COL215 Software Assignment 2: Wiring-aware Gate Positioning

Vansh Ramani 2023CS50804
Samyak Sanghvi 2023CS10807

October 5, 2024

Contents

1 Introduction

In this assignment, we extend on the gate packing problem. The gates contain pin locations, and the objective is to minimize the total wire length of the circuit by positioning the gates optimally. We aim to minimize the total Manhattan distance between connected pins.

2 Problem Statement

Given a set of rectangular logic gates $g_1, g_2, \ldots, g_n$, each with width, height, and pin coordinates on its boundary, the task is to assign positions to all gates in a plane such that:

- No two gates overlap.
- The total wire length is minimized using the semi-perimeter method.

2.1 Notes

- Gates cannot be re-oriented.
- All wiring is horizontal or vertical.
- Wire length is estimated using the semi-perimeter method by forming a rectangular bounding box around the pins.

3 Mathematical Formulation

Consider a structure consisting of gates and m wires, each connecting a pair of pins. Let the coordinates of the pins connected by wire w_i be $(x_i^{(1)}, y_i^{(1)})$ and $(x_i^{(2)}, y_i^{(2)})$. The length of each wire is measured using the Manhattan distance between its coordinates, which can be expressed as:

$$f(w_i) = |x_i^{(1)} - x_i^{(2)}| + |y_i^{(1)} - y_i^{(2)}|$$

The total objective function is the sum of the areas of the gates and the weighted sum of the wiring lengths. Let A_{total} denote the total area of the gates, and let $\bar{d}$ represent the mean width of the wires. The full objective function is:

$$\text{Objective Function} = A_{\text{total}} + \bar{d} \cdot \sum_{i=1}^{m} f(w_i)$$

Since A_{total} and $\bar{d}$ are constants, minimizing the objective function reduces to minimizing the total Manhattan distance between the pins, which is:

$$\sum_{i=1}^{m} f(w_i) = \sum_{i=1}^{m} \left(|x_i^{(1)} - x_i^{(2)}| + |y_i^{(1)} - y_i^{(2)}| \right)$$

Therefore, the task is to minimize the total Manhattan distance between the connected pins to achieve the optimal wiring configuration. Estimating wire length using Manhattan distance is known as the Semi Perimeter Method.

4 Design Decisions

The gates will be arranged such that the overlap constraint is satisfied, and the wiring length is minimized using the semi-perimeter method.

The following logic was used to optimize and find a good solution for the problem:

4.1 Identifying and creating clusters

Cluster is a group of gates that are connected to each other through some wire

It is possible that gates in the given set of gates may be disjoint. Hence optimizing between completely disjoint gates makes no sense. Therefore the first step was using **Depth First Search**.

4.2 Stochastic Relaxation Optimizer for Packing the Gates within a Cluster

Stochastic Relaxation Optimizer (SR) is an optimization technique inspired by the cooling process in metallurgy. The goal is to find an optimal or near-optimal solution by exploring the solution space. It initially allows sub-optimal solutions to avoid local minima but gradually becomes more selective as the "temperature" decreases.

- **Large Search Space**: SR explores various placements of circuit components effectively.
- **Avoids Local Minima**: By accepting worse solutions probabilistically, SR avoids getting stuck in sub-optimal configurations.
- **Exploration and Exploitation**: It balances the broad search in the beginning with detailed refinement at lower temperatures.

We started with a initial tightly packed placement, taken from the bin packing algorithm of Software Assignment 1 and then proceeded to variate in a small range and then iterate over the other gates.

Finally in the end, we place each of these clusters discretely on the 2-D plane, to avoid overlaps

5 Implementation Details

5.1 Input Format

The input contains:

- For each gate: the width, height, and pin coordinates relative to the bottom-left corner of the gate.
- The wire connections between gates, specifying which pins are connected.

5.2 Output Format

The output includes:

- The bounding box of the gate arrangement.
- The wire length calculated.
- The positions of each gate, specified as the coordinates of the bottom-left corner.

5.3 Wire Length Approximation

The wire length between connected pins is approximated using the semi-perimeter of the bounding box that encloses all connected pins. The following steps outline how the wire length is computed:

- For each pin p_i, check its connections with other pins using a connection matrix.
- If pin p_i is connected to one or more pins, calculate a bounding box that encloses all connected pins.
- The wire length for these connected pins is approximated by the semi-perimeter of the bounding box.

Bounding Box Calculation

Given the coordinates of the connected pins, the bounding box is defined by the minimum and maximum x-coordinates and y-coordinates of the pins. Let the coordinates of pin p_i be (x_i, y_i), and let S_i be the set of all pins connected to p_i.

The bounding box is determined by:

$$x_{\min} = \min_{p_j \in S_i} \{x_j\}, \quad x_{\max} = \max_{p_j \in S_i} \{x_j\}$$

$$y_{\min} = \min_{p_j \in S_i} \{y_j\}, \quad y_{\max} = \max_{p_j \in S_i} \{y_j\}$$

Semi-Perimeter and Wire Length Approximation

The wire length for pin p_i and its connected pins is approximated by the semi-perimeter of the bounding box. The semi-perimeter is calculated as:

$$\text{Semi-perimeter} = (x_{\max} - x_{\min}) + (y_{\max} - y_{\min})$$

This semi-perimeter represents the Manhattan distance required to connect the pins inside the bounding box.

Total Wire Length

The total wire length for the entire layout is the sum of the semi-perimeter of the bounding boxes for all connected pins:

$$\text{Total wire length} = \sum_i \left((x^i_{\max} - x^i_{\min}) + (y^i_{\max} - y^i_{\min}) \right)$$

where $x^i_{\max}, x^i_{\min}, y^i_{\max}, y^i_{\min}$ refer to the bounding box of the pins connected to pin p_i.

5.4 Clustering using Depth First Search

Algorithm 1 DFS Clustering Algorithm

```
1: Input: Components C, Wires W
2: Output: Clusters of components
3: Create adjacency list AdjList from W
4: Initialize an empty set visited
5: Initialize an empty list clusters
6: function DFS(component, cluster)
7:     Add component to visited
8:     Add component to cluster
9:     for each neighbor in AdjList[component] do
10:        if neighbor not in visited then
11:            DFS(neighbor, cluster)
12:        end if
13:    end for
14: end function
15: for each component in C do
16:    if component not in visited then
17:        Initialize an empty set current_cluster
18:        DFS(component, current_cluster)
19:        Add current_cluster to clusters
20:    end if
21: end for
22: return clusters
```

The DFS clustering algorithm groups connected circuit components into clusters based on the wires between them. The algorithm is structured as follows:

1. **Adjacency List Construction**: The first step is to convert the list of wires between components into an adjacency list, which is a data structure that maps each component to its connected neighbors.

2. **DFS Function**: The DFS function is used to explore all components connected to a given component. When called on a starting component, the function marks it as visited, adds it to the current cluster, and recursively explores all its neighbors. Neighbors are other components that are connected to the starting component via wires. This process continues until all components in the current cluster are visited.

3. **Main Loop**: The main loop iterates over all components. If a component hasn't been visited yet, it means it's the starting point of a new cluster. DFS is called on this component to discover the entire cluster, and once the DFS is finished, the cluster is added to the list of clusters.

4. **Cluster Creation**: Each DFS call explores all components reachable from the starting component. After the DFS completes for a given component, the set of visited components forms a cluster. This set is added to the `clusters` list.

Algorithm 2 Stochastic Relaxation Optimizer for Circuit Placement

Input: Components C, Wires W, Iterations K, Cooling Factor α
Output: Optimized positions of components, total wire length
Initialize component positions using a bin-packing algorithm.
Calculate initial total wire length L_0
Set initial temperature $T_{\max}$
for $i = 1$ to K **do**
    Select a random component $c \in C$
    Randomly move component c to a new position (x', y')
    Calculate the new wire length L_{new} based on the new position
    if New configuration is better ($L_{\text{new}} < L_{\text{current}}$) **then**
        Accept the new configuration
    else
        Accept with probability $P = \exp\left(\frac{L_{\text{current}} - L_{\text{new}}}{T_i}\right)$
    end if
    Update temperature $T_{i+1} = \alpha \times T_i$
end for
Return the final positions of the components and the optimized wire length

5.5 Stochastic Relaxation Optimizer

Stochastic Relaxation Optimizer is implemented to optimize the placement of circuit components in such a way that the total wire length between them is minimized. The key steps of the algorithm are as follows:

1. **Initialization**:
    - The algorithm starts with an initial placement of circuit components using bin packing algorithm used in assignment 1 and a high temperature T_0.
    - The initial wire length, representing the current energy of the system, is calculated.
2. **Component Relocation**:
    - A component is randomly selected and its new position is computed by placing it at a random spot within the maximum possible boundaries.
    - The wire length is recalculated based on this new position.
    - If there is a collision, they we skip.
3. **Acceptance Criteria**:
    - If the new configuration has a lower wire length (lower energy), the new position is accepted.
    - If the new configuration has a higher wire length, it is accepted with a probability $P = \exp\left(-\frac{\Delta E}{T}\right)$, where ΔE is the increase in wire length and T is the current temperature.
4. **Cooling Schedule**:

- After a certain number of iterations, the temperature is reduced according to a cooling factor α such that $T_{\text{new}} = \alpha \cdot T$.
- This gradual cooling reduces the likelihood of accepting worse solutions over time, allowing the algorithm to focus on refining the best solutions.

5. **Stopping Condition**:
   - The process repeats for a fixed number of iterations, when a fixed number of iterations of annealing has occurred, it exits the loop.
6. **Final Solution**:
   - The algorithm outputs the best layout of components found during the optimization process, with the corresponding minimized wire length.

6 Test Cases

6.1 Given test Cases

We get good results on given test cases. 32, 27, 55, 36 as wirelenghts for given testcases with figures attached below.

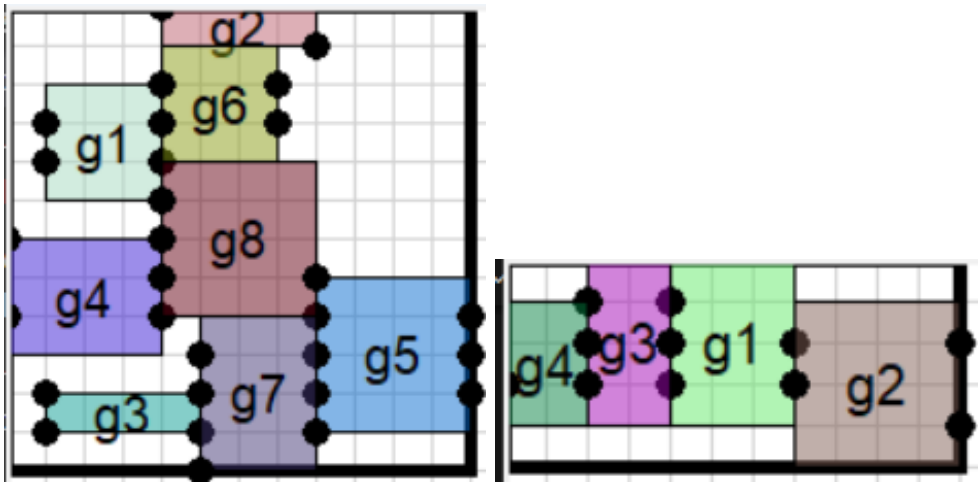

Figure 1: TestCase1 TestCase2

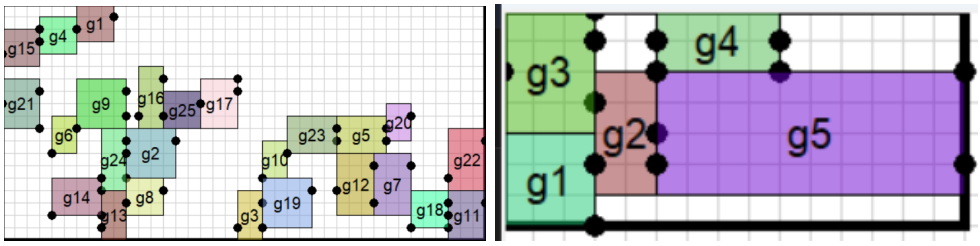

Figure 2: TestCase3 TestCase4

We can observe that on all test cases our algorithm given enough iterations provides better results for each test case, than the sample output.

6.2 Generated Test Cases

We considered and analysed our code over multiple test cases and proved its robustness in multiple cases.
When there are multiple gate clusters our method performs very well.
Our method doesn't perform very well with large sequential chains as it cannot find good random positions.

The following type of test cases were generated and tested for: Below are images and time taken for the test cases submitted

1. **Densely Connected**
2. **Multi Clustered Connections**
3. **Diagonally Sequential Connections**
4. **Star-Like Connections**
5. **Mesh-Like Connections**
6. **Randomly Generated Medium**
7. **Randomly Generated Large**

Images are attached after the conclusion section in the end.
Example Test case of each are attached in submission.
Note that for large we have not attached a figure as it was too large to fit in given grid.
The method used to generate all of these is attached in *generate_tc.py*. Kindly refer to that for specifics.

7 Time Complexity Analysis

With the inclusion of the bin-packing algorithm, the overall time complexity is divided into two parts:

- **Bin-Packing**: The bin-packing algorithm runs with a time complexity of $O(n^2)$, where n is the number of components.
- **Simulated Annealing**: For each iteration of SA, recalculating the wire length takes $O(m)$, and checking for collisions takes $O(n)$. Since the algorithm runs for K iterations, the overall complexity is $O(K \times (m+n))$.

Thus, the total time complexity becomes:

$$O\left(n^2 + K \times (m+n)\right)$$

Where:

- n: number of components,
- m: number of wires,
- K: total number of iterations.

For the worst case, On running 1000 gates 40000 wires K = 50, it took 168 seconds. (Worst Case)

8 Conclusion

The Wiring-aware Gate Positioning problem was approached using a combination of Depth First Search (DFS) for clustering and Stochastic Relaxation Optimization for gate placement. This method effectively minimized total wire length while avoiding gate overlaps. The algorithm performed well on various test cases, especially those with multiple gate clusters. However, it showed some limitations with large sequential chains. The time complexity analysis revealed that the algorithm scales reasonably well with the number of gates and wires. Overall, this approach provides a robust solution to the circuit placement problem, balancing between optimization quality and computational efficiency.

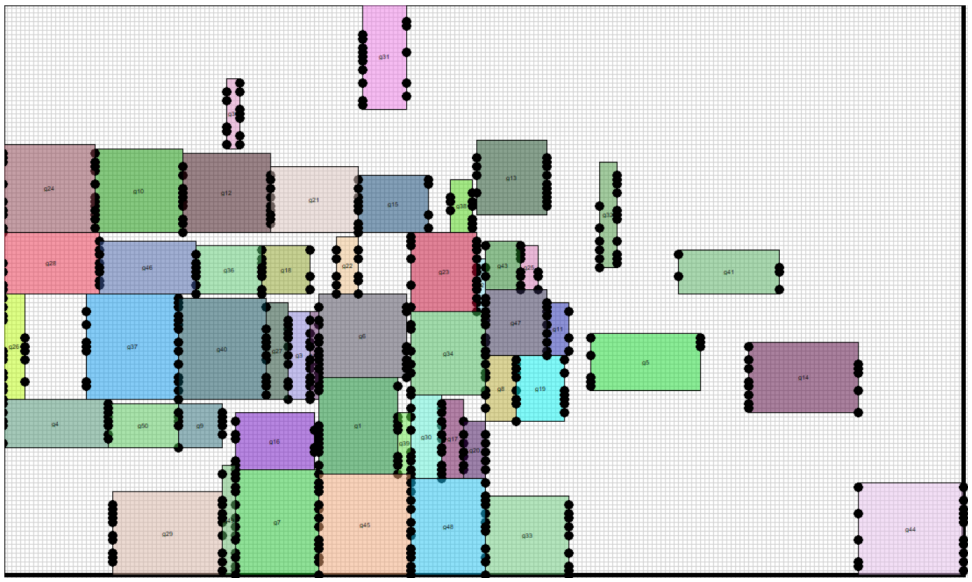

Figure 3: Densely Connected

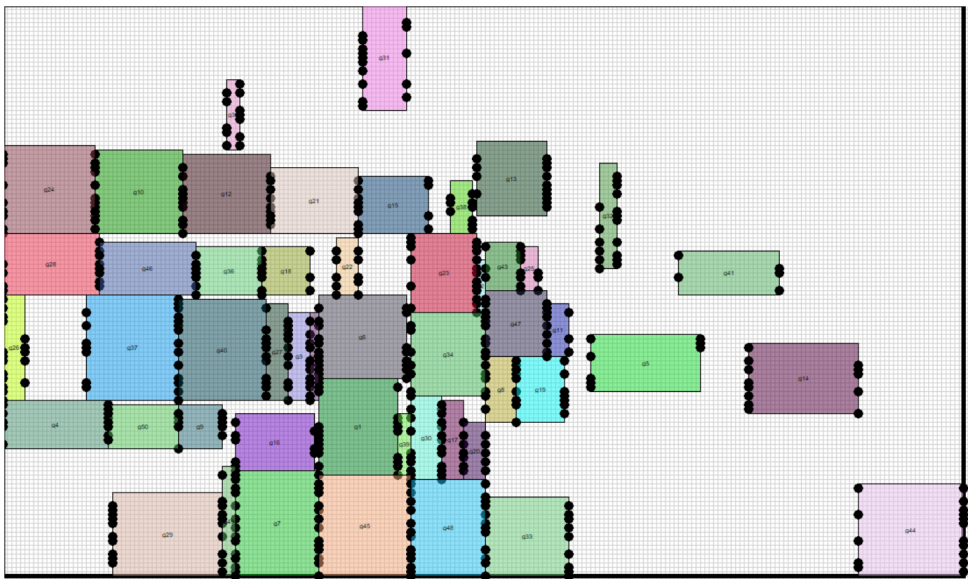

Figure 4: Multi Clustered Connections

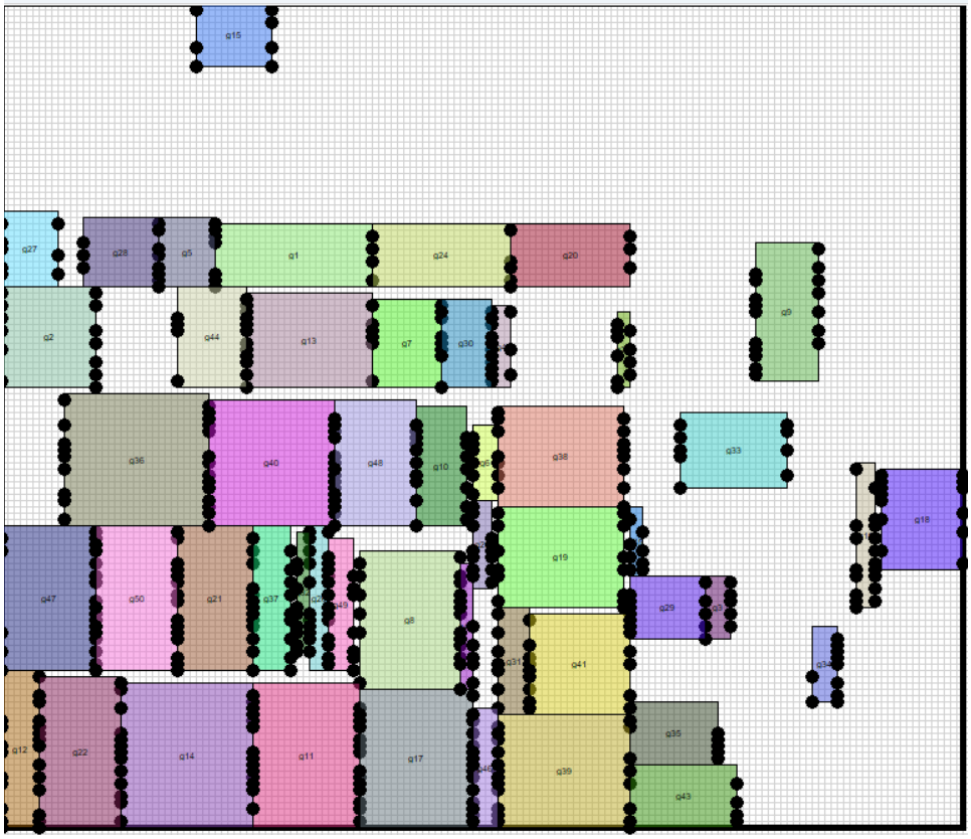

Figure 5: Diagonally Sequential Connections

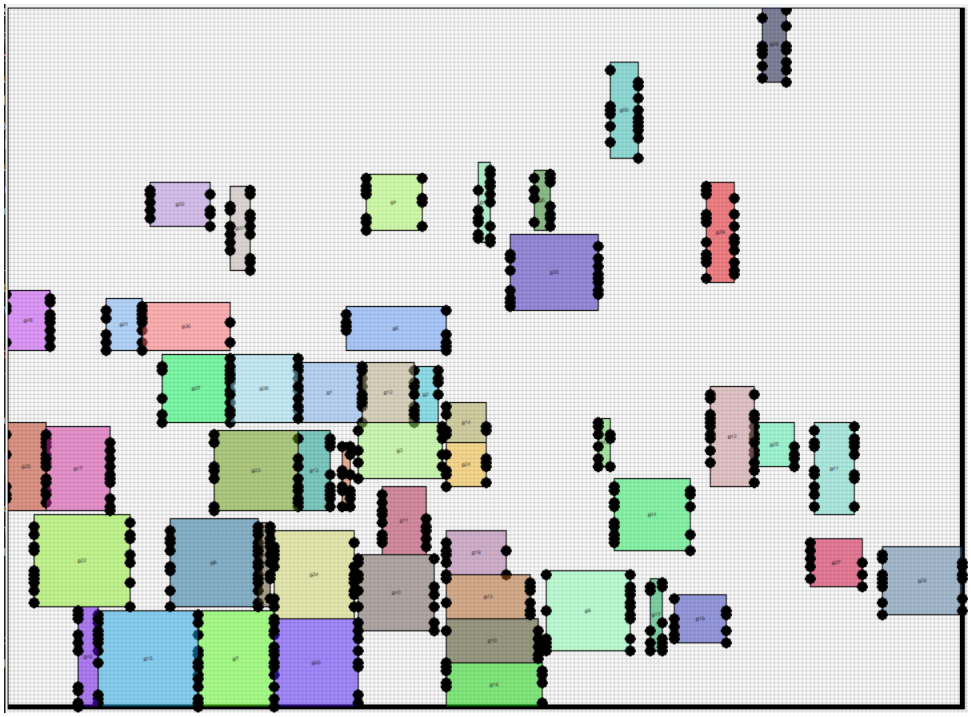

Figure 6: Star-Like Connections

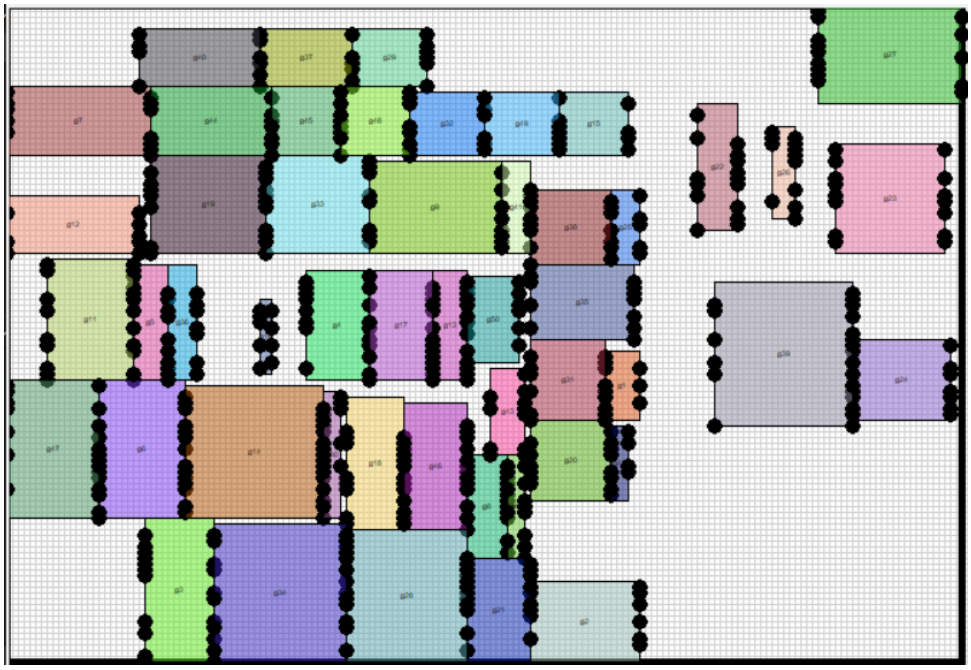

Figure 7: Mesh-Like Connections

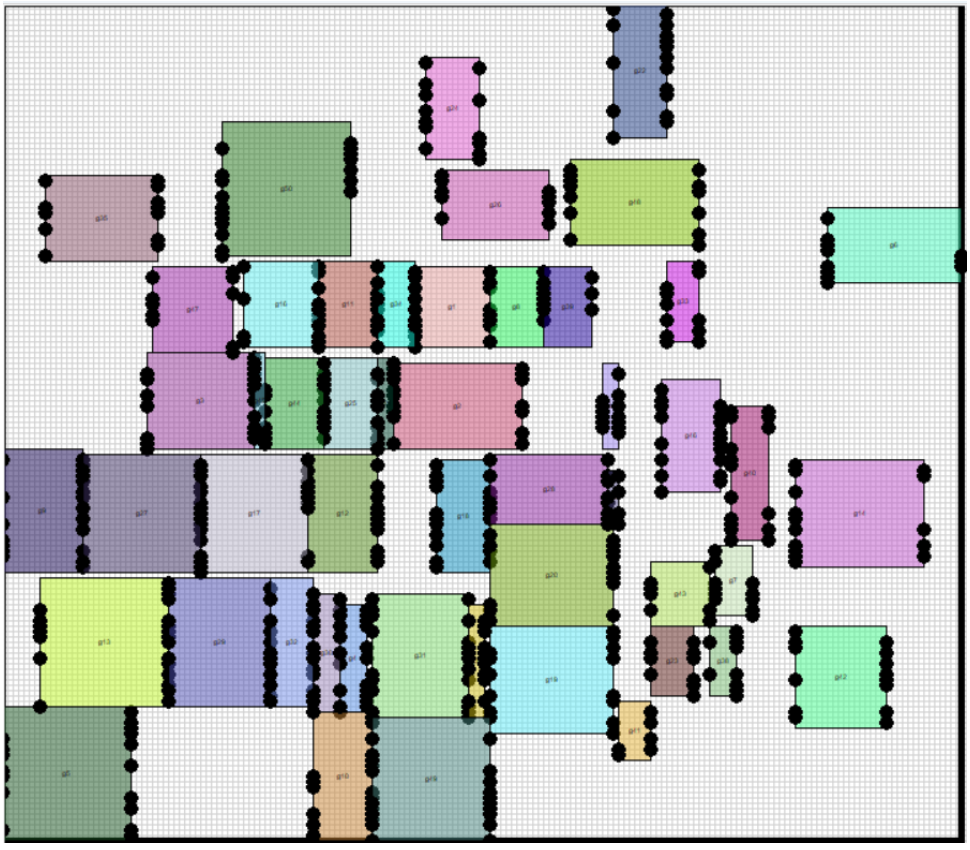

Figure 8: Randomly Generated (Medium)