Chip Placement Optimization using Integer Descent

Column-wise Restricted Movement with Discrete Conflict Minimization

Problem 4

1 Problem Overview

Problem Statement

Objective: Optimize the placement of rectangular chips on a 10×10 grid to minimize total conflict score using integer descent optimization.

Key Constraints:

- Each chip c_i locked to specific row y_i (no vertical movement)
- Horizontal movement restricted to integer steps only
- All chips are tall rectangles ($h_i > w_i$)
- Some chips connected via netlist requiring minimal wiring cost

Grid Configuration: 10×10 matrix with 8 chips of varying dimensions

1.1 Chip Specifications

Chip ID	Size (w×h)	Fixed Row (y)	Initial x	Final x
c1	2×4	0	0	1
c2	2×5	1	1	0
c3	1×3	0	1	3
c4	2×5	4	2	2
c5	2×4	3	3	4
c6	1×4	2	2	3
c7	2×5	5	0	0
c8	1×3	6	4	4

1.2 Netlist Connections

Connected chip pairs requiring wiring cost minimization:

$$\{(1,2), (2,6), (2,3), (3,5), (4,5), (5,6), (1,6), (7,4), (7,2), (8,5)\}$$

2 Conflict Score Formulation

Conflict Score Formulation

Total Conflict Score:

$$\text{Conflict}(P) = \sum_{(c_i,c_j)\in\text{Connections}} \text{WiringCost}(c_i, c_j) + \sum_{(c_i,c_j)\in\text{Overlaps}} \text{OverlapBlocks}(c_i, c_j)$$

Components:

1. **Wiring Cost:** Manhattan distance for connected chips
2. **Overlap Cost:** Number of grid cells occupied by multiple chips

2.1 Wiring Cost Calculation

For connected chips c_i and c_j:

$$\text{WiringCost}(c_i, c_j) = \max\{0, x_j - (x_i + w_i), x_i - (x_j + w_j)\} + |y_i - y_j|$$

Interpretation:

- **Horizontal gap:** Distance between chip boundaries (0 if overlapping/adjacent)
- **Vertical distance:** Absolute difference in y-coordinates
- **Total cost:** Sum of horizontal and vertical components

2.2 Overlap Cost Calculation

For any chip pair c_i and c_j:

$$\text{OverlapBlocks}(c_i, c_j) = |\text{Cells}(c_i) \cap \text{Cells}(c_j)|$$

Where $\text{Cells}(c_i)$ represents the set of grid cells occupied by chip c_i.

3 Integer Descent Algorithm

Integer Descent Algorithm

Core Principle: Iteratively move each chip left or right by one unit if it reduces the total conflict score.

Key Features:

- **Greedy Selection:** Choose the move with maximum improvement
- **Integer Constraints:** Only unit horizontal movements allowed
- **Convergence:** Stop when no beneficial moves remain

Algorithm Implementation

Algorithm 1 Integer Descent Optimization

Require: Initial chip positions, grid constraints, connection list
Initialize $iteration \leftarrow 0$
$conflict\_history \leftarrow []$
while $iteration < max\_iterations$ **do**
  $current\_score \leftarrow calculate\_total\_conflict\_score()$
  $conflict\_history.append(current\_score)$
  $best\_improvement \leftarrow 0$
  $best\_move \leftarrow None$
  for each chip c_i in chips **do**
    for each neighbor position $x_{new} \in \{x_i - 1, x_i + 1\}$ **do**
      if $is\_valid\_position(c_i, x_{new})$ **then**
        $old\_x \leftarrow c_i.x$
        $c_i.x \leftarrow x_{new}$
        $new\_score \leftarrow calculate\_total\_conflict\_score()$
        $improvement \leftarrow current\_score - new\_score$
        if $improvement > best\_improvement$ **then**
          $best\_improvement \leftarrow improvement$
          $best\_move \leftarrow (i, x_{new})$
        end if
        $c_i.x \leftarrow old\_x$ {Revert change}
      end if
    end for
  end for
  if $best\_improvement > 0$ **then**
    Apply $best\_move$
  else
    break {Converged}
  end if
  $iteration \leftarrow iteration + 1$
end while
return $conflict\_history$

3.1 Implementation Details

```python
def integer_descent_step():
    current_score = calculate_total_conflict_score()
    best_improvement = None
    best_chip_index = None
    best_new_x = None

    for idx, chip in enumerate(chips):
        for new_x in get_neighbors(chip):
            old_x = chip['x']
            chip['x'] = new_x
            new_score = calculate_total_conflict_score()
            improvement = current_score - new_score

```

```python
            if improvement > 0 and (best_improvement is None or improvement > best_improvement):
                best_improvement = improvement
                best_chip_index = idx
                best_new_x = new_x

            chip['x'] = old_x  # Revert for next evaluation

    if best_improvement is not None:
        chips[best_chip_index]['x'] = best_new_x
        return True
    return False

def get_neighbors(chip):
    neighbors = []
    if is_valid_position(chip, chip['x'] - 1):
        neighbors.append(chip['x'] - 1)
    if is_valid_position(chip, chip['x'] + 1):
        neighbors.append(chip['x'] + 1)
    return neighbors
```

Listing 1: Core Optimization Functions

4 Experimental Results

Optimization Results

Optimization Performance:

- **Initial Conflict Score:** 40.0
- **Final Conflict Score:** 29.0
- **Total Improvement:** 11.0 (27.5% reduction)
- **Convergence:** 7 iterations

4.1 Iteration-by-Iteration Progress

Iteration	Conflict Score	Improvement
0 (Initial)	40.0	-
1	37.0	3.0
2	33.0	4.0
3	32.0	1.0
4	31.0	1.0
5	30.0	1.0
6	29.0	1.0
7 (Converged)	29.0	0.0

4.2 Final Conflict Breakdown

Cost Component	Value
Total Wiring Cost	20
Total Overlap Cost	9
Total Conflict Score	**29**

Detailed Wiring Costs:

- Chips 1-2: 1, Chips 1-6: 2, Chips 2-3: 2, Chips 2-6: 2
- Chips 2-7: 4, Chips 3-5: 3, Chips 4-5: 1, Chips 4-7: 1
- Chips 5-6: 1, Chips 5-8: 3

Remaining Overlaps:

- Chips 1-2: 3 blocks, Chips 2-7: 2 blocks, Chips 3-6: 1 block
- Chips 4-6: 2 blocks, Chips 5-8: 1 block

5 Grid Visualization Analysis

5.1 Initial State Grid

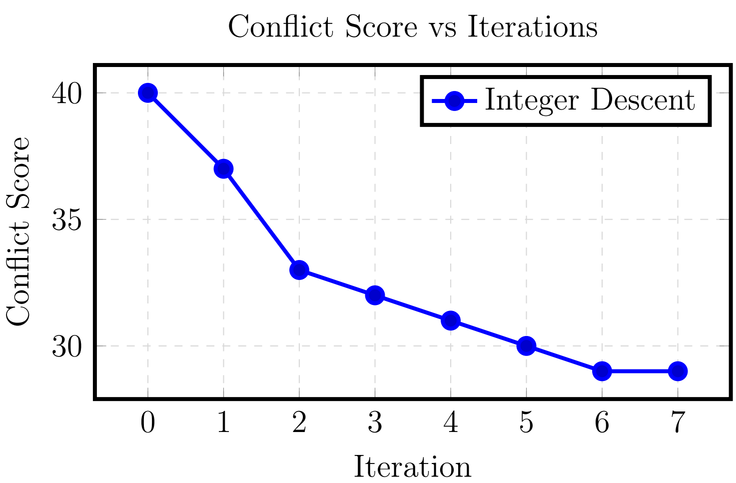
y	x=0	1	2	3	4	5	6	7	8	9
9	7	7	0	0	0	0	0	0	0	0
8	7	7	4	4	8	0	0	0	0	0
7	7	7	4	4	8	0	0	0	0	0
6	7	7	4/5	5/8	0	0	0	0	0	0
5	7	2/7	2/4/6	4/5	5	0	0	0	0	0
4	0	2	2/4/6	4/5	5	0	0	0	0	0
3	1	1/2	2/6	5	5	0	0	0	0	0
2	1	1/2/3	2/6	0	0	0	0	0	0	0
1	1	1/2/3	2	0	0	0	0	0	0	0
0	1	1/3	0	0	0	0	0	0	0	0

5.2 Final State Grid

y	x=0	1	2	3	4	5	6	7	8	9
9	7	7	0	0	0	0	0	0	0	0
8	7	7	4	4	8	0	0	0	0	0
7	7	7	4	4	8	0	0	0	0	0
6	7	7	4	4	5/8	5	0	0	0	0
5	2/7	2/7	4	4/6	5	5	0	0	0	0
4	2	2	4	4/6	5	5	0	0	0	0
3	2	1/2	1	6	5	5	0	0	0	0
2	2	1/2	1	3/6	0	0	0	0	0	0
1	2	1/2	1	3	0	0	0	0	0	0
0	0	1	1	3	0	0	0	0	0	0

6 Convergence Analysis

6.1 Optimization Trajectory



Convergence Characteristics:

- **Rapid Initial Improvement:** Score drops from 40 to 33 in first 2 iterations
- **Gradual Refinement:** Incremental improvements of 1 point per iteration
- **Local Optimum:** Convergence after 7 iterations (no further beneficial moves)

7 Algorithm Performance Analysis

7.1 Computational Complexity

Algorithm Implementation

Time Complexity Analysis:

- **Per Iteration:** $O(n \cdot 2 \cdot C)$ where n = number of chips, C = conflict calculation cost
- **Conflict Calculation:** $O(n^2)$ for all pairwise interactions
- **Total Complexity:** $O(I \cdot n^3)$ where I = number of iterations

Space Complexity: $O(n + G)$ where G = grid size
Observed Performance:

- 8 chips, 10×10 grid
- 7 iterations to convergence
- Approximately $7 \times 8 \times 2 = 112$ position evaluations

7.2 Movement Pattern Analysis

Chip	Initial x	Final x	Net Movement
c1	0	1	+1 (right)
c2	1	0	-1 (left)
c3	1	3	+2 (right)
c4	2	2	0 (no change)
c5	3	4	+1 (right)
c6	2	3	+1 (right)
c7	0	0	0 (no change)
c8	4	4	0 (no change)

Optimization Insights:

- **Spreading Pattern:** Chips generally moved apart to reduce overlaps
- **Connection Awareness:** Connected chips positioned to minimize wiring costs
- **Constraint Respect:** All movements maintained grid boundaries

8 Key Algorithmic Insights

1. **Discrete vs. Continuous:** Integer constraints lead to fundamentally different optimization landscape compared to continuous methods.
2. **Local Search Effectiveness:** Simple greedy moves achieved significant improvement (27.5
3. **Conflict Balance:** Algorithm effectively balanced wiring costs against overlap penalties.
4. **Convergence Guarantee:** Finite search space ensures termination at local optimum.

9 Limitations and Future Work

Current Limitations:

- **Local Optimum:** May not find global optimum
- **No Vertical Movement:** y-coordinates fixed by problem constraints
- **Greedy Selection:** No look-ahead or backtracking

Potential Improvements:

- **Simulated Annealing:** Allow occasional uphill moves to escape local optima
- **Multi-Start:** Run from multiple random initial configurations
- **Genetic Algorithm:** Explore broader solution space with population-based approach

10 Conclusion

The integer descent algorithm successfully optimized chip placement on the 10×10 grid, achieving a 27.5

- **Fast Convergence:** Rapid improvement in early iterations
- **Practical Implementation:** Simple, intuitive algorithm
- **Constraint Handling:** Natural integration of discrete movement restrictions
- **Multi-Objective Balance:** Effective handling of competing wiring and overlap costs

The final configuration eliminates most overlaps while maintaining reasonable wiring costs, demonstrating the algorithm's ability to find practical solutions for chip placement optimization problems.