

這段 **Algorithm 1: Indirect Macro Dataflow Connection Extraction Algorithm** 的目標是**提取間接的宏數據流連接**，並確保兩種類型的間接宏連接被考慮。該算法通過兩個層級來分析數據流：

1. **在「單元叢集級別（cell cluster level）」提取連接**
2. **在「單元實例級別（cell instance level）」提取連接**

**🔹 解析步驟**

**🔸 1. 在 Cell Cluster Level 提取數據流（第 2-12 行）**

這部分的目標是**從單元叢集（cell cluster）中提取間接的宏連接**。

📌 **步驟**

1. **遍歷每個 cell cluster**（第 2 行）。
2. **獲取該 cluster 內所有與宏（macro）有連接的單元**，存入 cell\_connected\_macro\_vec（第 3 行）。
3. **取得與該 cell cluster 相關的 macro 數量 num**（第 4 行）。
4. **雙重迴圈遍歷所有可能的 macro 連接組合**（第 5-10 行）。
   * 內部 for 迴圈（第 6-9 行）確保**所有的宏都與其他宏建立連接**。
   * macro\_src 和 macro\_sink 表示兩個互相連接的宏（第 7-8 行）。
   * addconnection(macro\_src, macro\_sink) 創建一條連接（第 9 行）。

📌 **總結**

* 這個步驟確保了 **如果某個 cell cluster 內的宏有共享數據流**，它們會被視為互相連接。

**🔸 2. 在 Cell Instance Level 提取數據流（第 14-30 行）**

這部分的目標是**從更細粒度的單元（cell instance）層級提取間接宏連接**。

📌 **步驟**

1. **遍歷 cell fanin map 中的每個 vertex**（第 14 行）。
   * **vertex 代表網表中的一個節點**，即數據流的關鍵單元。
2. **建立 same\_vertex\_macro\_vec，用來存儲與該 vertex 連接的 macro**（第 15 行）。
3. **遍歷該 vertex 內所有的 pin（端點）**（第 16 行）。
   * 查找該 pin 所屬的 macro\_id（第 17 行）。
   * 如果 macro\_id 在某個 macro cluster 內，則加入 same\_vertex\_macro\_vec（第 18-19 行）。
4. **對 same\_vertex\_macro\_vec 內的所有 macro 進行兩兩連接**（第 22-29 行）。
   * same\_vertex\_macro\_vec 內的 macro 代表的是**與同一 cell 相連的所有宏**。
   * 透過雙重迴圈（第 24-29 行），確保這些宏之間建立連接。

📌 **總結**

* 這個步驟保證了**如果多個宏與同一 cell 透過數據流相連接**，它們之間會建立一條間接連接。

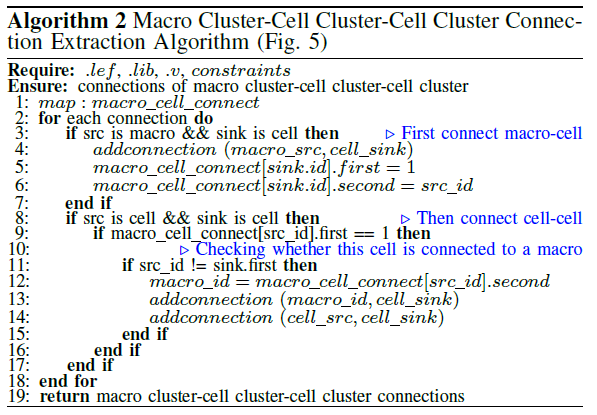
**🔹 整體算法總結**

1. **先在 cell cluster 層級**，透過 cell\_connected\_macro\_vec 確保 cluster 內的所有宏之間建立連接。
2. **再在 cell instance 層級**，透過 same\_vertex\_macro\_vec 檢查哪些宏與同一個 cell 相連，並建立連接。
3. **最終，該算法會回傳所有間接的宏數據流連接（return indirect macro connections）**。

**🔹 主要用途**

✅ **幫助更準確地分析數據流，進一步優化宏擺放（macro placement）**  
✅ **確保設計工具能夠識別宏之間的重要關係**，減少時序問題（timing violation）  
✅ **為後續的自動擺放佈線提供有價值的數據流資訊**，提升設計品質（QoR）

這個演算法的核心概念是**透過兩個層級的數據流提取來發現隱藏的宏連接**，從而幫助提升實體設計的自動化擺放品質！ 🚀



這段 **Algorithm 2: Macro Cluster-Cell Cluster-Cell Cluster Connection Extraction Algorithm** 主要目的是**提取宏叢集（macro cluster）與單元叢集（cell cluster）之間的連接**，並確保兩種類型的連接：

1. **宏（macro）與單元（cell）的直接連接**
2. **單元（cell）與單元（cell）之間的連接**，同時保留宏叢集信息

**🔹 解析步驟**

**🔸 1. 定義數據結構**

* **macro\_cell\_connect（map 映射）**
  + 這是一個映射表，記錄了**宏與單元之間的關係**。
  + 每個 cell\_id（單元 ID）會有一個對應的**是否與宏相連的標記（first）** 和 **對應的宏 ID（second）**。

**🔸 2. 遍歷所有連接（第 2 行）**

* **迴圈遍歷每一條連接**（代表 netlist 中的網線）。

**🔸 3. 連接 Macro 與 Cell（第 3-7 行）**

📌 **條件判斷**

* **如果 src（源節點）是宏，sink（終點）是單元**（第 3 行）：
  1. **建立 macro\_src 到 cell\_sink 的連接**（第 4 行）。
  2. **記錄該 sink（cell）的宏關聯資訊**：
     + macro\_cell\_connect[sink.id].first = 1 → 表示該 sink 已與某個宏連接（第 5 行）。
     + macro\_cell\_connect[sink.id].second = src\_id → 記錄與該 cell 相連的宏 ID（第 6 行）。

📌 **作用**

* 這部分確保 **每個 cell 是否與某個 macro 連接**，為後續 cell-cell 連接做準備。

**🔸 4. 連接 Cell 與 Cell（第 8-17 行）**

📌 **條件判斷**

* **如果 src 和 sink 都是 cell**（第 8 行），則進行 cell-cell 連接。
* **檢查 src 是否與某個 macro 連接過**（第 9 行）：
  + macro\_cell\_connect[src\_id].first == 1 → **這表示 src 曾經與某個 macro 連接過**（第 10 行）。
  + 這是為了確保 cell-cell 連接時能夠保留 macro 影響的數據流。

📌 **處理方式**

1. **如果 src\_id 和 sink\_id 不是同一個 cell**（第 11 行）：
   * 取出 macro\_id = macro\_cell\_connect[src\_id].second（第 12 行），獲取與 src 相連的宏 ID。
   * **建立 macro\_id 到 cell\_sink 的連接**（第 13 行）。
   * **建立 cell\_src 到 cell\_sink 的連接**（第 14 行）。

📌 **作用**

* 這確保了：
  1. **如果 src 曾經與某個 macro 連接，則將該 macro 影響擴展到 sink**。
  2. **除了 cell-cell 連接，還保持 macro-cell-cell 的影響關係**，這有助於更好的數據流優化。

**🔸 5. 返回提取的連接（第 19 行）**

* 最終，**返回所有宏叢集-單元叢集-單元叢集的連接結果**。

**🔹 總結**

✅ **這個演算法的作用是提取宏與單元之間的數據流連接，並確保這些關係在 cell-cell 連接中得到保留。**  
✅ **先建立 macro-cell 連接，再擴展到 cell-cell 連接，並記錄 cell 是否與 macro 相關，從而維持數據流影響。**  
✅ **這對於後續的宏擺放（macro placement）和時序優化（timing optimization）至關重要！** 🚀

在 **IC 佈局與繞線（Placement & Routing, P&R）** 領域，**強化學習（Reinforcement Learning, RL）** 和 **深度學習（Deep Learning, DL）** 已經被廣泛研究來自動化宏觀模塊（Macro）或標準單元（Standard Cell）的放置。然而，這些方法仍然面臨 **Legalization Problem（合法化問題）**、**高計算成本（Huge Computation）** 和 **數據集難以獲取（Dataset is not easy to access）** 等挑戰。以下是詳細解釋這些問題的原因：

**1️Legalization Problem（合法化問題）**

**🔹 問題描述**

**Legalization（合法化）** 指的是在放置元件後，確保它們滿足**設計規則（Design Rules）**，例如：

* **沒有重疊（No Overlap）**
* **對齊標準單元（Standard Cell Alignment）**
* **滿足設計約束（e.g., spacing, power grid alignment, DRC rules）**

**RL/DL 的主要挑戰是：**

1. **學習到的佈局可能會違反設計規則**
   * RL/DL 的學習目標主要是**優化佈局品質（QoR）**，如時序（Timing）、面積（Area）、功耗（Power），但它們可能無法直接學習到**IC 設計規則（DRC, Design Rule Constraints）**。
   * 這可能導致**元件重疊**，需要額外的後處理來調整。
2. **後處理可能破壞 RL/DL 產生的佈局**
   * 為了修復非法放置的元件，**需要對元件進行移動**，但這可能會影響 RL/DL 設計的**時序、擁塞、功耗**等優化結果。

**🔹 解決方案**

✅ **在 RL Reward Function 中加入 Legalization Cost**

* 設計 RL 時，可以讓 Agent 學習避免非法放置，例如對**重疊、違反設計規則的行為進行懲罰**。

✅ **結合傳統的佈局工具**

* 例如，使用 RL 進行**初步的 Macro 佈局**，然後讓傳統的 EDA 工具（如 Cadence Innovus, Synopsys ICC2）進行 **legalization**，確保合法性。

**2️ Huge Computation（巨大的計算成本）**

**🔹 問題描述**

RL 和 DL 在 IC 設計中的應用需要大量的計算資源，主要原因如下：

1. **設計空間巨大**
   * **佈局搜索空間（Search Space）極大**，即便只有 20 個 Macro，可能的放置方式就已經超過 $10^{18}$ 種。
   * RL 需要反覆嘗試不同的佈局，導致計算量激增。
2. **計算一次佈局 QoR 需要運行完整的 P&R**
   * RL 的 reward 計算通常來自 P&R（包括擁塞分析、時序分析），而這些步驟本身就**非常耗時**。
   * **一個完整的 P&R 流程可能需要數小時到數天**，如果 RL 需要多次試驗，訓練時間會變得**極為昂貴**。
3. **深度學習模型的推理開銷大**
   * DL 需要**處理高維特徵（e.g., Netlist Graph, Placement Map）**，通常使用 CNN、GNN（Graph Neural Networks）等模型。
   * 訓練和推理這些模型需要大量的 GPU/TPU 資源。

**🔹 解決方案**

✅ **使用預先訓練的模型**

* **先用小型電路訓練 DL/RL，然後轉移學習（Transfer Learning）到大型電路**，減少計算需求。

✅ **混合傳統方法與 ML**

* 例如，使用 **傳統啟發式方法（如 Simulated Annealing, Analytical Placement）** 來提供初始解，然後用 RL 進行微調，以減少搜索時間。

✅ **加速 QoR 評估**

* **使用快速擁塞估計**（如基於 GCN 的繞線擁塞預測）來替代完整的 P&R，減少計算成本。

**3️ Dataset is Not Easy to Access（數據集難以獲取）**

**🔹 問題描述**

IC 佈局需要大量的數據來訓練 RL/DL 模型，但：

1. **真實設計數據受 NDA 限制**
   * **晶片設計公司（如 Intel、TSMC、NVIDIA）** 的真實佈局數據是機密的，學術界很難獲得。
2. **現有的開源數據集有限**
   * 現有的開源 IC 佈局數據集（如 ISPD、DAC 比賽數據）規模有限，且不一定適用於先進製程（如 5nm、3nm）。
3. **標註數據（Label）很難獲取**
   * 訓練 RL/DL 需要大量的標註數據，例如「最佳佈局」、「優化後的 QoR」，但這些數據需要專業的 EDA 工具運行 P&R 才能得到，成本高昂。

**🔹 解決方案**

✅ **利用合成數據（Synthetic Data）**

* 透過 **程序生成（Procedural Generation）**，根據標準元件庫（Standard Cell Library）隨機生成合成電路，擴展數據集。

✅ **遷移學習（Transfer Learning）**

* **用小型開源電路訓練模型，然後遷移到真實電路**，降低對大量數據的需求。

✅ **與企業合作**

* 研究機構可以**與 EDA 公司合作（如 Synopsys, Cadence）**，獲取部分非機密的數據集。

**🔹 總結**

| **挑戰** | **主要原因** | **可能解決方案** |
| --- | --- | --- |
| **Legalization Problem** | RL/DL 可能產生非法佈局，違反設計規則 | 在 RL reward function 加入合法性懲罰，或結合傳統佈局工具 |
| **Huge Computation** | 佈局搜索空間大，QoR 評估需要完整 P&R，DL 訓練成本高 | 使用預訓練模型，混合傳統方法與 ML，使用快速 QoR 預測 |
| **Dataset is Not Easy to Access** | 真實晶片設計受 NDA 限制，標註數據難獲取 | 生成合成數據，遷移學習，與企業合作 |

在先進製程（如 5nm、3nm）下，EDA 工具的複雜度不斷增加，這些問題變得更加嚴峻。因此，**如何將 ML 與傳統佈局方法結合**，仍然是目前研究的核心方向！ 🚀

A design usually contains a sequential circuit, which is composed of two elements: one is control unit and the other is datapath.