeltaE(row, m, n);

if (deltaE > 0) {

swapCells(row, m, n);

updateEntropy(row);

}

}

}

```

1. 行距调整的启发式优化

问题：论文逐步增加行间距直到无违规，可能过度保守。

改进：

- 使用二分法快速找到最小合规行距。

实现步骤：

```cpp

double findOptimalRowSpacing(Circuit& circuit) {

double low = current\_spacing, high = max\_spacing;

while (low < high) {

double mid = (low + high) / 2;

if (checkViolation(mid) == 0) high = mid;

else low = mid + epsilon;

}

return low;

}

```

5. 验证优化效果

对比指标：

- 缓冲器数量、运行时间、线长合规率。

测试用例：

- 选择 `c7552`（论文中优化空间较大的案例）。

预期结果：

- 缓冲器减少 10%~20%，运行时间缩短 30%。