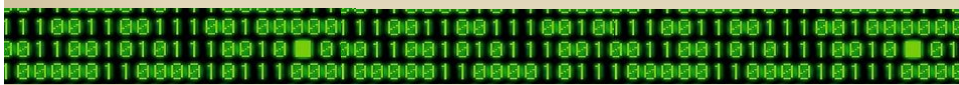



EE6094
CAD for VLSI Design



PA3 Analog Placement Review


Spring 2024
Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andygchen@ee.ncu.edu.tw
Slides Credit: TA Yu-Heng Tsao



2024/6/4


Andy Yu-Guang Chen

1



Outline



- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ Pack
- ◆ Result



2024/6/4


Andy Yu-Guang Chen

2





Outline

- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ Pack
- ◆ Result




2024/6/4 Andy Yu-Guang Chen 3



Workflow

Parse → Placement → Output

```
graph TD; Placement --- SP[Sequence Pair]; Placement --- BT[B*-tree]; Placement --- O[Others];
```



2024/6/4 Andy Yu-Guang Chen 4



Naïve Placement

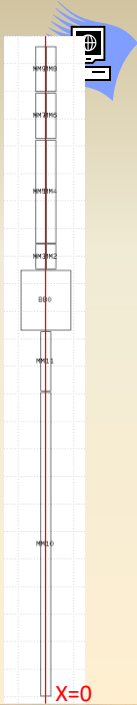
```
int main(int argc, char **argv) {
    using namespace std::chrono_literals;

    if (argc != 6) {
        return 1;
    }

    const double expected_aspect_ratio = stod(argv[1]);
    const std::string netlist_file_path = argv[2];
    const std::string symmetry_file_path = argv[3];
    const std::string block_file_path = argv[4];
    const std::string output_file_path = argv[5];

    HStarTree tree;
    if (!tree.load(netlist_file_path, symmetry_file_path, block_file_path)) {
        std::cerr << "Failed to initialize the tree" << std::endl;
        return 1;
    }
    tree.NaivePlacement();
    tree.Dump(output_file_path);
    return 0;
}
```

```
void HStarTree::NaivePlacement() {
    double x0 = 0.0, x1 = 0.0, y = 0.0;
    for (auto &unit : units_) {
        BuildingBlock *const representative = unit->right_half;
        if (unit->right_half == unit->left_half) /* self-symmetric */ {
            representative->MoveCenterTo(0.0);
            representative->MoveMinTo(y);
            x0 = std::min(x0, representative->GetMinX());
            x1 = std::max(x1, representative->GetMaxX());
        }
        else /* symmetry-pair */ {
            unit->right_half->MoveMinTo(0.0);
            unit->right_half->MoveMinTo(y);
            unit->left_half->MoveMaxTo(0.0);
            unit->left_half->MoveMinTo(y);
            x0 = std::min(x0, unit->left_half->GetMinX());
            x1 = std::max(x1, unit->right_half->GetMaxX());
        }
        y += representative->GetHeight();
    }
    chip_.SetMinX(x0);
    chip_.SetMaxX(x1);
    chip_.SetMinY(0.0);
    chip_.SetMaxY(y);
}
```



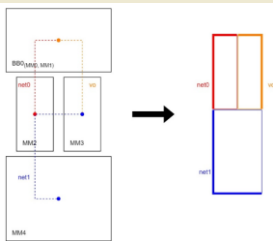
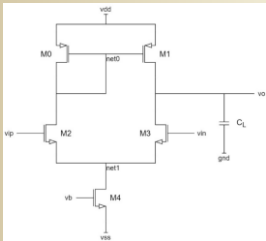
2024/6/4

Andy Yu-Guang Chen

5



Calculate Total HPWL



```
orig_net_map -> {
    "net0" -> {MM0, MM1, MM2},
    "vdd1" -> {MM0, MM1},
    "vo" -> {MM1, MM3, CL},
    ...
}

processed_net_map -> {
    "net0" -> {BB0, MM2},
    "vdd1" -> {BB0},
    "vo" -> {BB0, MM3},
    ...
}
```



2024/6/4

Andy Yu-Guang Chen

6



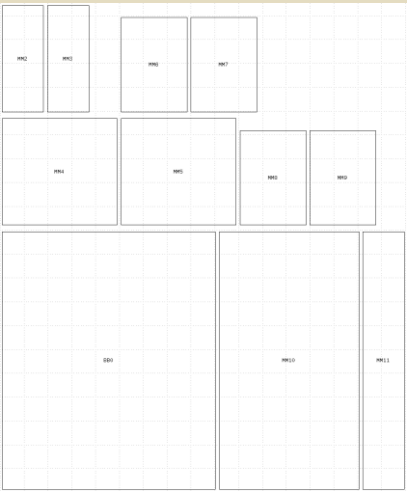
Calculate Total HPWL

```
double HBStarTree::GetTotalHPWL() const {
    double total_hpwl = 0.0;
    double x0, y0, x1, y1, xc, yc, w, h;
    for (auto &[_ , blocks] : nets_) {
        x0 = x1 = blocks[0]->GetCenterX();
        y0 = y1 = blocks[0]->GetCenterY();
        for (std::size_t i = 1; i < blocks.size(); ++i) {
            xc = blocks[i]->GetCenterX();
            yc = blocks[i]->GetCenterY();
            x0 = std::min(x0, xc);
            x1 = std::max(x1, xc);
            y0 = std::min(y0, yc);
            y1 = std::max(y1, yc);
        }
        w = x1 - x0;
        h = y1 - y0;
        total_hpwl += w + h;
    }
    return total_hpwl;
}
```



Case 1

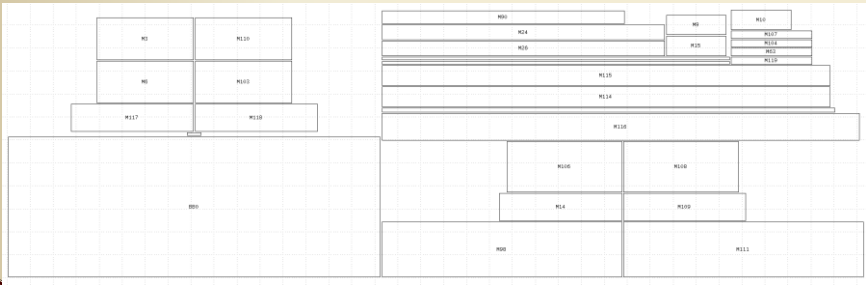
- ◆ Worst cost = 7491.386822
- ◆ Best cost = 164.319514





Case 2

- ◆ Worst cost = 2083624.674
- ◆ Best cost = 3923.390984



2024/6/4

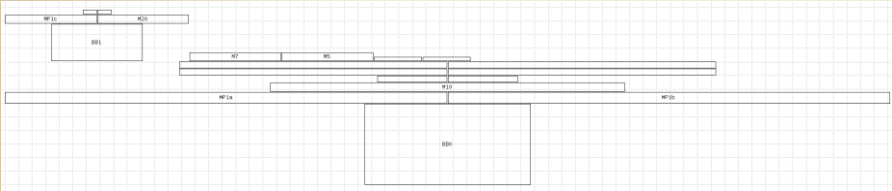
Andy Yu-Guang Chen

9



Case 3

- ◆ Worst cost = 99871.8212
- ◆ Best cost = 6535.670481



2024/6/4

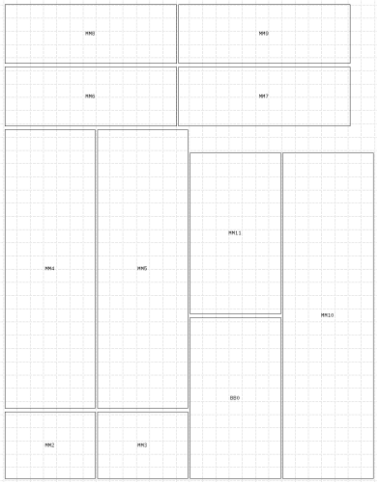
Andy Yu-Guang Chen

10



Case 4

- ◆ Worst cost = 316205.4015
- ◆ Best cost = 397.80213



2024/6/4

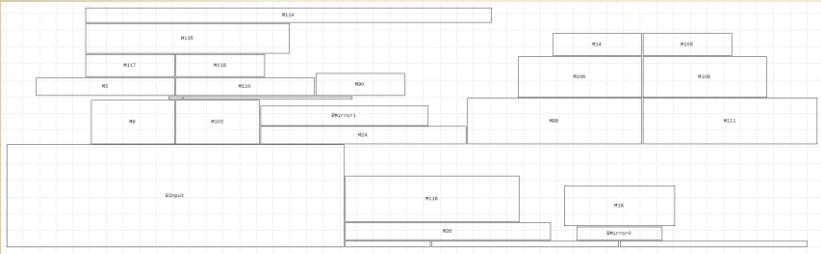
Andy Yu-Guang Chen

11



Case 5

- ◆ Worst cost = 1312004.893
- ◆ Best cost = 6490.263741



2024/6/4

Andy Yu-Guang Chen

12



Outline



- ◆ Overview
- ◆ Data Structure
 - Basic Components
 - Hierarchical B*-tree
 - Contour List
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ Pack
- ◆ Result



2024/6/4

Andy Yu-Guang Chen

13



Rectangle



```
class Rectangle {
public:
    Rectangle(double x = 0.0, double y = 0.0, double w = 0.0, double h = 0.0);

    double GetMinX() const;
    double GetMaxX() const;
    double GetMinY() const;
    double GetMaxY() const;
    double GetWidth() const;
    double GetHeight() const;
    double GetCenterX() const;
    double GetCenterY() const;

    // Reshape the rectangle
    void SetMinX(double);
    void SetMinY(double);
    void SetMaxX(double);
    void SetMaxY(double);
    void SetWidth(double);
    void SetHeight(double);

    // Move the rectangle without reshape
    void MoveMinXTo(double);
    void MoveMinYTo(double);
    void MoveMaxXTo(double);
    void MoveMaxYTo(double);
    void MoveCenterXTo(double);
    void MoveCenterYTo(double);

private:
    double x0_ = 0.0;
    double y0_ = 0.0;
    double x1_ = 0.0;
    double y1_ = 0.0;
}; // class Rectangle
```



2024/6/4

14



Pattern & BuildingBlock



```
struct Pattern {
    double width, height;

    // column multiple and row multiple
    int col_mul, row_mul;

    constexpr bool operator==(Pattern const&) const = default;
    constexpr auto operator<=>(const Pattern &rhs) const = default;
}; // struct Pattern

struct BuildingBlock : public Rectangle {
    // The name of the building block
    std::string name;

    // The names of the merging devices
    std::vector<std::string> merge_device_names;

    // The shape of the building block
    std::vector<Pattern> patterns;

    BuildingBlock() = default;
    BuildingBlock(std::string line);
    int GetColumnMultiple() const;
    int GetRowMultiple() const;
}; // struct BuildingBlock
```



2024/6/4

Andy Yu-Guang Chen

15



Outline



- ◆ Placement With Optimization
- ◆ Data Structure
 - Basic Components
 - Hierarchical B*-tree
 - Contour List
- ◆ Overview
- ◆ Perturb
- ◆ Pack
- ◆ Appendix
- ◆ Result



2024/6/4

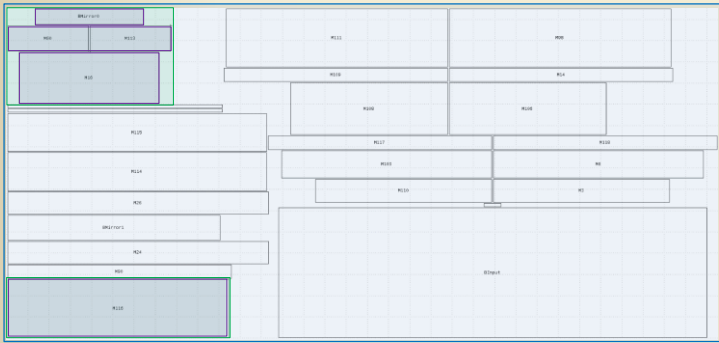
Andy Yu-Guang Chen

16



Tree Hierarchy

- ◆ Tree Hierarchy
 - *HBStarTree* -> *ASFBStarTree* -> *SymmetryUnit*
- ◆ Treat a *BuildingBlock* without symmetry constraints as an one-*SymmetryUnit* *ASFBStarTree*



2024/6/4

Andy Yu-Guang Chen

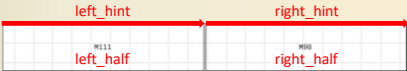
17



SymmetryUnit

- ◆ Represents a self-symmetric or a symmetry pair


```
struct SymmetryUnit {  
    // The left and right halves of the symmetry module  
    BuildingBlock *right_half(), *left_half();  
  
    SymmetryUnit *parent();  
    SymmetryUnit *lchild();  
    SymmetryUnit *rchild();  
  
    // Store the starting points of the left and right halves in the ContourList,  
    // which allows for amortized O(1) time complexity to find the feasible Y coordinates of the children  
    ContourCIter left_hint(), right_hint();  
}; // struct SymmetryUnit
```




2024/6/4

Andy Yu-Guang Chen

18





ASFBStarTree

◆ Represents a symmetry island

```
struct ASFBStarTree {
    std::string name();
    std::vector<SymmetryUnit *> units();
    Rectangle bounding_box();

    // The root of internal ASF B*-tree (symmetry island)
    SymmetryUnit *root();

    // 'begin' is an iterator to the beginning of the top contours contributed by this ASFBStarTree
    // 'end' is an iterator to the end of the top contours
    ContourCiter begin(), end();

    ASFBStarTree *parent();
    ASFBStarTree *lchild();
    ASFBStarTree *rchild();
}; // struct ASFBStarTree
```

rchild

↓


begin

↓

lchild

↓

end



2024/6/4

Andy Yu-Guang Chen

19





HBStarTree

◆ A normal B*-tree, each node is an *ASFBStarTree*

```
// Hierarchical B*-tree
class HBStarTree {
public:
    /* ... */
private:


    std::default_random_engine rng_{std::random_device{}()};
    std::vector<ASFBStarTree *> asfbtrees_{};
    std::vector<SymmetryUnit *> units_{};
    std::vector<BuildingBlock *> blocks_{};
    std::vector<std::pair<std::string, std::vector<BuildingBlock *>>> nets_{};
    Rectangle chip_{0.0, 0.0, 0.0, 0.0};
    ASFBStarTree *root_{};
}; // class BStarTree
```



2024/6/4


Andy Yu-Guang Chen

20



Outline


- ◆ Overview
- ◆ Placement With Optimization
- ◆ Data Structure
 - Basic Components
 - Hierarchical B*-tree
 - Contour List
- ◆ Perturb
- ◆ Pack
- ◆ Result



2024/6/4

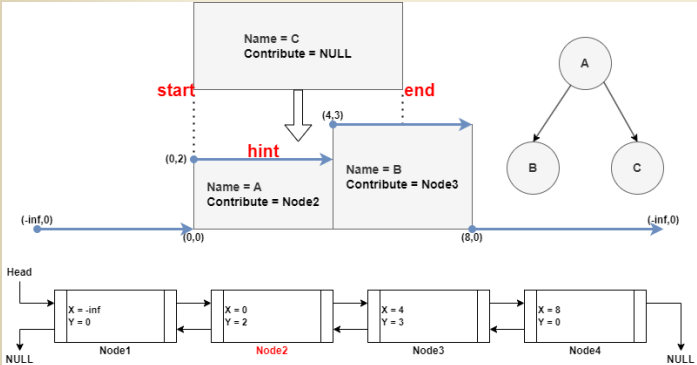
Andy Yu-Guang Chen


21



The concept of ContourList

- ◆ Use **pointer or iterator** to connect *ContourList::Node* and *BuildingBlock* to efficiently **query** & **update** *ContourList* (amortized $O(1)$)





2024/6/4

Andy Yu-Guang Chen

22



ContourList::Node



◆ Horizontal contour list

- X -> Node::start
- Y -> Node::offset

```
// Description
// -----
// Structure to represent a contour interval
//
// Horizontal Example
// -----
// - The Node's 'start' represents the left endpoint of an interval on the x-axis.
// - The subsequent Node's 'start' corresponds to the right endpoint of the same x-axis interval.
// - Additionally, the 'offset' value is associated with the y-axis.
struct Node {
    using Cmp = std::function<bool(const Node &, const Node &)>;

    double start; // Starting point of the contour interval [start, next node start)
    double offset; // Offset associated with the contour interval

    static bool StartLess(const Node &a, const Node &b){
        return IsLess(a.start, b.start);
    }
    static bool OffsetLess(const Node &a, const Node &b){
        return IsLess(a.offset, b.offset);
    }
    bool operator==(const Node &rhs) const {
        return IsEqual(start, rhs.start) && IsEqual(offset, rhs.offset);
    }
};
```



2024/6/4

Andy Yu-Guang Chen

23



ContourList



◆ Passing in a search or insertion starting point (**hint**), achieving time complexity of **amortized O(1)** and full Packing O(N)

```
// Represents a strictly increasing list of non-overlapping contour intervals.
// This class behaves similarly to managing the skyline contours in a tetris game.
class ContourList {
public:
    // Insert a new contour interval with a hint iterator and remove completely covered intervals.
    // (potentially more efficient for specific use cases)
    std::pair<Contiterator, Contiterator> InsertHint(Contiterator hint, double start, double end, double offset);

    // Find the maximum element in the given interval with a hint
    Contiterator MaxElementHint(Contiterator hint, double start, double end, Node::Cmp cmp);

    // Reset the list to an initial state
    void Reset();

    /* ... */

private:
    // Description
    // -----
    // Finds the range of nodes covering a specified interval (internal helper).
    // (potentially more efficient for specific use cases)
    //
    // Returns
    // -----
    // - 'first': An iterator pointing to the node with a 'start' value less than or equal to the given 'start', where 'start' is maximized
    // - 'second': An iterator pointing to the node with a 'start' value greater than or equal to the given 'end'
    std::pair<Contiterator, Contiterator> CoverRangeHint(Contiterator hint, double start, double end);

    // Internal list storing the contour intervals in increasing order
    std::list<Node> base_;

    // Recycle bin for unused nodes
    std::list<Node> recycle_;

    // Maximum number of recycled nodes
    std::size_t max_recycled_;
}; // class ContourList
```



2024/6/4

Andy Yu-Guang Chen

24

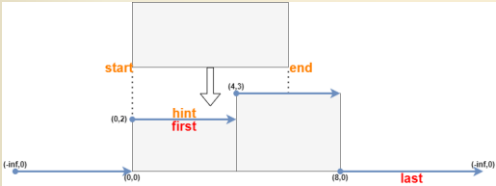


CoverRangeHint

```
std::pair<ContourList::ConstIterator, ContourList::ConstIterator> ContourList::CoverRangeHint(ConstIterator hint, double start, double end) {
    hint = (hint == ConstIterator()) ? hint : base_end();
    while (IsGreater(hint->start, start)) {
        --hint;
    }

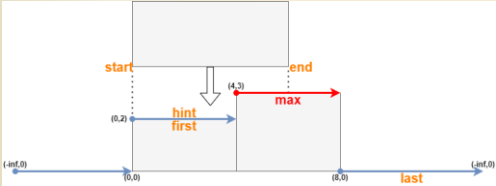
    auto first = hint;
    while (std::next(first) != base_end()) {
        if (IsLessEqual(first->start, start) && IsLess(start, std::next(first)->start)) {
            break;
        }
        ++first;
    }

    auto last = first;
    while (last != base_end()) {
        if (IsGreaterEqual(last->start, end)) {
            break;
        }
        ++last;
    }
    return (first, last);
}
```



MaxElementHint

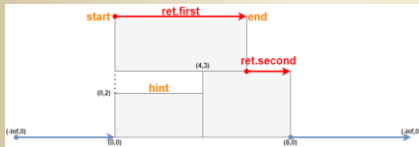
```
ContourList::Iterator ContourList::MaxElementHint(Iterator hint, double start, double end, Node::Cmp cmp) {
    auto [first, last] = CoverRangeHint(hint, start, end);
    return std::max_element(first, last, cmp);
};
```



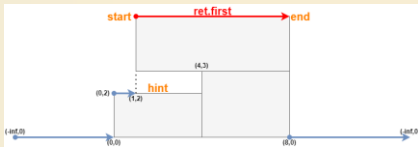


InsertHint

```
std::pair<ContourList::ConstIterator, ContourList::ConstIterator> ContourList::InsertHint(ConstIterator hint, double start, double end, double offset) {  
    // Update the list. Emplace a new node for any update.  
    std::pair<ConstIterator, ConstIterator> ret{};  
    auto [first, last] = CoverRangeHint(hint, start, end);  
    double prev_last_offset = std::prev(last)->offset;  
  
    // Update left endpoint.  
    if (!std::less(first->start, start)) {  
        ret.first = EmplaceWithRecycle(std::next(first), start, offset);  
    }  
    else /* first->start == start */ {  
        ret.first = EmplaceWithRecycle(first, start, offset);  
    }  
  
    // Update right endpoint.  
    if (!std::greater(last->start, end)) {  
        ret.second = last = EmplaceWithRecycle(last, end, prev_last_offset);  
    }  
  
    // Erase the covered contour nodes.  
    EraseWithRecycle(std::next(ret.first), last);  
    return ret;  
}
```



Case 1



Case 2



2024/6/4

Andy Yu-Guang Chen

27

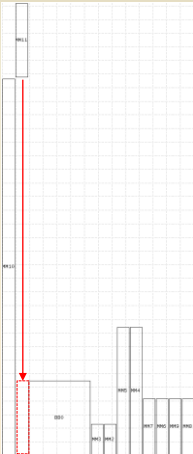


Precision Problem

- ◆ Set the precision of the double data type to prevent precision errors

Replace "a==b"

```
bool IsEqual(double a, double b) {  
    return std::fabs(a - b) < 1e-6;  
}  
bool IsLess(double a, double b) {  
    return a < b && !IsEqual(a, b);  
}  
bool IsGreater(double a, double b) {  
    return a > b && !IsEqual(a, b);  
}  
bool IsLessEqual(double a, double b) {  
    return IsLess(a, b) || IsEqual(a, b);  
}  
bool IsGreaterEqual(double a, double b) {  
    return IsGreater(a, b) || IsEqual(a, b);  
}
```



case1.output.gds



2024/6/4

Andy Yu-Guang Chen

28



Outline



- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ Pack
- ◆ Result



2024/6/4

Andy Yu-Guang Chen

29



Placement With Optimization



```
int main(int argc, char **argv) {
    using namespace std::chrono_literals;

    if (argc != 6) {
        return 1;
    }

    const double expected_aspect_ratio = std::stod(argv[1]);
    const std::string netlist_file_path = argv[2];
    const std::string symmetry_file_path = argv[3];
    const std::string block_file_path = argv[4];
    const std::string output_file_path = argv[5];

    HBStarTree tree;
    if (!tree.Load(netlist_file_path, symmetry_file_path, block_file_path)) {
        std::cerr << "Failed to initialize the tree" << std::endl;
        return 1;
    }

    auto pa3_cost = CostFunction(expected_aspect_ratio);
    auto stop_10_min = Timer{10min};
    auto greedy = [](double a, double b) { return a < b; };
    tree.BuildTree();
    tree.Pack();
    tree.Optimize(pa3_cost, stop_10_min, greedy);

    tree.Dump(output_file_path);

    return 0;
}
```



2024/6/4

Andy Yu-Guang Chen

30

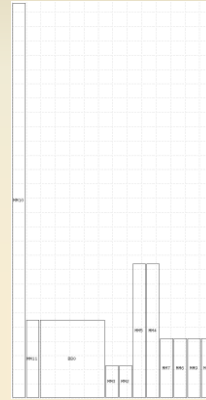


BuildTree



```
void HBStarTree::BuildTree() {
    // Construct the left list B*-tree
    std::shuffle(asfbtrees.begin(), asfbtrees.end(), rng_);
    root = asfbtrees.front();
    asfbtrees.front()->parent = nullptr;
    for (std::size_t i = 0; i < asfbtrees.size() - 1; ++i) {
        asfbtrees[i + 1]->parent = asfbtrees[i];
        asfbtrees[i]->lchild = asfbtrees[i + 1];
        asfbtrees[i]->rchild = nullptr;
    }
    asfbtrees.back()->lchild = nullptr;
    asfbtrees.back()->rchild = nullptr;

    // Construct the right list ASF B*-tree for each unit due to symmetry constraints
    for (auto &asfbtree : asfbtrees_) {
        auto &units = asfbtree->units;
        std::shuffle(units.begin(), units.end(), rng_);
        asfbtree->root = units.front();
        units.front()->parent = nullptr;
        for (std::size_t i = 0; i < units.size() - 1; ++i) {
            units[i + 1]->parent = units[i];
            units[i]->lchild = nullptr;
            units[i]->rchild = units[i + 1];
        }
        units.back()->lchild = nullptr;
        units.back()->rchild = nullptr;
    }
}
```



2024/6/4

Andy Yu-Guang Chen

31



Optimize



```
template <class CostFn, class StopFn, class AcceptStrategy>
void HBStarTree::Optimize(CostFn cost_func, StopFn stop_func, AcceptStrategy accept_func) {
    constexpr int kRejectLimit = 10000000;

    Placement placement;
    Perturbator perturb(this);
    Packon pack(this);
    double cost = cost_func(this);
    double new_cost;
    int reject_count = 0;

    placement = *this;
    while (!stop_func(this) && reject_count < kRejectLimit) {
        perturb();
        pack();
        new_cost = cost_func(this);
        if (accept_func(new_cost, cost)) {
            std::cout << std::format("cost={:.6f} new_cost={:.6f}", cost, new_cost) << std::endl;
            cost = new_cost;
            placement = *this;
            reject_count = 0;
        }
        else {
            *this = placement;
            ++reject_count;
        }
    }
}
```



2024/6/4

Andy Yu-Guang Chen

32



Outline



- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ **Perturb**
- ◆ Pack
- ◆ Result



2024/6/4

Andy Yu-Guang Chen

33



Perturbator



- ◆ 6 perturb operations
- ◆ Perform multiple operations at a time to escape the local optima

```
class Perturbator {
public:
    Perturbator(HBStarTree *);
    void operator()();

private:
    void RandomResizeSymmetryUnit();
    void RandomSwapASFBStarTree();
    void RandomSwapASFBStarTreeChild();
    void RandomFlipSymmetryUnit();
    void RandomRebuildASFBStarTree();
    void RandomTransplantASFBStarTree();

    void SwapWithoutLink(ASFBStarTree *, ASFBStarTree *);
    void Transplant(ASFBStarTree *, ASFBStarTree *);
    ASFBStarTree *GetRandomChild(ASFBStarTree *);

    std::vector<std::function<void()>> perturbations_();
    std::uniform_int_distribution<int> perturbation_dist_();
    std::discrete_distribution<int> map_perturbations_{0, 1 << 6, 1 << 5, 1 << 4, 1 << 3, 1 << 2, 1 << 1 << 0};
    RandomBoolGenerator rand_bool_gen_();
    std::vector<ASFBStarTree *> sample_trees_(), rebuildable_trees_();
    std::vector<SymmetryUnit *> sample_units_(), resizeable_units_(), flipable_units_();
    std::vector<Pattern> sample_patterns_();
    std::default_random_engine &eng_;
    std::vector<ASFBStarTree *> asfbtrees_;
    std::vector<SymmetryUnit *> &units_;
    ASFBStarTree *root_;
};
```



2024/6/4

34



Perturbator



```

HBStarTree::Perturbator::Perturbator(HBStarTree *hbtree) : rng_(hbtree->rng_), asfbtrees_(hbtree->asfbtrees_), units_(hbtree->units_), root_(hbtree->root_) {
    for (auto &asfbtree : hbtree->asfbtrees_) {
        if (asfbtree->xunits.size() > 1) {
            rebuildable_trees_.push_back(asfbtree);
        }
    }
    for (auto &asfbtree : hbtree->asfbtrees_) {
        for (auto &unit : asfbtree->xunits) {
            if (unit->left_half != unit->right_half) {
                flipable_units_.push_back(unit);
            }
            if (unit->right_half->patterns.size() > 1) {
                resizeable_units_.push_back(unit);
            }
        }
    }

    perturbations_ = {
        std::bind(&Perturbator::RandomResizeSymmetryUnit, this),
        std::bind(&Perturbator::RandomSwapASFBStarTree, this),
        std::bind(&Perturbator::RandomSwapASFBStarTreeChild, this),
        std::bind(&Perturbator::RandomFlipSymmetryUnit, this),
        std::bind(&Perturbator::RandomRebuildASFBStarTree, this),
        std::bind(&Perturbator::RandomTransplantASFBStarTree, this)
    };
    perturbation_dist_.param(std::uniform_int_distribution<int>::param_type(0, perturbations_.size() - 1));
}

void HBStarTree::Perturbator::operator()() {
    int n = max_perturbations_(rng_);
    while (n-- > 0) {
        perturbations_[perturbation_dist_(rng_)]();
    }
}

```



2024/6/4

Andy Yu-Guang Chen

35



Resize



```

void HBStarTree::Perturbator::RandomResizeSymmetryUnit() {
    constexpr std::size_t kSampleUnits = 1;

    sample_units_.clear();
    std::sample(resizeable_units_.begin(), resizeable_units_.end(), std::back_inserter(sample_units_), kSampleUnits, rng_);
    for (auto unit : sample_units_) {
        const auto &patterns = unit->right_half->patterns;
        sample_patterns_.clear();
        std::sample(patterns.begin(), patterns.end(), std::back_inserter(sample_patterns_), 1, rng_);

        const double w = sample_patterns_[0].width;
        unit->right_half->SetWidth(w);
        unit->left_half->SetWidth(w);

        const double h = sample_patterns_[0].height;
        unit->right_half->SetHeight(h);
        unit->left_half->SetHeight(h);
    }
}

```



2024/6/4

Andy Yu-Guang Chen

36

Swap

```
void HBStarTree::Perturbator::RandomSwapASFBStarTree() {
    constexpr int kNumSwap = 1;

    sample_trees_.clear();
    std::sample(asfbtrees_.begin(), asfbtrees_.end(), std::back_inserter(sample_trees_), 2 * kNumSwap, rng_);
    for (std::size_t i = 1; i < sample_trees_.size(); i += 2) {
        SwapWithoutLink(sample_trees_[i], sample_trees_[i - 1]);
    }
}

void HBStarTree::Perturbator::RandomSwapASFBStarTreeChild() {
    constexpr int kNumSwap = 1;

    sample_trees_.clear();
    std::sample(asfbtrees_.begin(), asfbtrees_.end(), std::back_inserter(sample_trees_), kNumSwap, rng_);
    for (auto asfbtree : sample_trees_) {
        // std::cout << "SwapChild: " << asfbtree->name << std::endl;
        std::swap(asfbtree->lchild, asfbtree->rchild);
    }
}

void HBStarTree::Perturbator::RandomFlipSymmetryUnit() {
    constexpr int kNumFlip = 1;

    sample_units_.clear();
    std::sample(flipable_units_.begin(), flipable_units_.end(), std::back_inserter(sample_units_), kNumFlip, rng_);
    for (auto unit : sample_units_) {
        // std::cout << "Flip: " << unit->right_half->name << " <-> " << unit->left_half->name << std::endl;
        std::swap(unit->right_half, unit->left_half);
    }
}
```

2024/6/4

Andy Yu-Guang Chen

37

Rebuild

```
void HBStarTree::Perturbator::RandomRebuildASFBStarTree() {
    constexpr int kNumRebuild = 1;

    sample_trees_.clear();
    std::sample(rebuildable_trees_.begin(), rebuildable_trees_.end(), std::back_inserter(sample_trees_), kNumRebuild, rng_);
    for (auto asfbtree : sample_trees_) {
        // std::cout << "Rebuild: " << asfbtree->name << std::endl;

        // Construct the right list ASF B*-tree for each unit due to symmetry constraints
        auto &units = asfbtree->units;
        std::shuffle(units.begin(), units.end(), rng_);
        asfbtree->root = units.front();
        units.front()->parent = nullptr;
        for (std::size_t i = 0; i < units.size() - 1; ++i) {
            units[i]->lchild = nullptr;
            units[i]->rchild = units[i + 1];
        }
        units.back()->lchild = nullptr;
        units.back()->rchild = nullptr;
    }
}
```

2024/6/4

Andy Yu-Guang Chen

38



Transplant

```
void HBStarTree::Perturbator::Transplant(ASBStarTree *trans, ASBStarTree *to) {
    // std::cout << "Transplant: " << trans->name << " -> " << to->name << std::endl;

    // Remove 'trans'
    ASBStarTree *child = GetHandChild(trans);
    if (child) {
        // Find a leaf and swap 'trans' with it
        ASBStarTree *grandchild = GetHandChild(child);
        while (grandchild) {
            child = grandchild;
            grandchild = GetHandChild(grandchild);
        }
        // std::cout << "Pick child=" << child->name << std::endl;
        SwapWithoutLink(trans, child);
        if (to == child) {
            to = trans;
        }
        trans = child;
    }
    if (trans == trans->parent->lchild) {
        trans->parent->lchild = nullptr;
    }
    else if (trans == trans->parent->rchild) {
        trans->parent->rchild = nullptr;
    }
    else std::runtime_error("unreachable");

    // Insert 'trans'
    auto insert = [&](ASBStarTree *to_child) {
        if (!to_child) {
            to_child = trans;
            return;
        }
        to_child->parent = trans;
        if (rand_bool_gen()) {
            trans->lchild = to_child;
        }
        else {
            trans->rchild = to_child;
        }
        to_child = trans;
    };
    if (rand_bool_gen()) {
        insert(to->lchild);
    }
    else {
        insert(to->rchild);
    }
    trans->parent = to;
}
```



2024/6/4




Outline


- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ **Pack**
- ◆ Result



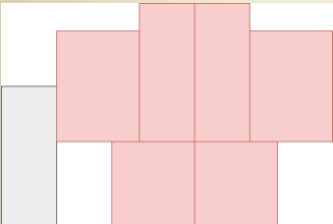
2024/6/4



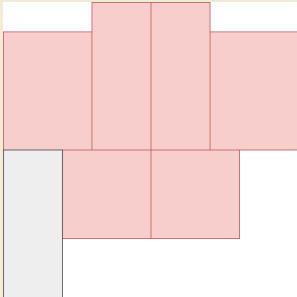
The concept of Packing




◆ How to make the left boundary of the symmetry island comply with the B*-tree definition



Lchild.x = Parent.x + Parent.width




Rchild.x = Parent.x




2024/6/4

Andy Yu-Guang Chen

41



The concept of Packing



◆ PackPass1

- Pack ASFBStarTree with **X=0** as the symmetry axis, and get the **width** of ASFBStarTree to get the real X coordinate of the symmetry axis

◆ PackPass2

- Pack ASFBStarTree with **real symmetry axis**


◆ Pack each node recursively

```
class PackorLinear {
public:
    PackorLinear(HBStarTree *);
    void operator()();

private:
    void PackDFS(ASFBStarTree *, ContourCiter);
    void PackPass1(ASFBStarTree *, SymmetryUnit *);
    void PackPass2(ASFBStarTree *, SymmetryUnit *, ContourCiter, ContourCiter);
    void PackSelfSymmetriC(ASFBStarTree *, SymmetryUnit *, ContourCiter);
    void PackSymmetryPair(ASFBStarTree *, SymmetryUnit *, ContourCiter, ContourCiter);
    ContourList contours_(100, 0.0);
    Rectangle &chip_;
    ASFBStarTree *root_;
};
```

```
void HBStarTree::PackorLinear::operator()() {
    if (!root_) return;

    contours_.Reset();
    chip_.SetMinX(0.0);
    chip_.SetMinY(0.0);
    chip_.SetMaxX(0.0);
    chip_.SetMaxY(0.0);
    PackDFS(root_, contours_.begin());
}
```



2024/6/4

Andy Yu-Guang Chen

42



PackDFS



```
void HBStarTree::PackOnLine::PackDFS(ASFBStarTree *asfbtree, ContourCIter hint) {
    // Traverse the right half of the ASFBStarTree. Set center X coordinate
    // of the ASFBStarTree to 0.0 and calculate its width and move all nodes
    // contained within the ASFBStarTree to their respective "relative" X coordinates.
    asfbtree->bounding_box.SetHeight(0.0);
    asfbtree->bounding_box.SetWidth(0.0);
    if (asfbtree->root->right_half == asfbtree->root->left_half) /* 'root' is self-symmetric */ {
        asfbtree->root->right_half->MoveCenterXto(0.0);
    }
    else /* 'root' is symmetry-pair */ {
        asfbtree->root->right_half->MoveMinXto(0.0);
        asfbtree->root->left_half->MoveMaxXto(0.0);
    }
    PackPass1(asfbtree, asfbtree->root);

    // Traverse the right half of the ASFBStarTree. Calculate the max Y
    // coordinate. Move all nodes contained within the ASFBStarTree to their
    // respective "real" X and Y coordinates.
    asfbtree->begin = std::prev(contours._end());
    asfbtree->end = contours._begin();
    PackPass2(asfbtree, asfbtree->root, hint, hint);
    asfbtree->bounding_box.MoveMinYto(asfbtree->root->right_half->GetMinY());
    chip_.setWidth(std::max(chip_.GetWidth(), asfbtree->bounding_box.GetMaxX() - chip_.GetMinX()));
    chip_.setHeight(std::max(chip_.GetHeight(), asfbtree->bounding_box.GetMaxY() - chip_.GetMinY()));


    // Recursively traverse the HBStarTree.
    if (asfbtree->lchild) {
        asfbtree->lchild->bounding_box.MoveMinXto(asfbtree->bounding_box.GetMaxX());
        PackDFS(asfbtree->lchild, asfbtree->end);
    }
    if (asfbtree->rchild) {
        asfbtree->rchild->bounding_box.MoveMinXto(asfbtree->bounding_box.GetMinX());
        PackDFS(asfbtree->rchild, asfbtree->begin);
    }
}
```




2024/6/4

Andy Yu-Guang Chen


43



Outline




- ◆ Overview
- ◆ Data Structure
- ◆ Placement With Optimization
- ◆ Perturb
- ◆ Pack
- ◆ Result




2024/6/4

Andy Yu-Guang Chen

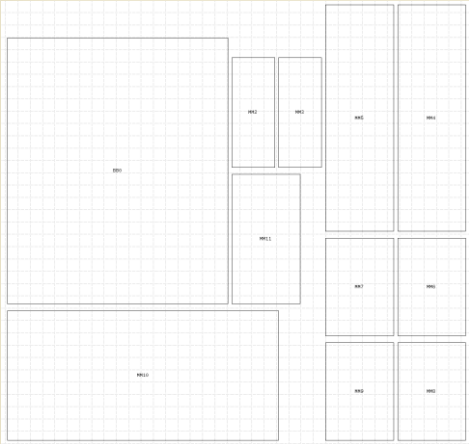
44




Case 1



◆ Cost = 157.62023







2024/6/4

Andy Yu-Guang Chen

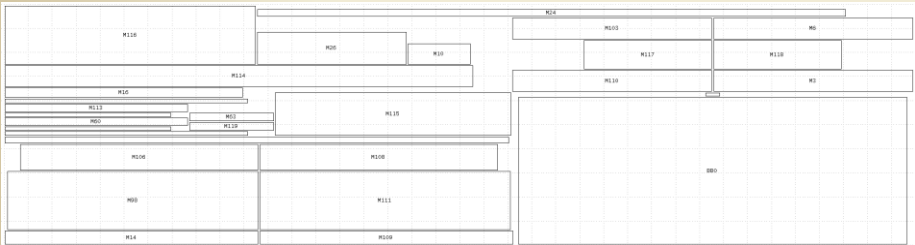
45




Case 2



◆ Cost = 3708.74382







2024/6/4

Andy Yu-Guang Chen

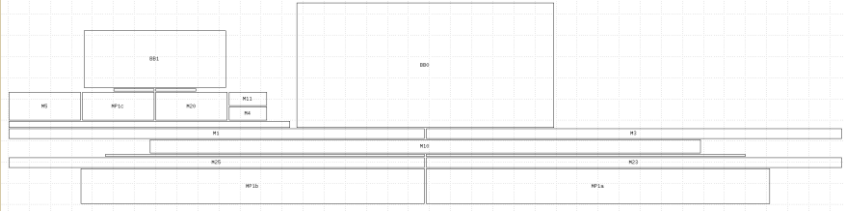
46




Case 3



◆ Cost = 3409.205383







2024/6/4

Andy Yu-Guang Chen

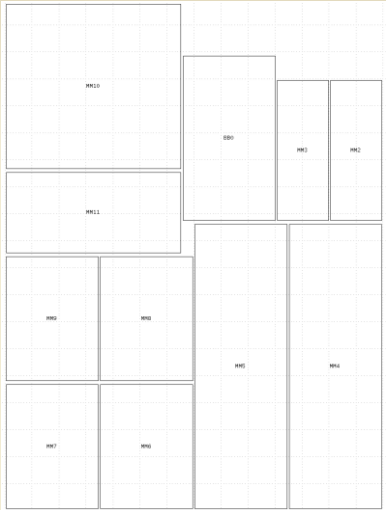
47




Case 4



◆ Cost = 394.178649







2024/6/4

Andy Yu-Guang Chen

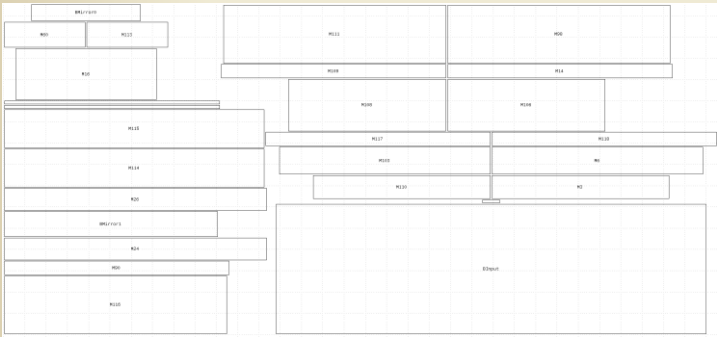
48




Case 5



◆ Cost = 4706.925973





2024/6/4

Andy Yu-Guang Chen

49



Q&A



◆ Email

➢ adcarr00334@gmail.com





Lecture01

Slide 50

Andy, Yu-Guang Chen
 Assistant Professor, Department of EE, NCU
 Email: andygchen@ee.ncu.edu.tw
 FB: Yu-Guang Chen
 IG: ncu.eda.andy
 Google account: andyygchen.ncu
 Slides credit: TA Yu-Heng Tsao

2024/6/4

Andy Yu-Guang Chen

51