### Programming Assignment #2 - Soft Block Floorplanning

B110300XX 陳○○

#### Introduction

To optimize floorplanning, I decided to adopt *Sequence Pair* in this project, due to its advantage over *Slicing Tree* and *B\*-tree*. To clarify, tree-based representations have some inherent constraints that may reduce the chance of finding a better solution. That makes it a less favorable type of representation, and by contrast, *Sequence Pair* is a fully topological representation, making it the representation of choice.

*Longest Common Subsequence (LCS) Algorithm* [1] along with a generic simulated annealing algorithm is implemented in this program, efficiently carrying out floorplan optimization.

#### The data structures and algorithms

Each macro is stored in one *Macro* and resides in *Macros* of type *vector<Macro>*.

class Macro {

public:

    string macroName;

    int w, h;

    int x1, y1, x2, y2;

*// int x2 = PINF, y2 = PINF;*

    int cx, cy;

    void findCenter();

    void swapwh();

    void findx2y2();

};

The sequence pair (Γ+, Γ-) is stored in two integer vectors, the *seqPos* and *seqNeg* respectively. An *unordered\_map* named *match* maps the ID of a *Macro* (it’s the order of the macro inputs by default) to its position in the *seqNeg*, which is used by the Fast LCS algorithm.

The *perturb* function determines the action for each simulated annealing (SA) iteration. It focuses on single and double swaps as they outperform inversions (removed for efficiency - see page 19, [2])

The entire *calcCoords* is basically the Fast LCS algorithm mentioned in [1], but it can cause time wastage because the coordinates of the blocks are not needed during the SA process. Therefore, I made *calcCost* to slightly improve the performance.

typedef struct SP {

    int v;

    int max\_w, max\_h;

    vector<int> seqPos; *// Gamma+*

    vector<int> seqNeg; *// Gamma-*

    unordered\_map<int, int> match;

    SP();

    SP(const vector<Macro>& *M*);

    void random\_P(); *// Random Permutation*

    int getCost();

    int perturb(); *// SA*

    void translate(); *// To be called in perturb and constructor*

    void calcCoords(vector<Macro>& *M*); *// LCS*

    void calcCost(const vector<Macro>& *M*); *// LCS*

} SP;

The following code is one part of the SA process, where it accepts a worse solution with probability 𝑒−Δ/𝑇. Specifically, the *T* here (the variable *t\_cool*) is the time elapsed since the last solution update (last improvement time) and the *Δ* here is the cost (in fact it’s cost1/2) difference between the current floorplan and the optimal one.

        } else { *// if no improvement (delta >= 0)*

            if (dis(gen) < exp(-delta / t\_cool.count())) {

                tempSP = currSP;

*// cout << "\nAn Uphill move" << endl;*

            }

            currSP = tempSP; *// Recover to the last state*

        }

After completing SA, a random rotation algorithm (not shown here) begins to identify potentially lower-cost solutions. And after that, the program terminates.

#### Reference:

1. [343647.343713 (acm.org)](https://dl.acm.org/doi/pdf/10.1145/343647.343713)
2. S.-Y. Fang (2024). Floorplanning x Placement