Programming Assignment #1

B11901029 電機三 李致頡

Algorithm

My implementation is similar with the Fiduccia–Mattheyses heuristic [1], but I have incorporated optimizations from several studies to further reduce the cut size:

1. CLIP: As described in [2], this clustering-based optimization for iterative partitioning utilizes an "updated gain" for selecting the move with the maximum gain, where "updated gain" $=$ actual gain $-$ initial gain.
2. Toward tie breaking for different partitions: According to the experiments in [3], when cells from both partitions are movable and the maximum gains in the two bucket lists are identical, it is preferable to select the cell whose movement direction is consistent with the previous move.
3. LIFO tie breaking in the same partition: According to the experiments in [4], within the maximum gain bucket, employing a last-in-first-out (LIFO) selection strategy yields better performance than using random selection or a first-in-first-out (FIFO) approach.

These optimization techniques are straightforward to implement; however, compared to the original Fiduccia–Mattheyses heuristic, they significantly enhance the reduction of the cut size.

Data Structure

To implement the bucket list of doubly linked lists as described in [1], I use an STL `vector` for $O(1)$ bucket access, where each element is a `Node*` pointer pointing to the first element of its respective doubly linked list. In addition, a MAXGAIN pointer is maintained, which points to the first element of the doubly linked list corresponding to the bucket with the highest gain. There are two key considerations for ensuring the implementation aligns with the algorithm mentioned above:

1. Gain range: Since the "updated gain" $=$ actual gain $-$ initial gain, its range becomes $[-2P_{\max}, 2P_{\max}]$, differing from that of the original algorithm. Moreover, because the STL `vector` only supports non-negative index, an offset of $2P_{\max}$ must be added when accessing the vector.
2. LIFO tie breaking: Given that the MAXGAIN pointer always points to the first element of the doubly linked list, when inserting a new cell into a bucket, it is imperative to insert at the front. This approach ensures that the algorithm consistently selects the most recently inserted cell for movement.

Both the insertion and deletion operations in a doubly linked list are $O(1)$. Following the method proposed in the original paper [1], the MAXGAIN pointer is updated as follows:

1. Deletion: When the cell pointed by the MAXGAIN pointer is removed, check if there are any remaining elements in the same bucket. If elements exist, the pointer is updated to the next element in the list. Otherwise, the gain index is decremented until a non-empty bucket is located. Since the total number of indeices is $4P_{\max} + 1$, this operation is $O(P_{\max}) \in O(P)$.

```cpp
void Partitioner::removeBucketList(Cell* cell) {
    Node* cell_node = cell->getNode();
    // The cell to be removed is in the middle of the bucket list
    if (cell_node->getPrev()) {
        cell_node->getPrev()->setNext(cell_node->getNext());
        // The cell to be removed is at the front of the bucket list
    } else {
        bool part                        = cell->getPart();
        int clip_gain                    = cell->getCLIPGain();
        auto& blist_part                 = blist_[part];
        blist_part[getBlistId(clip_gain)] = cell_node->getNext();
        // Update Max Gain pointer
        if (max_clip_gain_cell_[part] == cell_node) {
            max_clip_gain_cell_[part] = nullptr;
            for (auto it = blist_part.rbegin() + (blist_part.size() - 1 - getBlistId(clip_gain));
                 it != blist_part.rend(); ++it) {
                if (*it) {
                    max_clip_gain_cell_[part] = *it;
                    break;
                }
            }
        }
    }
    if (cell_node->getNext()) cell_node->getNext()->setPrev(cell_node->getPrev());
    cell_node->setPrev(nullptr);
    cell_node->setNext(nullptr);
}
```

2. Insertion: Check if the gain of the newly inserted cell exceeds that of the current MAXGAIN cell. If it does, the MAXGAIN pointer is updated to the new cell, which is $O(1)$.

Findings

While debugging my program, I observed that the test case `input_0.dat` exhibited unexpected behavior. Upon further investigation, I discovered that the Fiduccia–Mattheyses heuristic presupposes the absence of single-pin nets, that is, nets that connect to only one cell. However, `input_0.dat` contains a considerable number of single-pin nets. Consequently, I decided to disregard all single-pin nets, as they do not influence the cut size.

Result

Run my program on EDA Union server edaU13.

Input file	Cell #	Net #	Cut size	Runtime (s)	Score
input_0.dat	150750	166998	742	2.12	9.19526
input_1.dat	3000	5000	1198	0.06	8.61224
input_2.dat	7000	10000	2075	0.10	8.67885
input_3.dat	66666	88888	26690	2.81	8.34467
input_4.dat	150750	166998	42524	19.62	8.47526
input_5.dat	382489	483599	136697	163.72	8.373256

Reference

[1] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in Papers on Twenty-five years of electronic design automation, 1988, pp. 241–247.

[2] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in Proceedings of International Conference on Computer Aided Design, 1996, pp. 194–200.

[3] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Design and implementation of the fiduccia-mattheyses heuristic for vlsi netlist partitioning," in Algorithm Engineering and Experimentation: International Workshop ALENEX'99 Baltimore, MD, USA, January 15–16, 1999 Selected Papers 1, 1999, pp. 182–198.

[4] L. W. Hagen, D.-H. Huang, and A. B. Kahng, "On implementation choices for iterative improvement partitioning algorithms," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 10, pp. 1199–1205, 1997.