P2P: Point Cloud to Panel Layout Optimization

Nisha Deborah Philips, Yifang Liu, Nolan W. Hayes, Diana Hun, and Bryan P. Maldonado

Buildings and Transportation Science Division, Oak Ridge National Laboratory*, United States of America

philipsn@ornl.gov, liuy5@ornl.gov, hayesnw@ornl.gov, hunde@ornl.gov, maldonadopbp@ornl.gov

Abstract -

Building envelope retrofits, despite their benefits on enhancing energy efficiency, progress slowly due to high operating costs. Overclad panelized systems present an attractive solution to make retrofits affordable and easy to install. However, several stages of the retrofit process remain disconnected, suboptimal, and require significant human intervention. This study aims to bridge the gap between digital twin generation and overclad panel installation by automating the design of an optimal panel layout directly from a building envelope point cloud. In this end-to-end approach, the dimensional twin is generated by segmenting the facade point cloud. The facade then undergoes a three-step process to generate an optimized panel layout for integration with automated placement systems.

Keywords -

Building envelope retrofit, Panel layout design, Constraint satisfaction problem

1 Introduction

Rapid consumption of world energy has raised concerns over the depletion of energy resources and the adverse effects it has on the environment. With the construction industry accounting for 20–40% of the total energy consumption [1], significant effort has been directed to improve the energy efficiency of existing buildings. In response, several government organisations have implemented initiatives to enhance the energy performance of buildings, while extensive research efforts have been made to optimize the energy performance of existing structures. Results have shown that energy consumption can significantly be reduced via retrofitting existing buildings as opposed to constructing new ones [2].

With this surge of interest in retrofitting, leveraging technologies such as Digital Twins, which are digital replicas of physical objects, are imperative to optimize retrofits and improve the energy efficiency of existing buildings [3]. Digital twins have a variety of applications from being integrated with Internet of Things (IoT) for data collection to Artificial Intelligence (AI) for processing the collected data [4]. In this context, digital twins are used to virtually reconstruct a building and generate optimized panel layout designs for each facade through an automated process.

Panel layout generation from architectural drawings falls into two categories: "heavy" panel layouts for facades with openings (windows, doors) and "light" panel layouts for facades without such features [5]. Heavy solutions use prefabricated panels, while light solutions require on-site panel rework. Despite these differences, both types can utilize the same optimization algorithm. Furthermore, automated panel layout tools should be integrated with existing technologies for digital twin generation like the Automatic point Cloud Building Envelope Segmentation (Auto-CuBES) [6] and for automated installation like the Real-Time Evaluator (RTE) [7, 8] to create a unified retrofit framework.

The panel layout designer could either be performed manually by an architect or automated via AI. The manual panel layout design is inherently labor intensive and requires substantial amount of man-hours. Given these challenges, advancements in computational methods have paved the way for automating the panel layout design process using AI. By transforming the problem into a mathematical optimization framework, such as the bin packing problem [9], it becomes possible to model the allocation of panels as a computationally solvable task. The bin packing problem, a well-known combinatorial optimization challenge, involves efficiently packing objects of varying sizes into fixed-capacity bins, minimizing wasted space. For panel layout design, this approach aims to maximize panel coverage while adhering to the architectural constraints.

In this paper, we propose modeling panel layout design as a *meta-rectangle packing problem*, extending traditional rectangle packing [10] to non-uniform bins. This mathematical framing allows us to use AI-driven algorithms, specifically Constraint Satisfaction Problem (CSP), to optimize panel packing. Section 2 describes existing and proposed panel layout design algorithms. Section 3 presents the implementation and analyzes execution time and coverage. Section 4 concludes and outlines future work.

2 Optimal panel layout design

2.1 Digital twin generation

Building envelope retrofit extensively uses point cloud data to generate as-built dimensions of the building. However, the generation of digital twins from point cloud data has traditionally been a manual process, making it laborious and time-consuming. To address this limitation, machine learning methods, such as the Automatic point Cloud Building Envelope Segmentation (Auto-CuBES) [6] algorithm, can automate the generation of digital twin from point cloud data.

Auto-CuBES divides the algorithm into sequential tasks that generate a digital twin from building point cloud data of 3mm density. Despite its computational efficiency and accuracy, the input parameters scale linearly with features, requiring considerable time for parameter calibration. To address this bottleneck, the authors developed Auto-CuBES++, a digital tool providing real-time parameter calibration for building facade segmentation, available for academic or commercial use [11]. This tool assists users at every algorithm stage by visualizing results and allowing parameter adjustments in real time without full re-execution. The software also preserves execution states, ensuring input parameters are retained and eliminating the need for re-entry when rerunning the algorithm.

In addition to preserving the state of the run, Auto-CuBES++ also stores the dimensions as a JSON file containing the coordinates of the facade and its associated openings. These JSON files facilitates validating the digital twin's accuracy in virtualizing the building. The coordinates in the JSON file are represented as tuples of real numbers (x, y, z), which are used to map the facade to a Cartesian coordinate system. This representation provides a structured framework for panelizing the facade and seamlessly integrating it with the automated installation systems, such as the RTE.

2.2 Bin packing problems

Panelization, the key theme of this paper, is the process of prefabricating building's structural elements off-site and assembling a panelized system to minimize on-site labor. In a traditional panelized construction of a wall, the wall is divided into a simple layout and the pre-manufactured panels are assembled accordingly. However, traditional techniques of panel layout design are not as straight forward and often require human intervention to optimize the generated layout design [5].

The panel layout design problem can be represented as a *Bin packing problem* [9], and more specifically as a *Rectangle Packing Problem* [12] since we focus on single-wall arrangements. Rectangle Packing is an extension of Bin packing that uses a single bin instead of multiple bins.

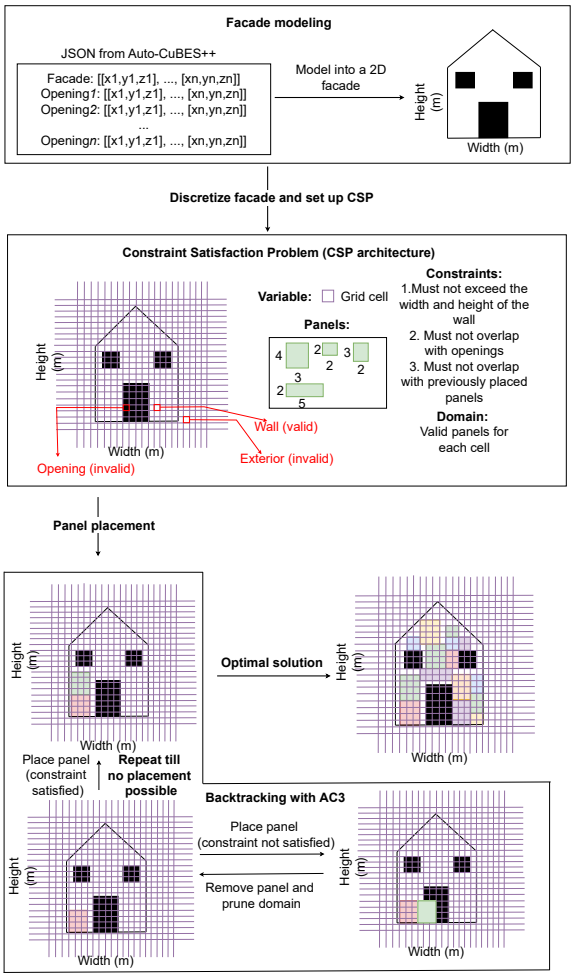


Figure 1. Facade to panel layout

As Bin packing problems are NP-hard, Rectangle Packing Problems inherit this computational complexity and are also NP-complete [10].

These Rectangle Packing Problems can be solved using deterministic algorithms, which use variables and policies to explore the solution space [10], and non-deterministic algorithms, such as Genetic Algorithm (GA) [13] and Simulated Annealing [14]. While deterministic approaches rely on predefined policies and constraints, non-deterministic methods leverage relationships between variables as data structures to search for optimal solutions.

2.3 Proposed panel layout design algorithm

This paper addresses panel packing using a deterministic algorithm, specifically the Constraint Satisfaction Problem (CSP). It combines backtracking [15] and arc consistency (AC-3) [16] to efficiently explore the solution space. Backtracking methodically scans solutions, reverting decisions that violate constraints, while AC-3 prunes infeasible

values to simplify the problem. Figure 1 illustrates this hybrid process, comprising three stages: 1) Facade modeling, 2) CSP formulation, and 3) solving via backtracking.

2.3.1 Facade modeling

Initially, the algorithm processes the JSON files generated by Auto-CuBES++ and extracts the geometric data required for facade modeling. Auto-CuBES++ generates two types of JSON files, one comprising of the facade boundary points, and another set of rectangular prism files for each individual opening in the facade. The algorithm transforms the extracted geometric data from the 3D space to a 2D representation based on the following rules:

- Points (x, z) for a facade parallel to the x and z axes
- Points (y, z) for a facade parallel to the y and z axes

2.3.2 CSP formulation

Following the facade modeling, the CSP problem for meta-rectangle packing is formulated. In a traditional rectangle packing algorithm, a set of rectangular items are packed into a rectangular box. However, due to the algorithm being applied to buildings, the rectangular box is adapted to a polygonal facade with or without openings. Additionally, the rectangular items are customized to panels of varying sizes that are packed on the wall.

Every CSP architecture is uniquely identified by a set of variables, the domains associated with the variables and a set of constraints. In the initial stage, the algorithm assumes a simplified scenario where the grid represents a facade without any openings (light panel layout design). In this case, each cell in the grid is valid, and the algorithm proceeds by assigning panels from the available domain to each of these cells. The process involves evaluating the constraints to ensure that the panels meet the architectural requirement of not aligning with the openings.

Once this basic configuration is established, the algorithm is adapted to handle the more complex scenario involving openings (heavy panel layout design). Openings, such as windows or doors, alter the layout of the facade, and certain cells are no longer suitable for panel placement. Each cell can fall under one of the categories shown in Table 1. Based on this information, the grid is refined to include only the valid cells, more specifically the wall points, so cells falling under any of the other categories are marked invalid and a panel is not placed on those cells.

The algorithm generates solutions that satisfy a set of panel placement constraints, as shown in Algorithm 1. Firstly, IS_WITHIN_BOUNDS() verifies that panels are placed only on valid cells within the grid. Secondly, DOES_NOT_OVERLAP() ensures that newly placed panels do not overlap with existing panels or openings. Finally, for light panel layouts, DOES_NOT_ALIGN() checks that panel edges do not align with opening edges to prevent potential air/heat leakage points.

Table 1. Grid cells vs Validity

Type of grid cell	Validity
Only wall	Valid
Only opening	Invalid
Only exterior	Invalid
Wall and opening	Invalid
Wall and exterior	Invalid
Opening and exterior	Invalid

Algorithm 1 Valid panel placement check

Require: Wall, Cell (x, y), Panel
Ensure: Panel satisfies or violates constraints
1: **Function** CONSTRAINTS(wall, x, y, panel):
2: **for** each cell in panel.area on wall **do**
3: **if** IS_WITHIN_BOUNDS(x, y, PANEL) **AND** DOES_NOT_OVERLAP(x, y, PANEL)) **AND** DOES_NOT_ALIGN_WITH_WINDOW(x, y, PANEL)) **then**
4: **return** True
5: **else**
6: **return** False
7: **end if**
8: **end for**

2.3.3 Backtracking with Arc Consistency

Following the initialization of the grid and the CSP architecture, Backtracking, one of the most commonly used algorithms, was employed to solve the problem. This algorithm exhaustively searches the solution space to find assignments or, in this case, panel placements that satisfy the constraints. However, due to the large search spaces involved in most CSP problems, the algorithm could get computationally expensive. Therefore, to improve the efficiency of the algorithm, backtracking is integrated with AC-3, thereby reducing the complexity of the problem by pruning invalid panel assignments. This hybrid approach is well suited for problems where constraints are heavily co-dependent, as is the case for the panel layout design optimization problem.

As shown in Algorithm 2, Backtracking implements a Depth First Search (DFS) to explore possible solutions. Given grid cells, panel types, and placement rules, the algorithm traverses each cell and attempts to place the largest valid panel from the cell's domain. If the placement satisfies all constraints, the PLACEPANEL() function marks the cells as occupied. If invalid, REMOVEPANEL() reverts the placement and the algorithm backtracks to try the next panel from the domain.

While backtracking ensures completeness, it's limited by exponential growth in the solution space. To improve

Algorithm 2 Backtracking for meta-rectangle packing

Require: Wall dimensions, panels, openings
Ensure: Maximum coverage with minimum panels
Initialize valid wall grid and domains
Apply AC-3 to enforce initial arc consistency
Call BACKTRACK(wall, panels, $x = 0$, $y = 0$, panel_id=1)

Function BACKTRACK(wall, panels, x, y, panel_id):
if $y \geq$ wall.height **then**
  Update best grid and return
end if
if $x \geq$ wall.width **then**
  BACKTRACK(wall, panels, $x = 0$, $y + 1$, panel_id)
  return
end if
if wall.current_grid[y][x] $\neq 0$ **or** $(x, y) \in$ wall.openings **then**
  BACKTRACK(wall, panels, $x + 1$, y, panel_id)
  return
end if
for panel $\in$ SORTBYSIZE(wall.domains[(y, x)]) **do**
  if CONSTRAINTS(wall, x, y, panel) **then**
    PLACEPANEL(wall, x, y, panel, panel_id)
    Apply AC-3 for arc consistency
    BACKTRACK(wall, panels, x+ panel.width, y, panel_id+1)
    REMOVEPANEL(wall, x, y, panel)
  end if
end for
BACKTRACK(wall, panels, $x + 1$, y, panel_id)

Function PLACEPANEL(wall, x, y, panel, panel_id):
for each cell in panel.area **do**
  wall.current_grid[cell] $\leftarrow$ panel_id
end for
wall.panel_count $\leftarrow$ wall.panel_count + 1

Function REMOVEPANEL(wall, x, y, panel):
for each cell in panel.area **do**
  wall.current_grid[cell] $\leftarrow 0$
end for
wall.panel_count $\leftarrow$ wall.panel_count - 1

Algorithm 3 Arc Consistency (AC-3)

Initialize queue with all cells from the grid
while queue is not empty **do**
  $(x, y) \leftarrow$ dequeue(queue)
  if REVISE(wall, x, y) **then**
    for neighbor $\in$ GETNEIGHBORS(x, y) **do**
      Enqueue(neighbor)
    end for
  end if
end while

Function REVISE(wall, x, y):
revised $\leftarrow$ false
for panel $\in$ wall.domains[(y, x)] **do**
  if not CONSTRAINTS(wall, x, y, panel) **then**
    Remove panel from wall.domains[(y, x)]
    revised $\leftarrow$ true
  end if
end for
return revised

efficiency, we implemented AC-3 pre-processing to prune inconsistent panels from a cell's domain after successful placement. As shown in Algorithm 3, AC-3 initializes a queue with all grid cells and iteratively prunes the domains of each cell and its neighbors, removing invalid panel configurations. This maintains local consistency and reduces the search space size.

2.3.4 Complexity Analysis

The backtracking approach remains exponential in complexity as the number of valid grid cells increases. AC-3 improves efficiency by dynamically reducing domain sizes, but its impact depends on constraint density. Two key factors influence performance:

Facade Shape Complexity: Complex geometries increase both the search space and constraint network density, resulting in higher AC-3 processing overhead.

Panel Size Variety: A larger panel dimension set increases domain sizes, creating more potential assignments for backtracking. Though AC-3 prunes infeasible configurations, it cannot eliminate the problem's inherent exponential worst-case growth.

3 Results

The algorithm begins by identifying and importing the JSON files containing the dimensions for a desired facade and its associated openings. These files are then pre-processed to ensure they are aligned either with the (x, z) plane or the (y, z) plane depending on the orientation of the selected facade. For this study, the algorithm is tested on both a facade aligned with the (x, z) plane and another aligned with the (y, z) plane, as depicted in Figure 2.

Following the pre-processing stage, the corresponding 2D points are extracted from the aligned JSON files. These points are used to map the grid onto a cartesian space without loss of any spatial data. This mapping ensures accurate representation of the facade that would serve as an input to the CSP architecture.

Additionally, due to the complexity added by the openings, the algorithm was run independently for two cases: 1) Facades without openings (light solution), and 2) Facades with openings (heavy solution). For both cases, the wall and opening dimensions in the 2D space were scaled to three different units: millimeters (mm), centimeters (cm), and decimeters (dm). This multi-scale analysis highlights the relationship between grid density and computational time. Notably, as the grid density increases tenfold (e.g., transitioning from cm to mm scales), the execution time also increases approximately tenfold, demonstrating the algorithm's scaling behavior.

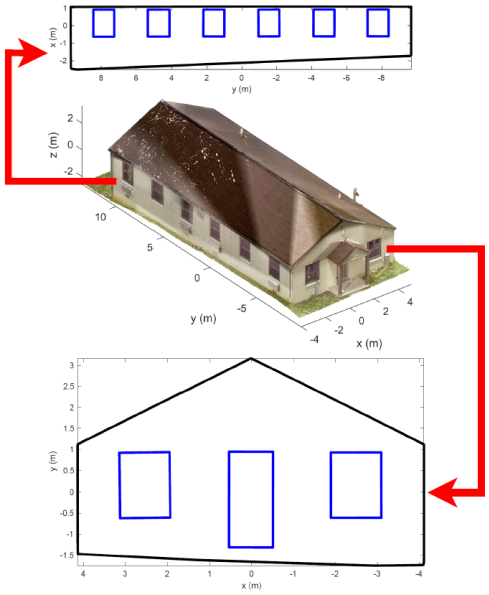


Figure 2. Dimensional facades extracted with Auto-CuBES++ and pre-processed

3.1 Facade without openings

After the initial set up, the inputs for the CSP architecture are set up and initialized. As elaborated in Section 2, the inputs provided to the CSP algorithm are the following:

Grid resolution: determines the scale of each grid cell onto which the facade is mapped. It can take up any value from {10, 100, 1000} depending on the dimensions the facade is scaled to: mm, cm, or dm.

Panel sizes: customized values based on the facade under consideration. The height of each panel is fixed to 2.44 m (8 ft), while the width varies in factors of the grid resolution, typically multiples of 1.22 m (4 ft).

Facade grid: is the 2D cartesian representation of the facade generated by Auto-CuBES.

Following the set up of these parameters, the algorithm tackles the problem of mapping the facade to a grid. In order to adapt the rectangle-packing problem to the meta-rectangle packing problem, the area where the panels can be placed has to be clearly defined. This is achieved via the Shapely package in Python [17]. The 2D points extracted from the JSON files generated by Auto-CuBES are mapped onto a rectangular grid mimicking a rectangular bin. Due to the algorithm initially tackling facades without any opening, the valid grid is initialized to the entire facade area.

After grid initialization, each cell receives a list of available panels. The algorithm evaluates three grid resolutions: decimeters (dm), centimeters (cm), and millimeters (mm). Three panel sizes (8×4 ft, 8×8 ft, and 8×12 ft) are scaled to match each resolution: 24×12, 24×24, and 24×36 dm; 240×120, 240×240, and 240×360 cm; and 2400×1200, 2400×2400, and 2400×3600 mm.

In due course, the backtracking algorithm is triggered to explore potential solutions for all the three grid resolutions. This process iteratively constructs panel layouts by placing the panel with the largest area from each cell's domain. while also systematically retracting and reevaluating the assigned panel positions to maximize coverage. As this problem was modeled as an optimization problem, the final result that is visualized aims to maximize the coverage of the valid grid in terms of cells while minimizing the number of panels used.

3.2 Facade with openings

The setup for this case builds on Section 3.1 but accounts for openings like windows and doors. The wall geometry is read from a JSON file containing the facade's polygonal representation, and its integrity is ensured through a validation and repair process. Similarly, all opening geometries are dynamically loaded from matching files, each representing an opening in the wall, and undergo the same validation and repair procedure to ensure they are well-formed polygons.

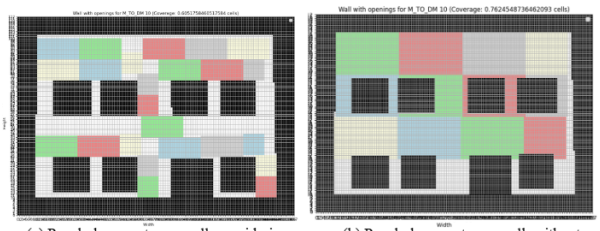
After preparing the wall and opening polygons, the valid area is computed by subtracting the opening polygons from the wall polygon using a geometric difference operation. This process removes areas covered by openings, ensuring panels are not placed over windows or doors. Additionally, cells along the facade's edges are invalidated to prevent panels from extending beyond the wall's dimensions. The resulting valid area excludes both openings and boundary cells, creating an optimized grid for panel placement.

The algorithm is then run for three facades at different grid resolutions, including one from a two-story experimental building at Oak Ridge National Laboratory (ORNL), known as the Flexible Research Platform 2 (FRP2), as illustrated in Fig.3. The other two facades are from a single-story residential building being retrofitted by the Knoxville's Community Development Corporation (KCDC). The two facades chosen from the single-story building cover the cases of facades being aligned with the $x-z$ plane (KCDC 21) and the $y-z$ plane (KCDC 11), as illustrated in Fig.4 and Fig.5, respectively. In these figures, part (a) shows the panel placement with openings, while part (b) assumes no openings, with panels placed to fully cover any openings. These facades were converted into all three grid resolutions being tested and evaluated based on the time taken for initialization, valid grid computation,

Table 2. Result: Panel placement execution time

Dimension	Initializing (sec)	Valid grid computation (sec)	Backtracking + AC-3 (hrs)	Coverage (%)	Number of panels used
FRP2 (with opening)				Excluding openings	
Decimeter (dm)	1.87	0.44	2.38	60.51	10
Centimeter (cm)	69.89	36.33	17.31	54.27	10
Millimeter (mm)	2805.06	2595.82	>30	~55	10
FRP2 (without opening)					
Decimeter (dm)	2.05	1.03	1.4	76.24	12
Centimeter (cm)	13.96	36.65	10.89	75.22	15
Millimeter (mm)	479.21	3392.06	>30	~ 75	15
KCDC 11 (with opening)				Excluding openings	
Decimeter (dm)	0.20	0.14	0.01	25.2	2
Centimeter (cm)	8.84	9.47	0.48	23.06	2
Millimeter (mm)	822.82	1111.53	28.11	22.87	2
KCDC 11 (without opening)					
Decimeter (dm)	0.55	0.51	0.63	57.44	3
Centimeter (cm)	7.26	11.52	9.33	55.54	3
Millimeter (mm)	533.11	1048.32	>30	~ 55	3
KCDC 21 (with opening)				Excluding openings	
Decimeter (dm)	0.71	0.32	0.02	43.8	7
Centimeter (cm)	24.29	29.52	0.76	39.76	7
Millimeter (mm)	1948.51	1929.85	31.2	39.48	7
KCDC 21 (without opening)					
Decimeter (dm)	1.42	0.90	0.22	77.77	6
Centimeter (cm)	18.93	38.09	2.93	69.81	6
Millimeter (mm)	867.78	1806.78	>30	~ 70	6

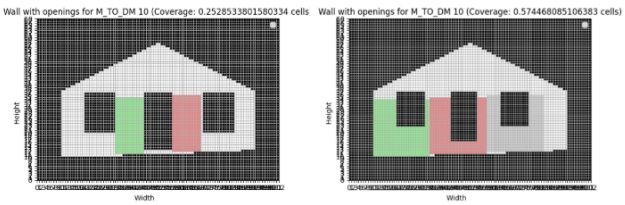
and the execution of the backtracking algorithm with AC-3, as shown in Table 2. The results in Table 2 correspond to the panel placements shown in Fig. 3, Fig. 4, and Fig. 5, with each row in the table corresponding to a specific facade and scenario. The first set of rows corresponds to the FRP2 facade, shown in Fig. 3. The second set corresponds to the KCDC 11 facade, shown in Fig. 4. And the final set corresponds to the KCDC 21 facade, shown in Fig. 5. Each row in the table also presents the execution times, coverage percentages, and number of panels used for the different grid resolutions, reflecting the specific conditions for each facade in the corresponding figures.



(a) Panel placement on a wall, considering openings (b) Panel placement on a wall, without considering openings

Figure 3. Panelized FRP2 with and without openings

The backtracking algorithm evaluates each panel in every cell's domain, leading to exponential complexity as grid resolution increases. At millimeter resolution, both initialization and valid grid computation times rise significantly compared to decimeter and centimeter scales, sometimes exceeding 30 hours even with AC-3 as shown in Table 2. To mitigate this, a 20% sampling rate is applied for constraint satisfaction, balancing efficiency and accuracy—higher rates caused excessive panel overlap, while lower rates added unnecessary overhead without notable accuracy gains.



(a) Panel placement on a wall, considering openings (b) Panel placement on a wall, without considering openings

Figure 4. Panelized KCDC 21 with and without openings

The results from the algorithm were evaluated by focusing on two key attributes: the optimization objectives, such as total area coverage, the number of panels used, as outlined in Table 2 and the execution times, namely the time taken to initialize, compute the valid grid, and

execute the algorithm.

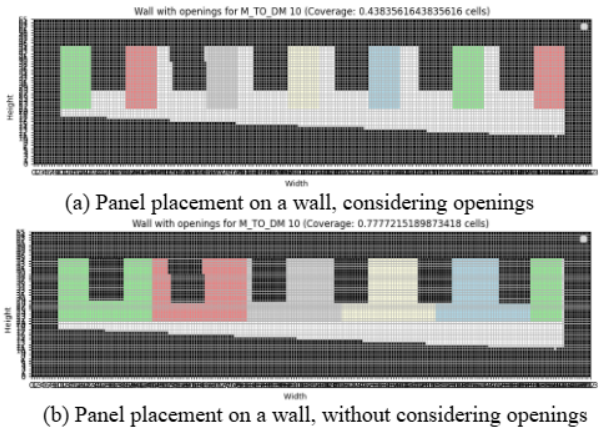
Coverage is the primary objective of the algorithm, with the aim of maximizing the valid area of the grid that corresponds exclusively to the facade. The algorithm ensures that as much of the facade as possible is covered effectively, while excluding non-relevant areas such as the attic as shown in Fig. 4, where wall insulation is not required. Additionally, to improve efficiency, the algorithm was designed to minimize the number of panels used. This dual-objective optimization assessment provides a comprehensive evaluation of the algorithm's capability in balancing these competing goals.

In Figures 3, 4, and 5 coverage is assessed with and without considering facade openings. In part (a), panels avoid overlapping openings based on constraint enforcement (Algorithm 1), resulting in more panels and gaps around these regions. In part (b), openings are ignored, leading to fewer panels and full facade coverage, including areas originally designated as openings.

This experiment revealed that maximizing coverage becomes notably more difficult when openings, such as windows and doors, are present. These openings create irregularities and constraints on the facade, which often result in uncovered regions due to the limitations of panel dimensions and placement options. This challenge is especially pronounced for facades with irregular shapes, which require the algorithm to navigate a more complex solution space. As a result, compromises must be made in either coverage or panel efficiency, as illustrated in the left plots of Fig.3 and Fig.4. These plots clearly show the trade-offs the algorithm faces when balancing the competing goals of maximizing coverage while minimizing the number of panels used, especially in the presence of openings and irregular facade shapes.

A key challenge in the results is the unoccupied panel areas, caused by placement restrictions around openings and fixed panel dimensions. While the algorithm optimizes coverage, irregular facades and panel limitations prevent perfect coverage. Future improvements could include more flexible panel shapes, such as modular panels, or allowing for smaller panel sizes that could better adapt to the irregularities of the facade.

Execution time, as a secondary evaluation metric, scales predictably with grid resolution. As the density of the grid increases, the execution time also increases, as shown in Table 2, reflecting the greater computational effort required to evaluate the finer cells of the grid. This trend highlights the increased computational burden when dealing with higher-resolution grids. In some millimeter-scale cases, the combination of higher grid granularity and the algorithm's complexity has led to execution times exceeding 30 hours, emphasizing the significant computational challenges posed by high-resolution analyses.



(a) Panel placement on a wall, considering openings

(b) Panel placement on a wall, without considering openings

Figure 5. Panelized KCDC 11 with and without openings

In conclusion, the algorithm successfully addresses the dual objectives of maximizing the coverage of the facade and minimizing the usage of the panels, even when faced with the restrictions introduced by openings and irregular facade geometries. Looking ahead, the method could be applied to more complex building structures, such as multi-story buildings with varying facade shapes, by further enhancing the algorithm's ability to accommodate different panel configurations and structural constraints. Future improvements, such as the introduction of more diverse panel sizes or the further optimization of execution times, could further enhance its applicability to a wider range of facade types and building geometries.

4 Conclusion

Our research highlights a significant void in existing systems capable of retrofitting buildings through an end-to-end process—from virtualizing point cloud data to automatically generating panel layouts for each facade to facilitate panel installation. The goal of this study was to bridge the gap between building envelope digital twin generation for panelized systems with automatic tools for prefabricated panel installation using a novel digital tool for automated panel layout design. We propose a three step process to achieve the this: Facade Modelling, CSP Formulation, and Backtracking with AC-3. The proposed algorithm was demonstrated on a variety of facades from a simple rectangle to more complex polygons.

Operationally, the results of the algorithm are also quite robust, with panel layouts being generated in under two or three hours, which is a significant improvement compared to the days typically required for manual creation. Consequently due to the application of a backtracking algorithm for solving the meta-rectangle packing problem,

the user is also presented a variety of optimal solutions to choose from, depending on the optimization factor; rather than a single solution as is the case with other rectangle packing algorithms. While the proposed algorithm demonstrates efficient performance, there is always scope for improvement. Future work involves adapting the algorithm to incorporate a dynamic backtracking algorithm with memorization. This would allow the storage of results of previously calculated sub-problems, facilitating efficient domain pruning whilst reducing redundant computations. Such an approach has the potential to significantly improve the algorithm's efficiency and scalability, paving the way for the development of robust solutions in facade panelization and building retrofitting workflows.

Acknowledgement

This research was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy, and used resources at the Building Technologies Research and Integration Center.

References

[1] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and buildings*, 40(3):394–398, 2008.

[2] Zhenjun Ma, Paul Cooper, Daniel Daly, and Laia Ledo. Existing building retrofits: Methodology and state-of-the-art. *Energy and buildings*, 55:889–902, 2012.

[3] Arva Arsiwala, Faris Elghaish, and Mohammed Zoher. Digital twin with Machine learning for predictive monitoring of CO2 equivalent from existing buildings. *Energy and Buildings*, 284:112851, 2023.

[4] Chukwuka Christian Ohueri, Md Asrul Nasid Masrom, and Taki Eddine Seghier. Digital twin for decarbonizing operating buildings: A systematic review and implementation framework development. *Energy and Buildings*, page 114567, 2024.

[5] Michel Aldanondo, Julien Lesbegueries, Andrea Christophe, Élise Vareilles, and Xavier Lorca. Building insulation renovation: a process and a software to assist panel layout design, a part of the ISOBIM project. In *ISARC 2023-40th International Symposium on Automation and Robotics in Construction*, pages 40–47, 2023.

[6] Bryan P Maldonado, Nolan W Hayes, and Diana Hun. Automatic point Cloud Building Envelope Segmentation (Auto-CuBES) using Machine Learning. In *Proceedings of the 40th International Symposium on Automation and Robotics in Construction*, pages 48–55, 07 2023. doi:10.22260/ISARC2023/0009.

[7] Nolan W. Hayes, Bryan P. Maldonado, Diana Hun, and Peter Wang. Real-time evaluator to optimize and automate crane installation of prefabricated components. In *Proceedings of the 40th International Symposium on Automation and Robotics in Construction*, pages 192–199, Chennai, India, July 2023. doi:10.22260/ISARC2023/0028.

[8] Diana Hun, Peter Lee-shein Wang, Nolan W Hayes, Bryan Maldonado Puente, Philip R Boudreaux, and Stephen M Killough. Real-Time Evaluator to Optimizing Prefab Retrofit Panel Installation, June 27 2024. US Patent App. 18/509,721.

[9] Andrea Lodi, Silvano Martello, Michele Monaci, and Daniele Vigo. Two-dimensional bin packing problems. *Paradigms of combinatorial optimization: Problems and new approaches*, pages 107–129, 2014.

[10] Richard E Korf. Optimal Rectangle Packing: Initial Results. In *ICAPS*, pages 287–295, 2003.

[11] B. P. Maldonado. Automatic Point Cloud Building Envelope Segmentation (AutoCuBES). Computer Software. Registration Number: TXu002412058, Dec 2023. URL `https://doi.org/10.11578/dc.20231214.1`.

[12] Eric Huang and Richard E Korf. Optimal rectangle packing: An absolute placement approach. *Journal of Artificial Intelligence Research*, 46:47–87, 2013.

[13] Andreas Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 172(3):814–837, 2006.

[14] Kevin Tole, Rashad Moqa, Jiongzhi Zheng, and Kun He. A simulated annealing approach for the circle bin packing problem with rectangular items. *Computers & Industrial Engineering*, 176:109004, 2023.

[15] Peter Van Beek. Backtracking search algorithms. In *Foundations of artificial intelligence*, volume 2, pages 85–134. Elsevier, 2006.

[16] Richard J Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In *IJCAI*, volume 93, pages 239–245, 1993.

[17] Sean Gillies, Casper van der Wel, Joris Van den Bossche, Mike W. Taves, Joshua Arnott, Brendan C. Ward, and others. Shapely, August 2024. URL `https://github.com/shapely/shapely`.