Automated Laydown Area Configurator: Rapid Generation & Evaluation of Laydown Area Plan Design Options to facilitate decision making

Nishanth A. Patil[1]

[1]Department of Civil and Environmental Engineering, University of Illinois at Urbana Champaign, Urbana, IL 61801-2352; PH (217) 904-9100; email: napatil3@illinois.edu

ABSTRACT

Construction site utilization planning esp. laydown area has implications for project safety, construction efficiency, scheduling, and budgetary performance of a project. An important aspect not identified in past research efforts are the current practices for laydown area plan development currently used by the construction industry. Therefore, the objectives of this research were to: 1) determine the best practices for laydown area planning; 2) develop a procedure that outlines the laydown area planning process; and 3) show how current Building Information Modeling (BIM) technology can be harnessed for site utilization plan development.

The current research hypothesizes a system that can computationally generate vast numbers of design options, respect project constraints, and analyze for laydown area hygiene goals, can assist the project team and procurement coordinator make better decisions thus reducing overhead costs and time routinely associated with manually planning this process. It also discussed the design and development of a parametric tool built from insights into space planning from architects and computational design, coupled with algorithms for evolutionary systems and computational geometry, to develop an automated computational framework that enables rapid generation and analysis of space plan layouts.

The system as proposed in the paper automatically generates hundreds of design options from inputs typically provided by a construction manager, including a site outline and program document with desired spaces, areas, quantities, and adjacencies to be satisfied. We envision that this novel approach combines analytical tools and heuristics to model the dynamic nature of material management, encompassing commonly-used optimization techniques which use weighted target functions based on rule of thumb. A case study demonstrates the suitability and efficiency of the proposed optimization method in reactive laydown yard management.

THE PROBLEM

Failure to plan the site layout in advance is a prime cause of operational inefficiency and can increase the overall cost of a project substantially. In the absence of a precise site layout plan, the following problems may occur:

a. *Material stacks wrongly located*: Materials arriving on site are off-loaded into what someone guesses to be the correct location. This problem may involve multiple handling of materials to another location. For example:
    - They may be stocked over a drainage line or near the edge of excavation;
    - They are too far from the work area;
    - They are too remote from the hoist or not within the radius of the crane;
    - They impede the smooth flow of work traffic across the site;
    - Their delivery was wrongly phased, and they are not needed until much later in the project;

- They are fragile and susceptible to damage.

b. *Plant and equipment wrongly located*. For example:

- The mixer is inaccessible for the delivery of materials; not enough room for the storage of aggregates;
- Fixed cranes are unable to reach all parts of the works;
- Hoists have insufficient capacity or height to handle the loads or badly located in relation to the floor layout;

c. *Inadequate space allowed*. Where inadequate space is allowed for the stacking of materials or activities:

- Materials may be stacked to high or stacked on roadways causing hazards;
- Working areas may become too cramped or additional areas may have to be allocated with the consequent waste of time caused by having to travel between them.

d. *Office Trailers wrongly located in relation to their effective use*, such as:

- Site office located too near noisy activities such as mixer, or located too near to site roads in dusty conditions, or too remote with insufficient overview of the site;
- Warehouses having inadequate access for loading and unloading or located in insecure area.

Therefore, before moving on to a site, it is necessary to prepare a detailed site plan, depicting the locations of every item of equipment, accommodation, ancillary work areas and materials storage areas.

CURRENT PRACTICE

Currently, the construction manager or site superintendent manually marks up a single site drawing to include major temporary facilities needed on site throughout the duration of the project. They depend on knowledge of years of experience, common sense, and adoption of past layouts in determining positions of temporary facilities on site. But, they cannot keep track of all factors that could affect the selection, location, and interactions of all facilities to be positioned esp. for activities scheduled in the future. In fact, site layout planning is one of the preplanning tasks to be accomplished in a construction project. This task has an interactive relationship with the other planning tasks such as scheduling, selection of construction method, procurement and material planning, manpower and equipment planning, and financial planning. So, it becomes a task as important as other tasks that project managers have to accomplish. A well-planned site including all temporary facilities and utilities lead to: 1) increasing productivity and safety, 2) reducing area(s) needed for temporary construction, and 3) maximizing utilization.

LITERATURE REVIEW

The current research builds on the implicit and explicit domain knowledge and processes in space planning and leverages computation for repetitive design generation and analysis tasks, liberating construction managers from investing time in problem formulation and decision making related to dynamic laydown area planning. We first briefly describe foundational work in data structures and space-planning methodologies, then explain the working methodology via an analogy to space planning employed in architecture design practice (refer Figure 1) by describing the

Hierarchical Space Assignment strategy, a top-down approach from whole to part, and the K-dimensional (K-d) Tree Data Structure, which showcases an efficient data storage, retrieval, and traversal technique. Next, we discuss the splitting strategy to assign departments and programs to the site, and the implementation of a cell grid that enables circulation computation with analysis and scoring of the generated space plans. We then describe the implementation of the methodology in Autodesk Dynamo, and the

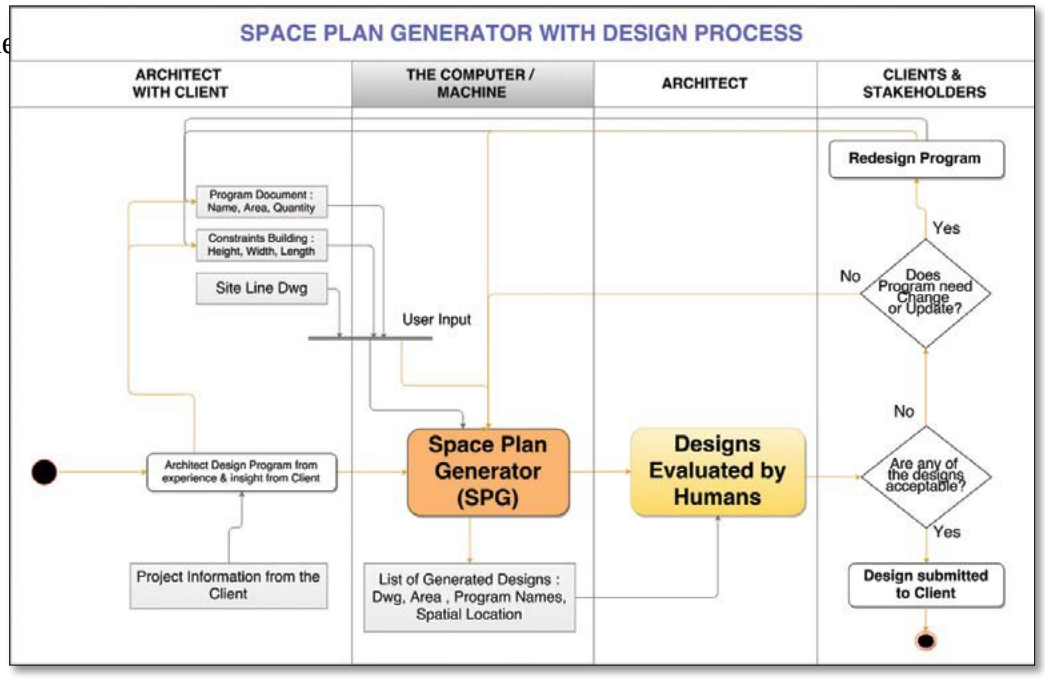


Figure 1 Flow Chart depicting the interaction between architecture design and various stakeholders in a design process

Any space planning process necessitates multiple iterations until design goals are achieved. Consequently, we have implemented a data structure which can rapidly access identical data multiple times and store spatial data in a manner supporting nearest neighbor search, which is necessary to build department and program topology maps for design fitness appraisal. Owing to its spatial partitioning structure, K-d tree

serves our purposes by using nested elements to store data. Nested elements provide a means to retrieve data through bidirectional traversal, top down and bottom up.

K-d trees store and organize data as a set of ‘n’ points in a multi-dimensional space, in a structure of a binary search and partitioning trees. Due to their efficiency and speed, they are primarily applied to nearest-neighbor queries, search algorithms, database applications, and ray-trace methods. Originating in computational geometry, their efficiency arises from the space partitioning algorithm organizing objects in K-d space (Knecht and König 2010). Average running time of an ‘n’ data point database for:

- Insertion – O(log n);
- Deletion of root – O(n(k-1)/k)
- Deletion of any random node – O (log n) (Bentley 1975).

Eastman (1972) automated space generation in two dimensions by implementing decision rules to guide subsequent placement and arrangement of design units. These rules were driven by operators that transformed the state of the design units iteratively to satisfy a set of relationships between them. Jo and Gero (1998) highlighted an evolutionary-design model describing a schema to represent design knowledge capable of providing design solutions for the given problem requirement. Their work highlighted topological and geometrical arrangements of spatial elements tested on a large office layout problem. Though not many variations of the spatial arrangement were highlighted, their work showed the robustness of coupling genetic algorithm-based searches with design workflows to produce good results in space planning. Michalek, Choudhury, and Papalambrosa (2002) presented an optimization model integrating optimization and subjective decision making during conceptual design.

Coupling gradient-based algorithms and evolutionary algorithms, they innovated to include human decision making in the workflow. They implemented topology optimization algorithms on top of geometry optimization algorithms. The results were automated space plans but limited in variation. Nassar (2010) presented new findings in graph theory with direct implications in space planning problems. He described architectural space plans as simple, connected, labeled planar graphs, and elaborated on the relevance of finding a rectangle dual for every planar graph to increase solution space. This work outlined a tool for architects to generate space plans. Realizing spatial relationships as planar graphs with nodes as rooms and edges as adjacencies, it claimed that the proposed model could provide a truly exhaustive set of potential designs. However, this model was limited to two-dimensional space plans only.

Boon et al. (2015) employed genetic algorithms to generate 3D space stacking, respecting input adjacency requirements by the user. Their algorithm evaluates space plans based on adjacency constraints to minimize the total distance of all interconnected programmatic elements. They prioritized the program spaces based on practice expertise and user input, which helps discard unrealistic design solutions. The algorithm stacks spaces in three dimensions, distributing program elements over multiple floors and sometimes unnecessarily complicating architectural space layouts.

Some relevant research in automated space plan generation comes from game design, which requires extensive 3D environments at architectural scale. Lopes et.al (2010) generate floor plans for different classes of buildings, many with connected floor levels, offering limited control to the designer for functional constraints. One of the salient features of this system is the grid- based strategy for placing and growing rooms,

which generates building zones, followed by room areas, constrained by adjacency and connectivity. Marson and Musse (2010) implement a squarified treemap algorithm (previously used to represent hierarchical information graphically) to compartmentalize the input space into different zones or regions. These zones are organized into a hierarchy that satisfies design goals and site constraints and are visualized into a square tree map to generate various floor layouts. Their work excels at building sophisticated circulation networks. After the rooms are site located, a circulation network graph is built to understand which rooms are connected or disconnected from each other. They use an A* algorithm to traverse the connectivity graph and find shortest paths to access spaces from the lobby.

METHODOLOGY ADOPTED

The Laydown Area Configurator has distinct components orchestrated to generate and analyze space plans. It generates layouts as closed polylines representing each space type (either the department or the program element), with the circulation network represented as colored poly surfaces for any generic architectural design problem.

Hierarchical Space Assignment

Our approach is hierarchical (refer Figure 2). Program elements are spaces to fit on the site. Certain types of program elements can be clustered to form departments. The recursive algorithm first places the department on site and then programs within departments, finally placing circulation within and between departments.

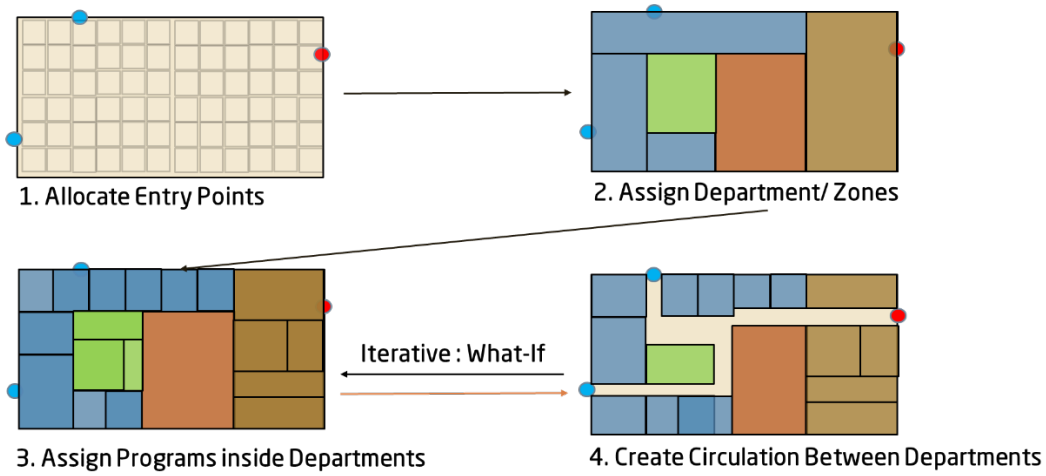


Figure 2 The Hierarchical Space Planning approach adopted

K-d Tree Data Structure

The proposed space-assignment algorithm uses a K-d data structure dividing the K-d space by partition planes perpendicular to one of the coordinate axes (refer Figure 3).

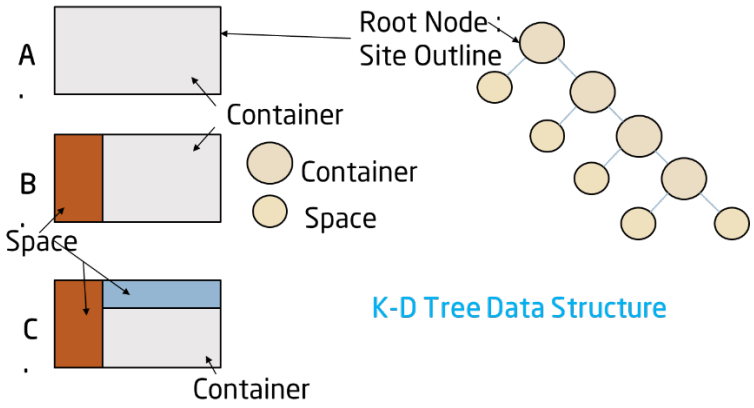


Figure 3 Space assignment algorithm adopted based on K-d data structure binary tree

Conventionally, the median or average of the point coordinates in the database is calculated in the split dimension. Site space is the root and the first node after the spatial split, dividing the point set in two. All points whose coordinates are smaller than the split value reside in the tree's left branch, while the other points are relegated to the right branch. This step is repeated until reaching a threshold depth of the K-d tree.

Similar to an approach to Bentley 1990, each generated node is assigned a space or region, each of which can contain child nodes. Bentley 1975 corroborates that K-d Trees efficiently search and traverse data sets to create spatial plan partitions.

Space data trees implement a modified version of K-d data structures, where the point set is split by a line dividing a monolithic space into two, using an algorithm to allocate program elements at site locations. For example, a programmatic requirement to locate patient rooms on the site periphery or the need of an entry lobby on the site's west side. Any region on the site unassigned to a department or program element is termed a 'container,' while any region allocated to a department or program is termed a 'space.' The root of the data tree is the container (the site), with its left child the region of the site to the left of the splitting line. The remaining region becomes the right child of the root and the current container. The algorithm recursively repeats the operation for each node until all spaces are stored in the tree. After each split of the container, the left node becomes a space node, and the right node becomes a container node. The direction of the split can be flipped from horizontal to vertical alignment at every iteration, similar to Shneiderman's 1992 slice and dice techniques, depending upon the aspect ratio of the container and the program elements awaiting allocation.

Space and Container Node Data

After the split to represent site location, every space node contains data including: Space Node ID, Parent Node ID, Left Child Node ID, Right Child Node ID, Split Line, Department ID Assigned, Cells Allocated, and Polyline representation.

Every container node retains data above except for Department ID. As each node stores links to its parent and child nodes, the space data tree can be traversed upwards and downwards (refer Figure 3).

Benefits of using Space Data Tree are elucidated below:

a. Provides fast and efficient data storage to represent spatial data.
b. Supports nearest neighbor searches, which help build department topology maps and appraise the efficiency of a layout by representing department neighbors and adjacency.
c. Finds neighbors for departments and program elements and shared edges between them, with each Space and Container Node storing the splitting line. This allows the computation of circulation networks between and within departments.
d. Can randomly ascertain circulation paths between any two spatial locations in the space plan, aiding in space plan appraisal through key metrics such as nurse travel routes and distances, patient flow paths, fire egress routes, etc., in a healthcare facility.

Splitting Strategies

The current research deployed several custom methods to split a polyline into two or more by a dividing line, including:

- Split by Distance: Creates a splitting line from a given point on a polyline by a specified distance or by a specified line orientation. Returns two polylines on a successful split.

- Split by Ratio: Splits a polyline into two polylines by a supplied ratio and direction.
- Split by Area: Ensures split polylines meet area requirements as specified by the user or the algorithm, employing a brute force splitting strategy by iteratively applying the split by distance method, altering the distance variable after each iteration.
- Split by Line: Requires the user or the algorithm to provide a splitting line when there is an existing splitting line for further splits.
- Split Recursively to meet Minimum Dimension: A recursive split strategy adapted to split a single polyline into multiple polylines until the minimum dimension of each polyline meets an acceptable width or length constraint specified by the user. Employing a slice and dice technique, after each split the split direction is toggled between horizontal and vertical. This strategy arrives at interesting spatial patterns with an accept- able level of architectural rationality. The listed methods divide the input site polyline into individual departments containing program elements. Method outputs coupled with department and program information are assigned to the space data tree to organize them for further computation.
- Split by Offsetting Polyline Points: A computationally faster strategy generates two polylines by shifting the points of the parent polyline, avoiding more expensive algorithms such as line-poly intersection or ordering points in a list to rebuild a polyline as used in other split strategies.

Cell Grid Underneath the Spaces Assigned

From an input site outline, the system builds a bounding object containing the site polyline (refer Figure 4). The algorithm supplies the bounding box with points on an orthogonal grid, with spacing specified by the user. A site outline test determines the contained points used to build each cell. A collection of cells is termed a grid object, employed to compute the shortest path results between program elements, placement of doors and windows, and similar problems. Grid objects are used to build the cell neighbor matrix, which tracks the neighbors of every cell in three dimensions. The benefit of such a matrix is in its capability to efficiently traverse the site.

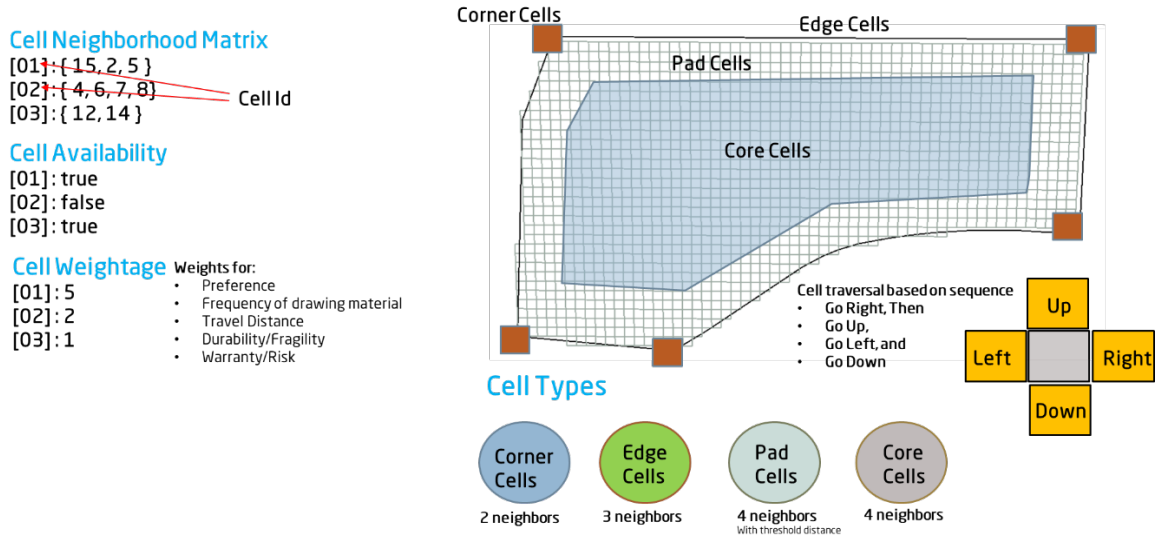


Figure 4 Cell storage and neighborhood bounding based on ID, Cell neighbor matrix, Cell weights etc.

Cell Neighbor Matrix

Each cell in the grid object possesses a location ID. The neighbor matrix of a cell is a list of lists, which stores identifiers of all neighboring cells. Neighboring cells are traversed in a fixed order of right, up, left, and down. A cell identified as 0 might have cells 05, 08, 02, 15 stored in its neighbor matrix row, with 05 identifying the right-side cell, 08 denoting the upside cell, cell 02 on the left side, with 15 identifying the downside cell. -1 indicates a cell lacking any neighbors. The matrix provides a means

to traverse the site from one corner to another to find specific locations, such as doors and exit routes (refer Figure 4).

Cell Weighting for Specific Metrics

Each cell is given a custom weight to gauge different aspects of the project, such as acoustic performance, daylighting, and site constraints. Weighting influences the allocation of spatial locations for a certain program. For example, if the northwest side of the site is conducive for daylighting, then cell weights can be increased for that zone to arrive at an appropriate distribution bias.

Shortest Path to Find Doors / Access Points

The cell neighbor matrix also helps find the shortest path between site regions by implementing Dijkstra's Algorithm (Wang 2012). Specific cells are marked as doors or windows, with the objective of pathfinding to discover the shortest path to those cells from a given program element.

Polyline Boundary

This algorithm finds the maximum sized orthogonal polyline boundary of any input polyline having both orthogonal and non-orthogonal angles between its sides (refer Figure 5). To locate the bounding polyline, the algorithm first identifies border cells by traversing the cell neighbor matrix. A cell is identified as a border cell if it's either a corner cell with two neighbors or an edge cell with three neighbors. The algorithm rebuilds the cell neighbor matrix from the border cell, finds the lowest leftmost cell in the border cell list, and subsequently traverses the cells. Initially, traversal attempts a rightward search, proceeding on failure through successive tries up, left, and down. The centroid of each visited cell is stored in a point list, and the cell is marked as

unavailable for further visits, preventing infinite search loops. Upon traversing every cell, the complete centroid list will be used to build the polyline boundary.

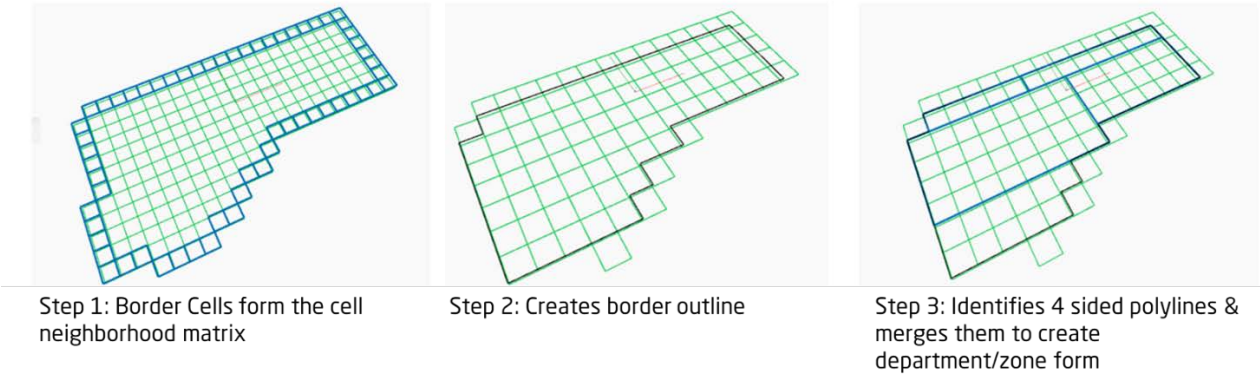


Figure 5 Polygon Boundary algorithm depicting the orthogonal boundary for any input arbitrary site outline

In addition to the design Configurator, an analyzer component validates the efficacy of the generated space plan with respect to the original goals. Some analysis types implemented include gauging the percentage of program elements fitted to the site outline in comparison to specified requirements, determining the quantity of day-lit rooms with external views, and determining circulation efficiency. Kindly refer Figure 6 and 7 to understand the working methodology of the Laydown Area Configurator further.

IMPLEMENTATION

The prototype employs an Autodesk Dynamo Package written in C# as a library of ‘zero touch’ custom nodes (Helsberg et al. 2010), which helps construct the space plan Configurator as explained below (refer Figure 7). The Dynamo graph is composed of two components, a ‘Configurator’ and an ‘Analyzer.’

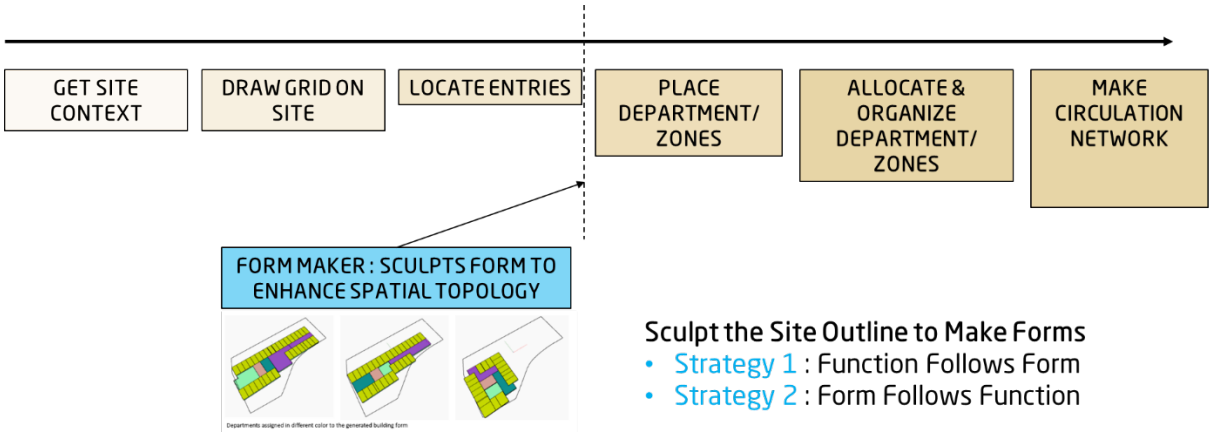


Figure 6 Flow diagram depicting the working methodology of the Laydown Area Configurator

After every graph evaluation, the 'Configurator' yields distinct plan layouts rendered as a list of polyline geometries. The 'Analyzer' gauges the fitness of each design option to input goals and constraints.

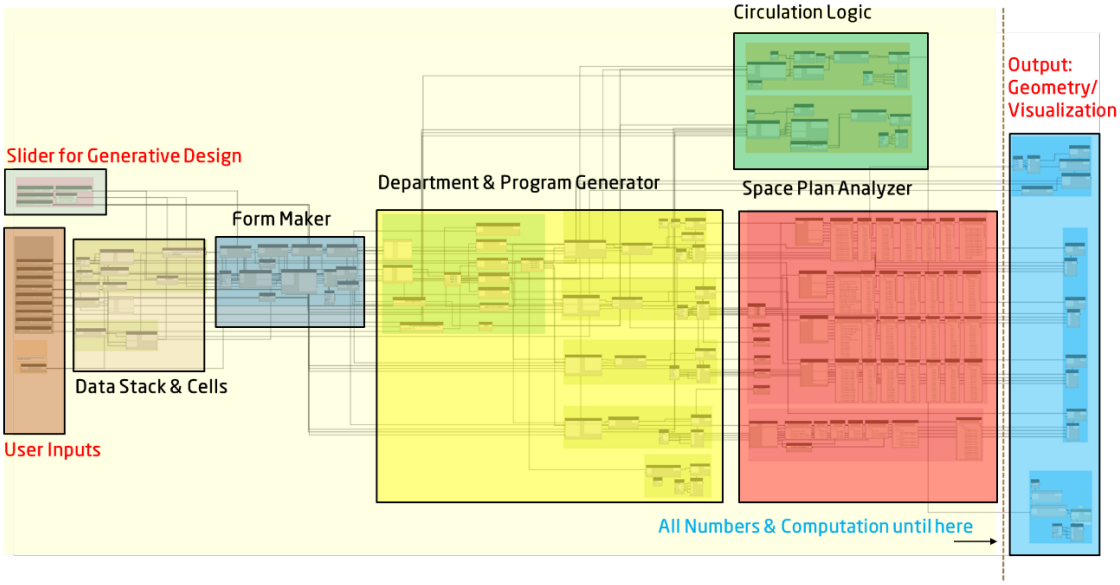
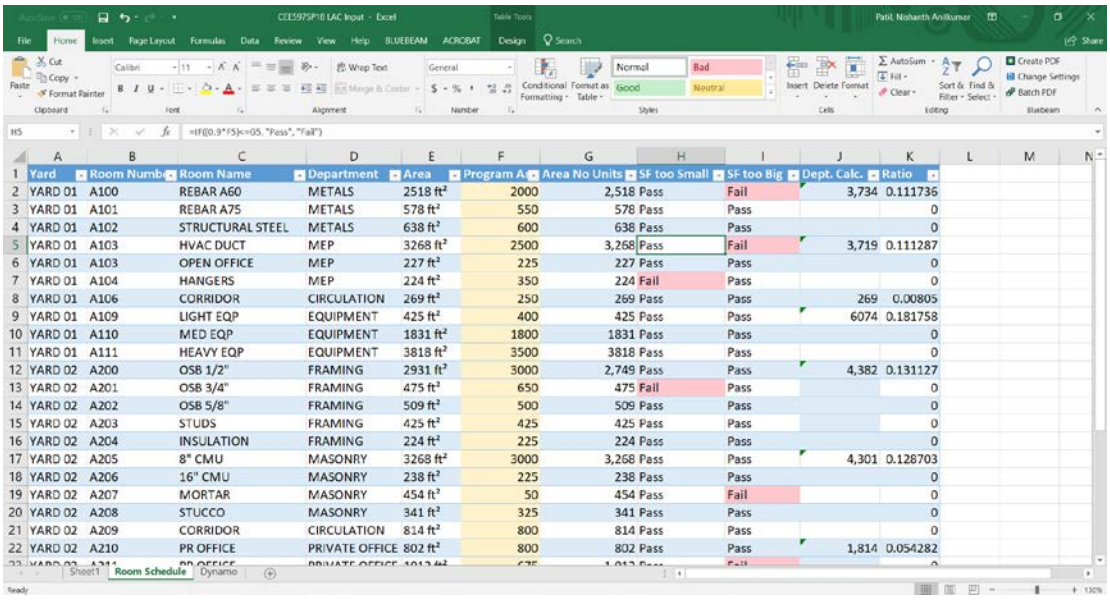


Figure 7 Computation and information flow diagram of the Autodesk Dynamo Graph

The prototype proceeds as follows:

Step 1: User Inputs

The graph has input requirements in the form of a site outline '.sat' file and program document '.csv' file (refer Figure 8). The '.csv' should contain information for each program element to be fit onto the site, with each element possessing the attributes, namely: ID, name, department or zone or material, quantity, area or dimensions (width x depth), preference value, and adjacency list. Preference value for each program element specifies the order in which the material will be placed onto the site. The adjacency list is a comparison chart to which the material zone topology map will adhere while allocating space/area elements on the site. Other user inputs include corridor width, the aspect ratio for circulation and program elements, grid cell dimension, etc.



Yard	Room Numbe	Room Name	Department	Area	Program A	Area No Units	SF too Small	SF too Big	Dept. Calc.	Ratio
YARD 01	A100	REBAR A60	METALS	2518 ft²	2000	2,518	Pass	Fail	3,734	0.111736
YARD 01	A101	REBAR A75	METALS	578 ft²	550	578	Pass	Pass		0
YARD 01	A102	STRUCTURAL STEEL	METALS	638 ft²	600	638	Pass	Pass		0
YARD 01	A103	HVAC DUCT	MEP	3268 ft²	2500	3,268	Pass	Fail	3,719	0.111287
YARD 01	A103	OPEN OFFICE	MEP	227 ft²	225	227	Pass	Pass		0
YARD 01	A104	HANGERS	MEP	224 ft²	350	224	Fail	Pass		0
YARD 01	A106	CORRIDOR	CIRCULATION	269 ft²	250	269	Pass	Pass	269	0.00805
YARD 01	A109	LIGHT EQP	EQUIPMENT	425 ft²	400	425	Pass	Pass	6074	0.181758
YARD 01	A110	MED EQP	EQUIPMENT	1831 ft²	1800	1831	Pass	Pass		0
YARD 01	A111	HEAVY EQP	EQUIPMENT	3818 ft²	3500	3818	Pass	Pass		0
YARD 02	A200	OSB 1/2"	FRAMING	2931 ft²	3000	2,749	Pass	Pass	4,382	0.131127
YARD 02	A201	OSB 3/4"	FRAMING	475 ft²	650	475	Fail	Pass		0
YARD 02	A202	OSB 5/8"	FRAMING	509 ft²	500	509	Pass	Pass		0
YARD 02	A203	STUDS	FRAMING	425 ft²	425	425	Pass	Pass		0
YARD 02	A204	INSULATION	FRAMING	224 ft²	225	224	Pass	Pass		0
YARD 02	A205	8" CMU	MASONRY	3268 ft²	3000	3,268	Pass	Pass	4,301	0.128703
YARD 02	A206	16" CMU	MASONRY	238 ft²	225	238	Pass	Pass		0
YARD 02	A207	MORTAR	MASONRY	454 ft²	50	454	Pass	Fail		0
YARD 02	A208	STUCCO	MASONRY	341 ft²	325	341	Pass	Pass		0
YARD 02	A209	CORRIDOR	CIRCULATION	814 ft²	800	814	Pass	Pass		0
YARD 02	A210	PR OFFICE	PRIVATE OFFICE	802 ft²	800	802	Pass	Pass	1,814	0.054282

Figure 8 .csv file input to the Dynamo Graph

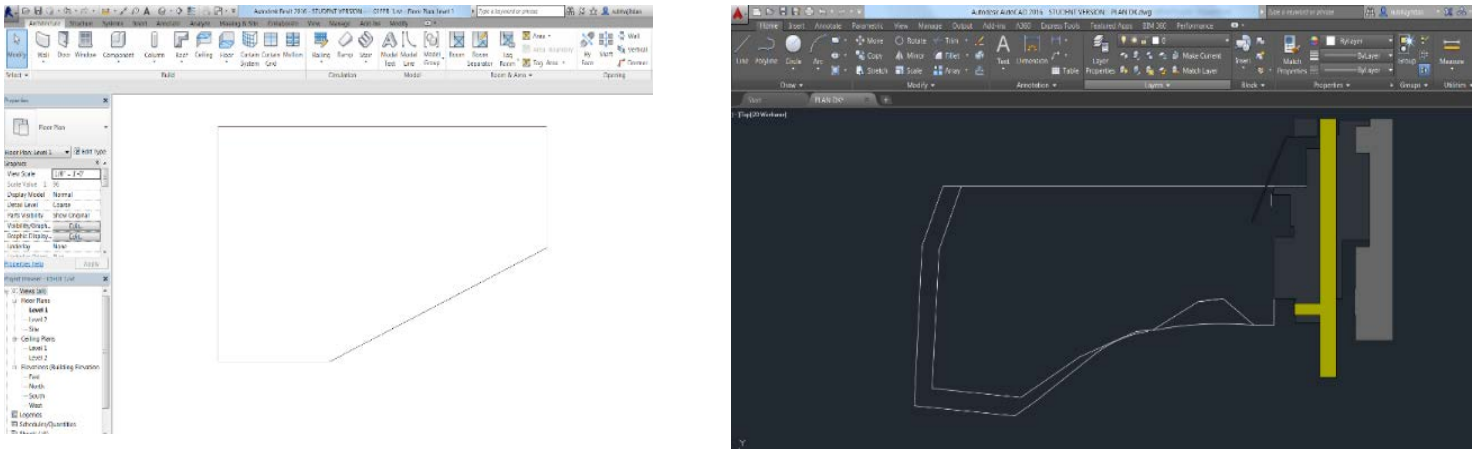


Figure 9 Site Boundary as .sat file input to the Dynamo Graph

Step 2: Cell Grid Information

The graph builds the cell grid on the site and the initial cell neighbor matrix, which are used repeatedly in the workflow explained above.

Step 3: Data Stack

This step extracts input information from the program document and builds a Department Data and Program Data class object. The data stack node sorts and stores the department and program data object based on the preference values from the program document. Any computation or analysis regarding the spatial assignment of program elements is done with the aid of the data stack of departments and programs.

Step 4: Form Generation

The form maker node constructs the orthogonal laydown area and zone outline for an arbitrary site outline (see Figure 10 and 11) traversing the cell grid via the cell neighbor matrix, implementing two strategies to pack programs.

Strategy 1: Function Follows Form

This strategy splits the orthogonal laydown area outline into sub-polylines until each polyline has exactly four sides, and subsequently merges a random number of such four-sided orthogonal polylines together to get the laydown area form. One advantage of this approach is that the user can set a percentage of total site area for the laydown area to occupy, resulting in building perimeters of specific site coverage. The merge operation employs the Polyline Boundary algorithm as explained above.

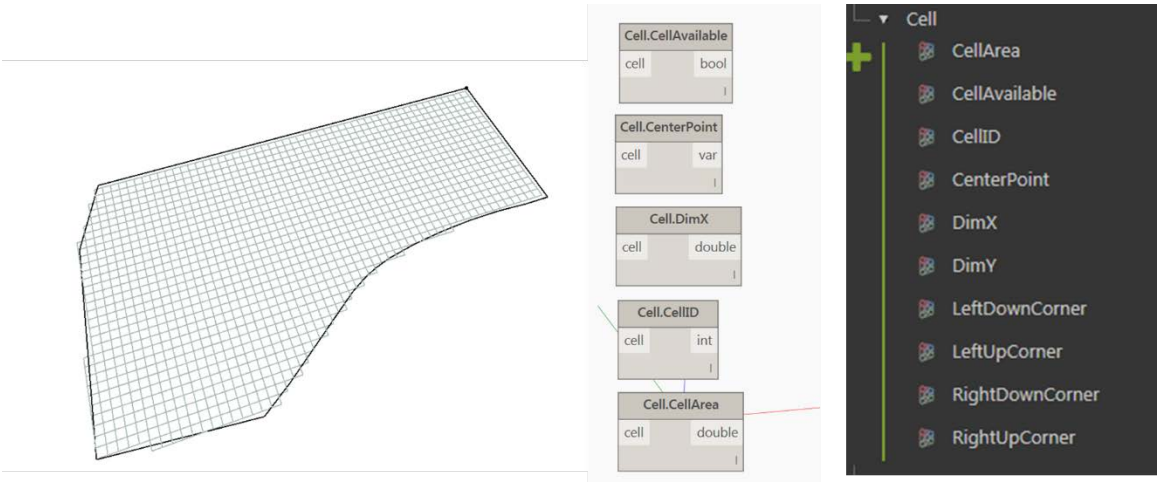


Figure 10 Snippet depicting conversion of site boundary into cells for further processing

Strategy 2: Form Follows Function

Unlike Strategy 1, Strategy 2 develops the building outline while placing department and program elements. After each department/zone polyline placement, the algorithm evaluates adherence to site constraints and design requirements. The algorithm will only proceed to subsequent departments/zones after constraints and requirements are satisfied or until reaching an allowed quantity of trials. When area requirements are satisfied for

departments/zones, the algorithm discards any leftover waste space, merging the outer lines of the department polylines to arrive at the final building outline.

For both of the strategies, custom functions remove single or multiple notches from the site outline using minimum edge distance, thus refining the laydown area form and increasing the number of design choices for the user.

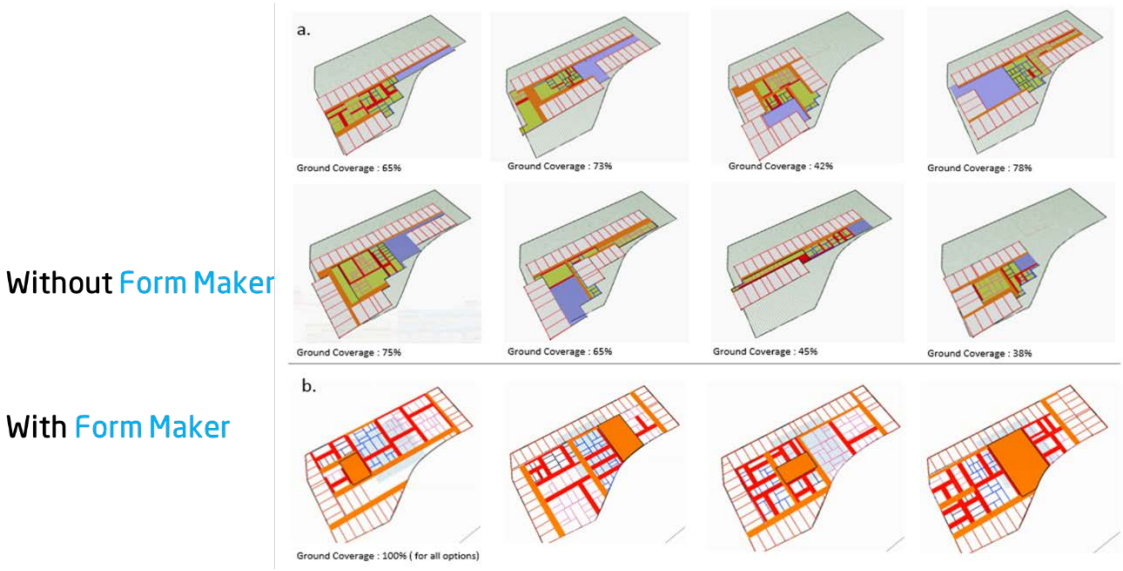


Figure 11 Snippet depicting simultaneous development of laydown area boundary and placement of departments/zones

Step 5: Department Placement and K-d Space Data Tree for Department

This step assigns departments to the site based on preference values from the user and simultaneously builds the space data tree. The iterative process is appropriately integrated with the form maker from the previous step, depending upon which form-making strategies are adopted and how well design goals and constraints are satisfied. Refer Fig 12 for understanding the visual interface incorporating these user assigned weights.

Step 6: Program Placement

This step assigns program elements inside each department, as prioritized by user goals, and updates the grid object by assigning each cell a certain program type and updating each cell's weight. Refer Fig 13 through Fig 18 for understanding the visual interface and some of the arrangements possible.

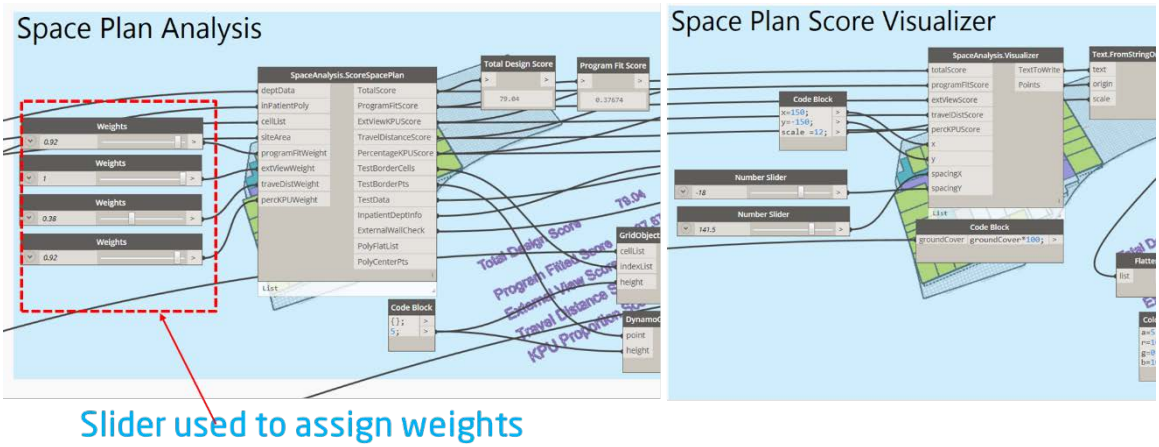


Figure 12 Snippet depicting the slider arrangement developed to ensure control using user assigned weights and preferences

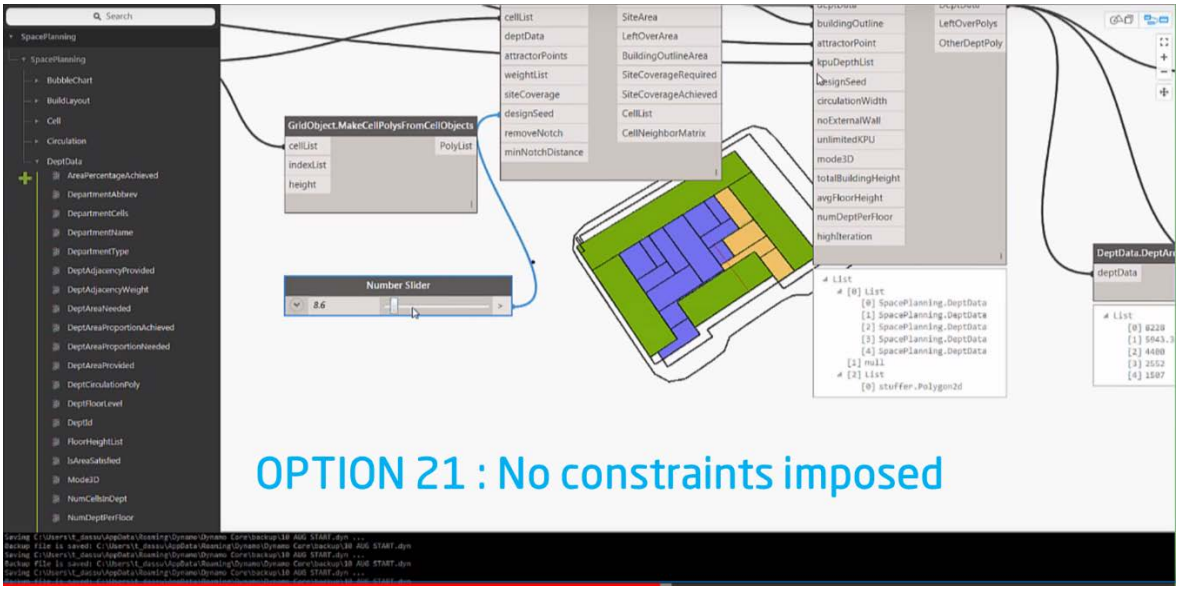


Figure 13 Snippet depicting the space plan generated with no constraints imposed

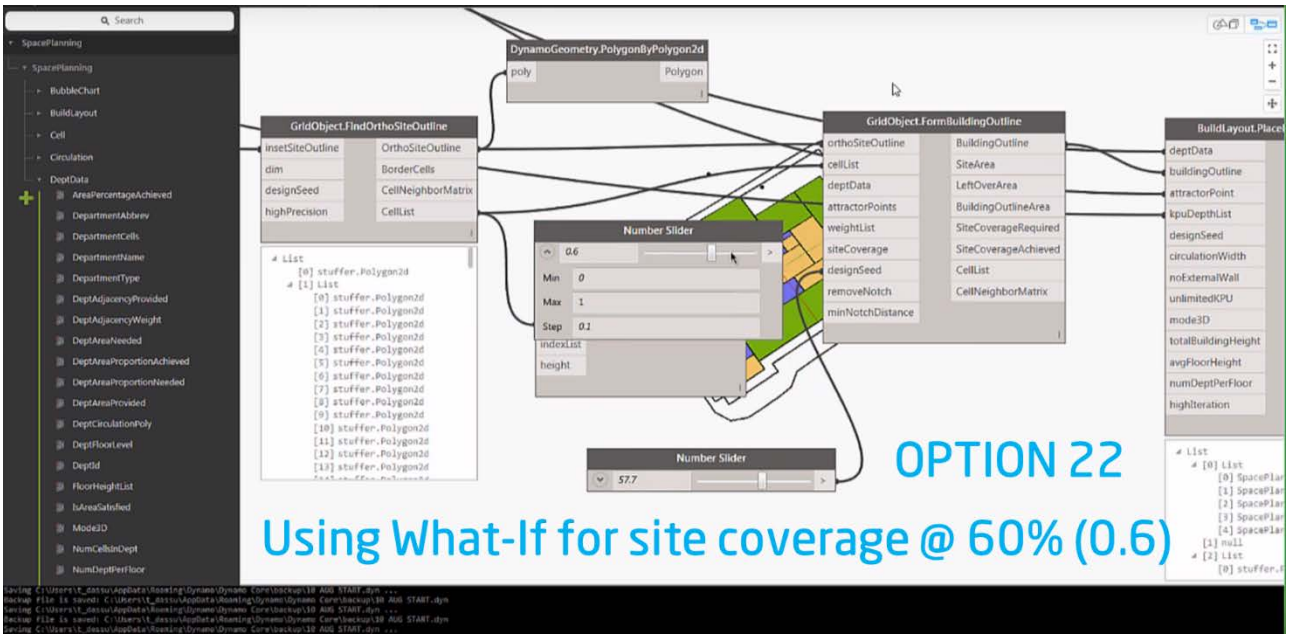


Figure 14 Snippet depicting the use of a slider to fix the site coverage at 60% of the total site area to generate and check for viable arrangements

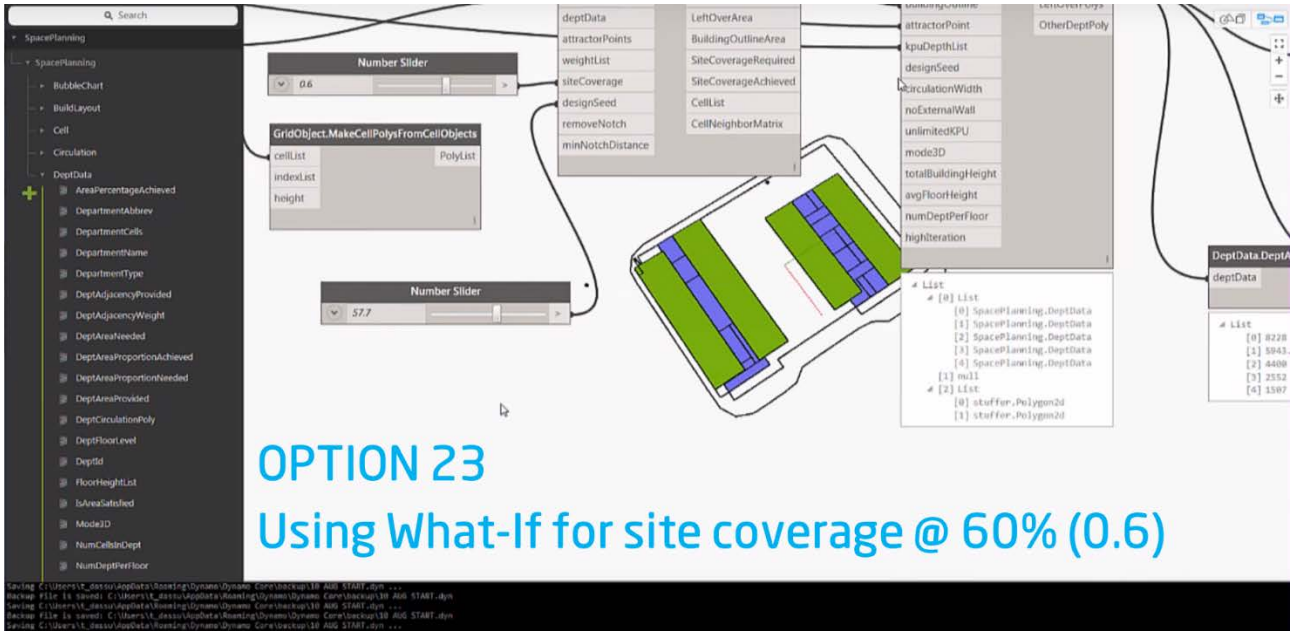


Figure 15 Snippet depicting one of the viable arrangements at 60% site coverage

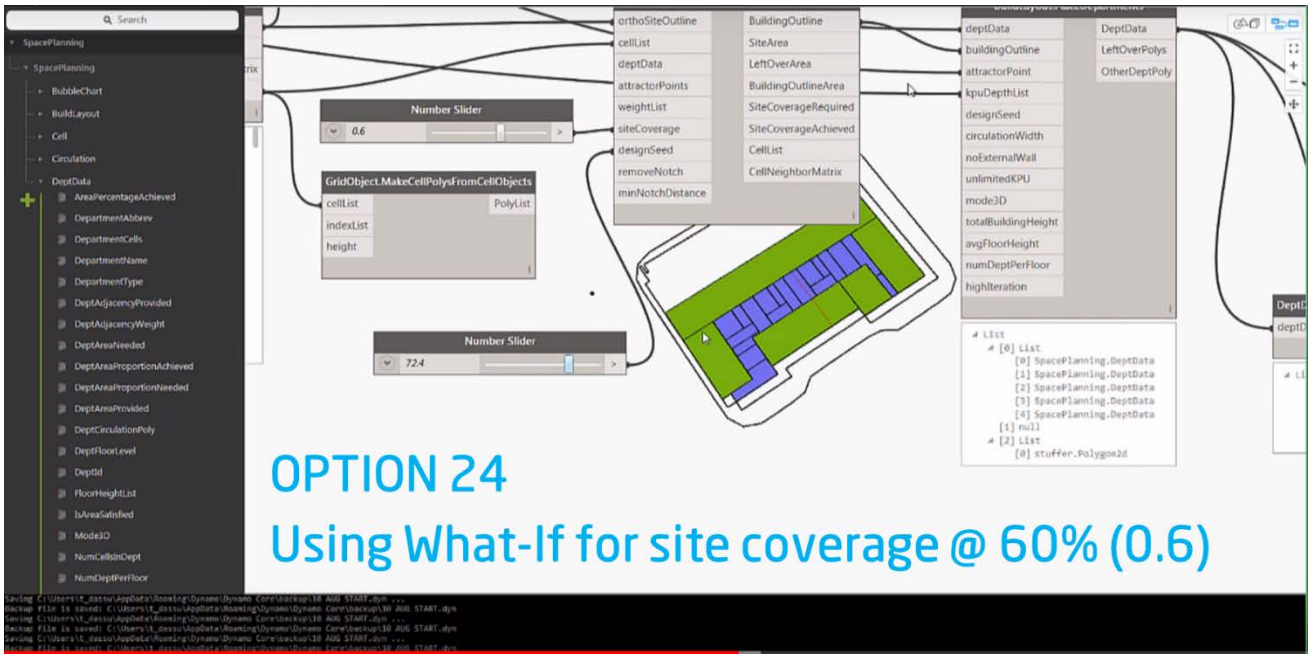


Figure 16 Snippet depicting one of the viable arrangements at 60% site coverage

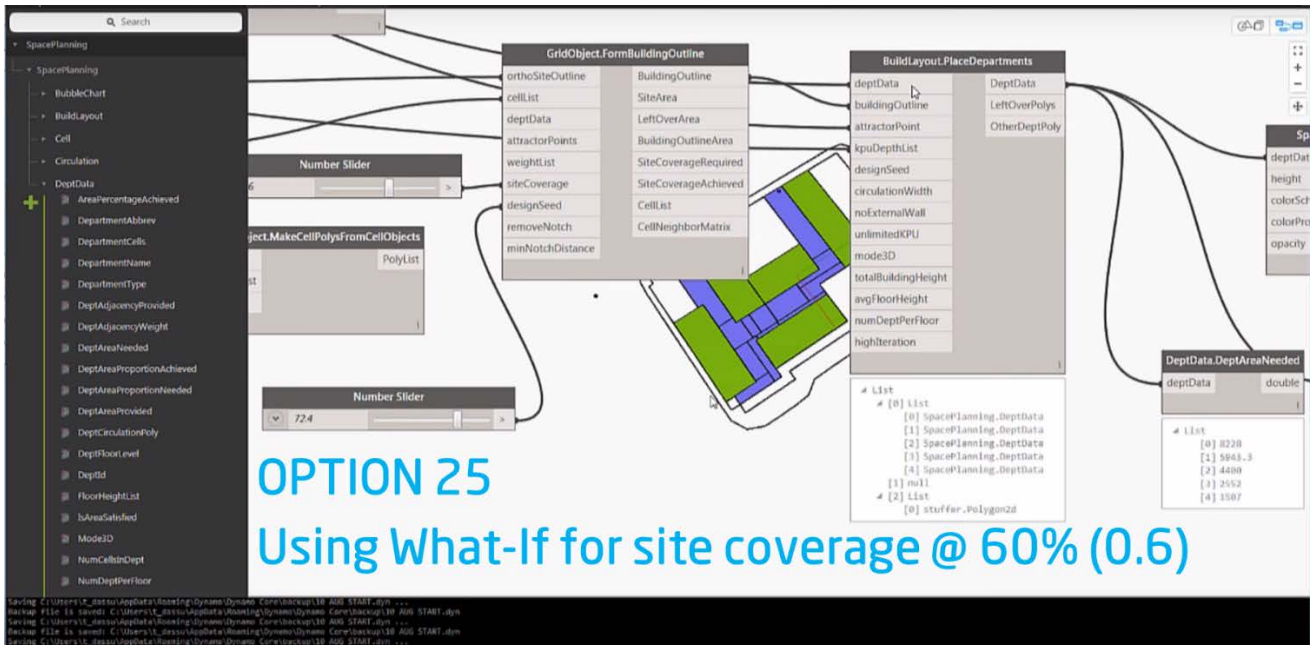


Figure 17 Snippet depicting one of the viable arrangements at 60% site coverage

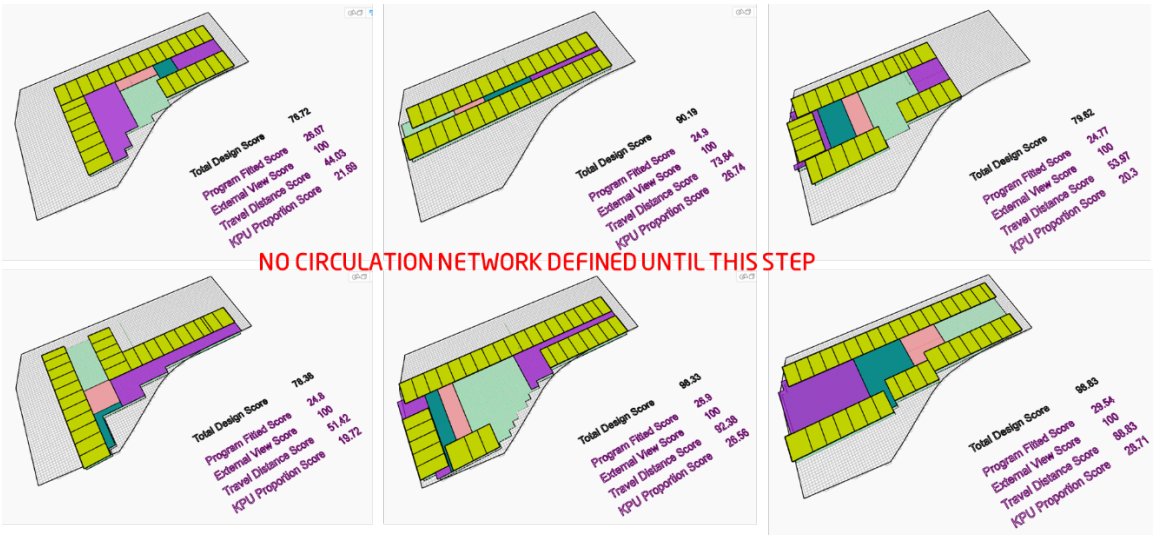


Figure 18 Snippet depicting a collection of program/zone arrangements as an output of the program Configurator node

Step 7: Circulation Computation

Circulation computation discovers circulation networks between departments/zones and subsequently finds circulation networks between program elements by using K-d data structure referenced above. Shared edges between departments and program elements form the initial circulation network. Next, a circulation redundancy check, with the aid of the grid object, is deployed to select only those edges in the network needed to access all spaces. Further, it employs pathfinding algorithms with the aid of cell neighbor matrix to discover the shortest paths within avail- able circulation pathways between points, and places doors and windows along the discovered path. Grid Object also accounts for those spaces which do not get any access and places additional network lines to render them accessible from the main public space in the layout (Mirahmadi and Shami 2012). Refer Fig 19 through Fig 22 to explore some of the arrangements generated.

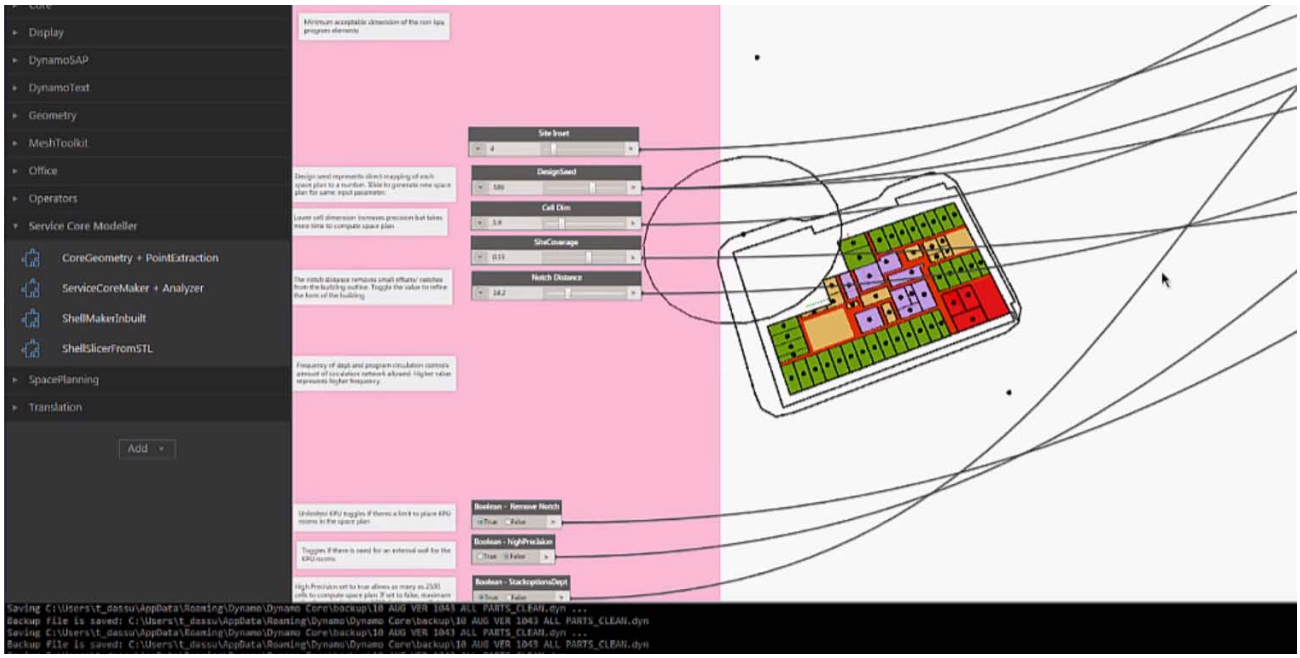


Figure 19 Snippet depicting the arrangement generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider

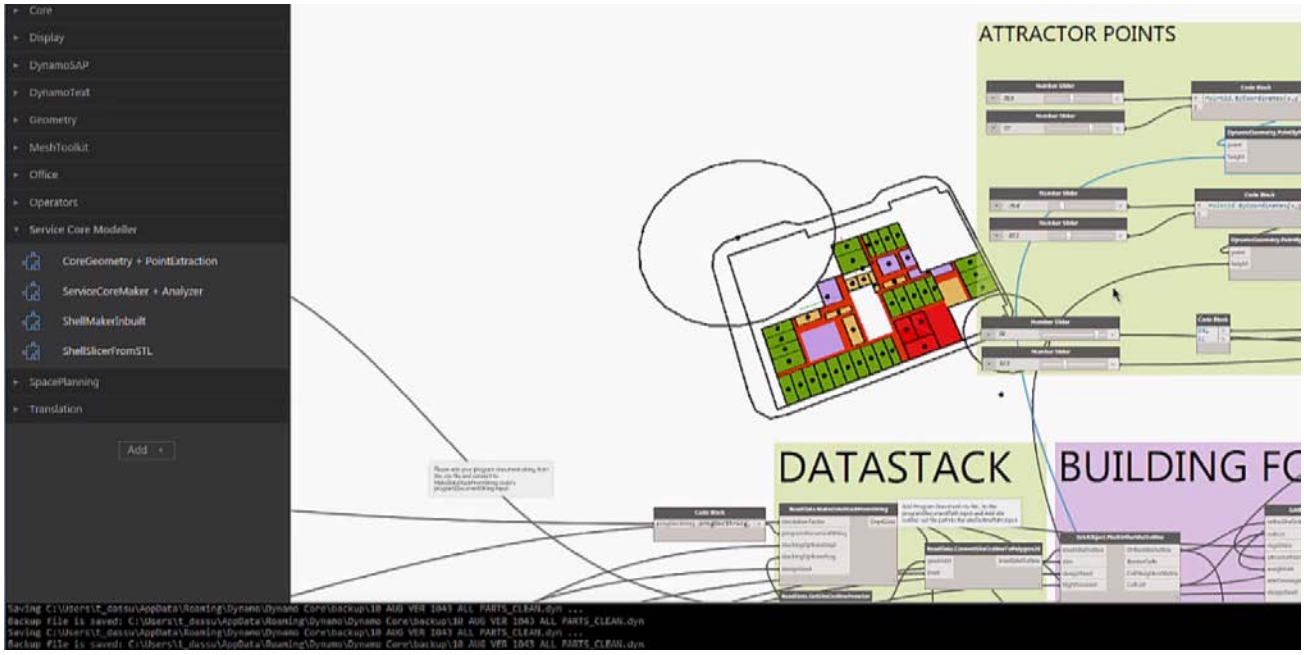


Figure 20 Snippet depicting the arrangement generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider

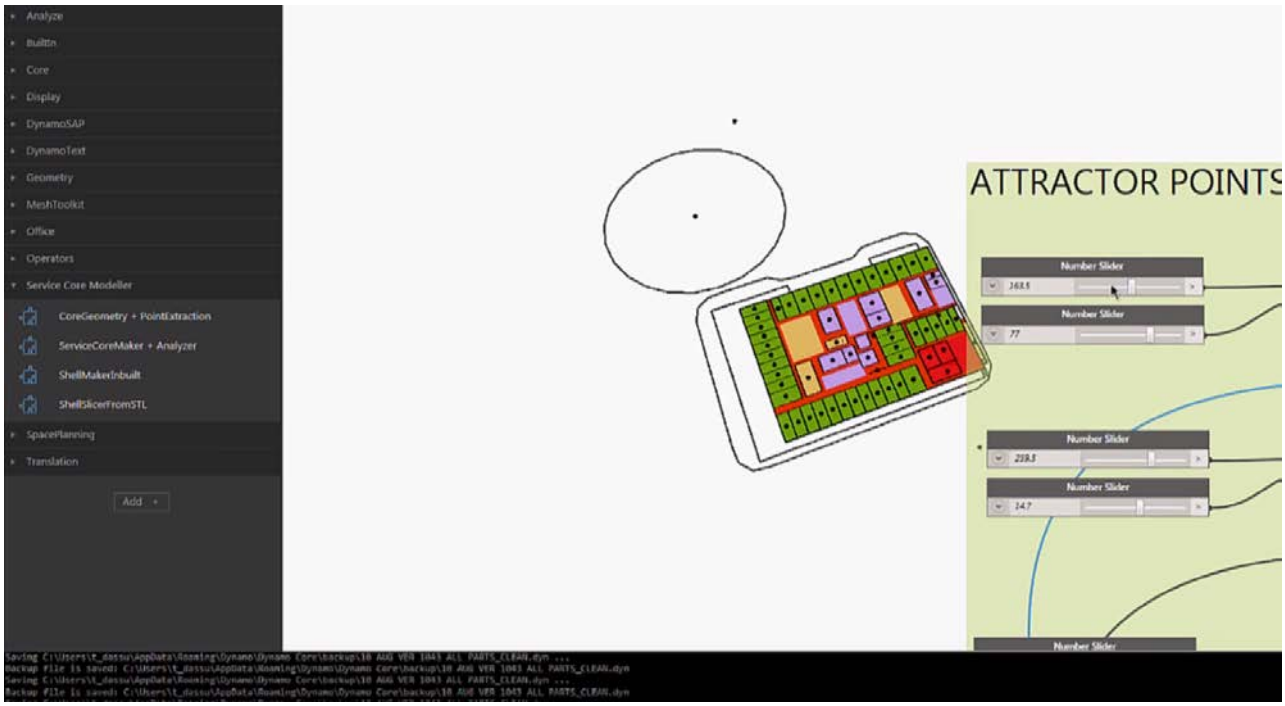


Figure 21 Snippet depicting the arrangement generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider

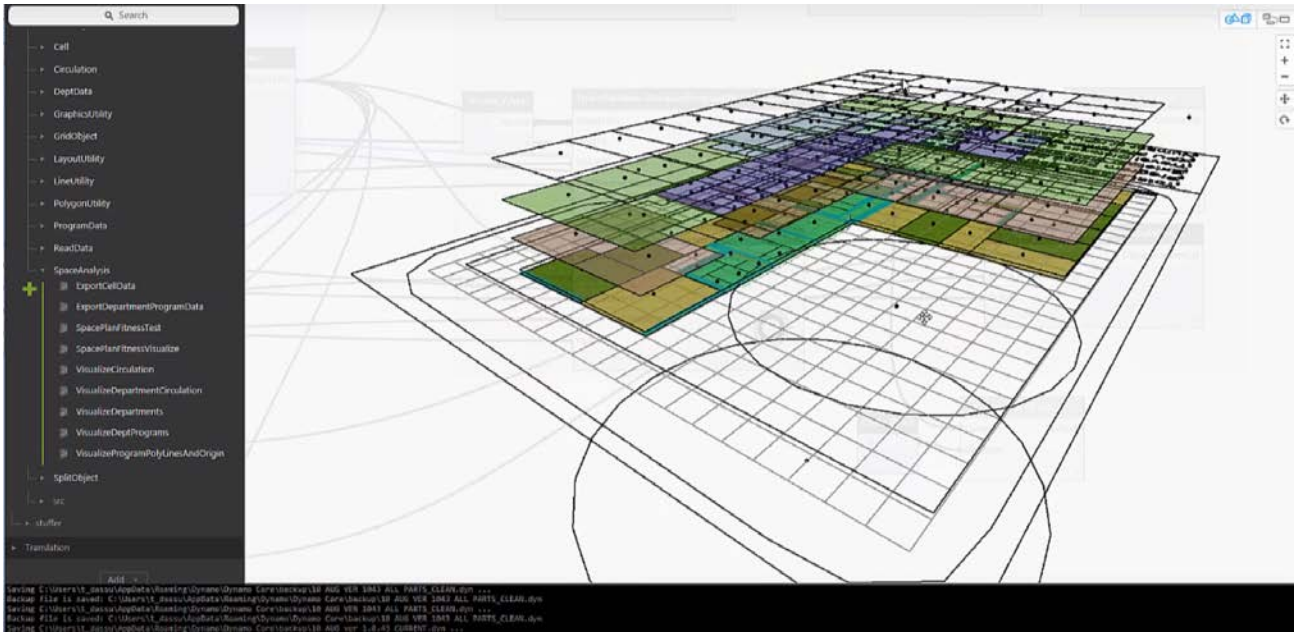


Figure 22 Snippet depicting comparison by superimposing of the arrangements generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider

Step 8: Space Plan Analytics

Evaluates the generated design based on pre-defined metrics as summarized in Fig 23 through Fig 25.

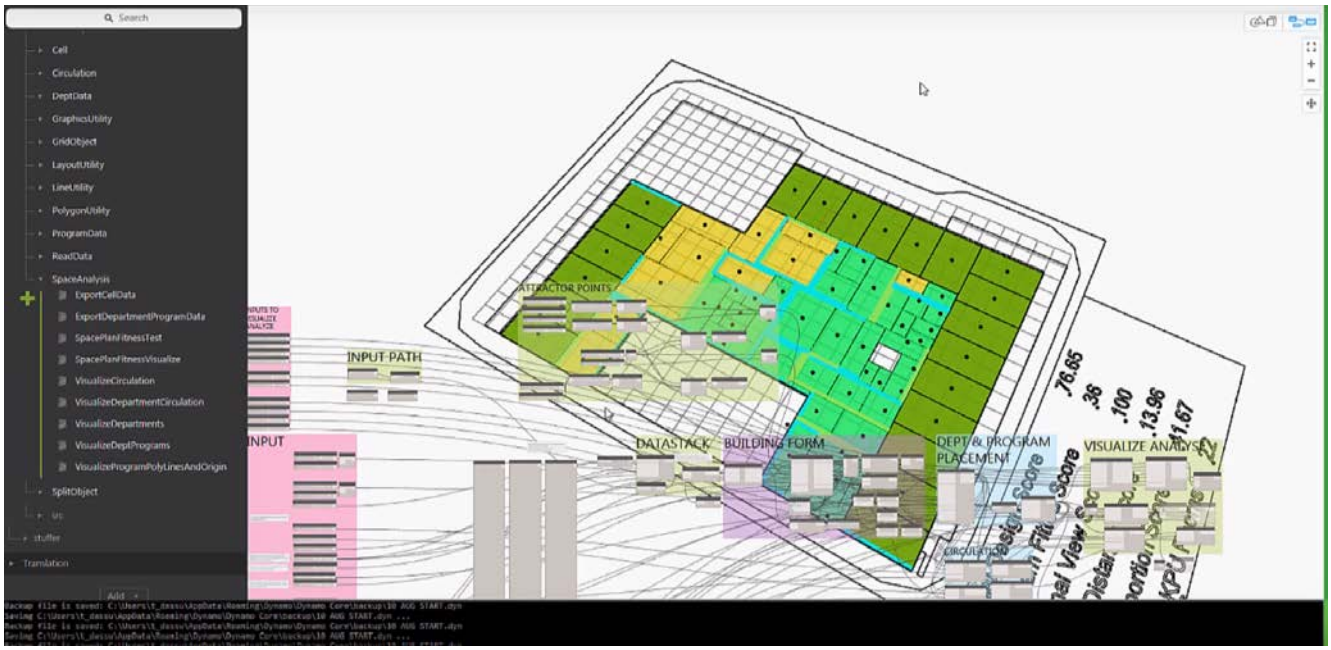


Figure 23 Snippet depicting one of the arrangements generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider along with performance metrics

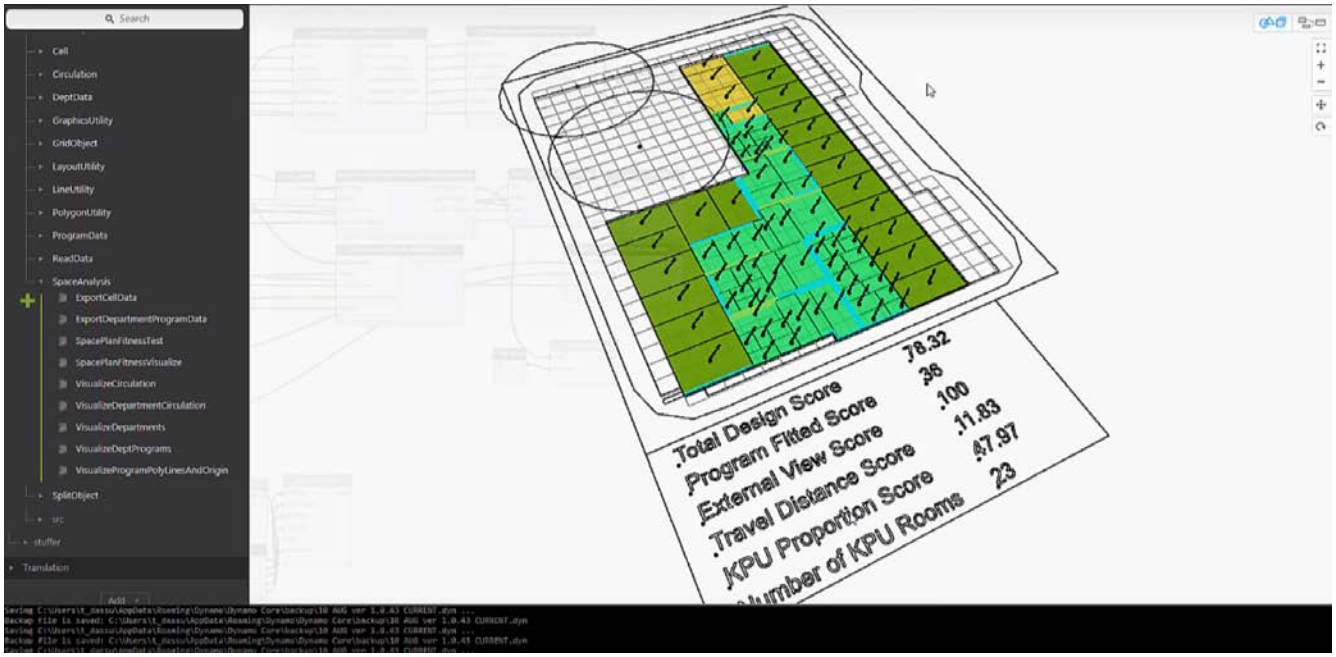


Figure 24 Snippet depicting one of the arrangements generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider along with performance metrics

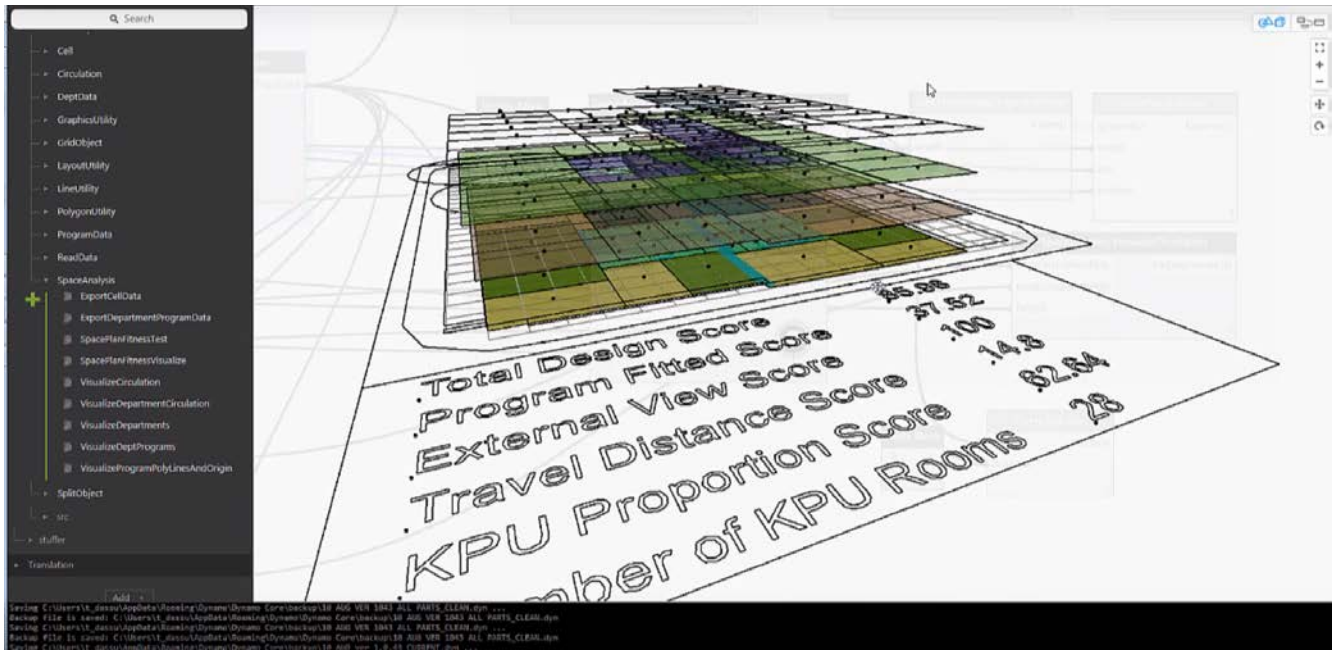


Figure 25 Snippet depicting comparison by superimposing of the arrangements generated after incorporating the circulation and adequacy logic along with user assigned weights on the slider along with performance metrics of least favorable arrangement

Step 9: Space Plan Output / Geometry Rendering

One of the salient features of the Laydown Area Configurator is that it maintains a distinction between geometry and computation, with geometry only rendered at the process conclusion for visualization. Until step 8, no geometry is rendered on screen or temporarily saved in memory, which significantly speeds the production of results. Since geometry is not used before this step, custom methods are included in determining line intersections and line/polygon intersections, removal of duplicate geometries, determine point containment, merge polylines, etc., and made available as ‘zero touch’ Dynamo nodes.

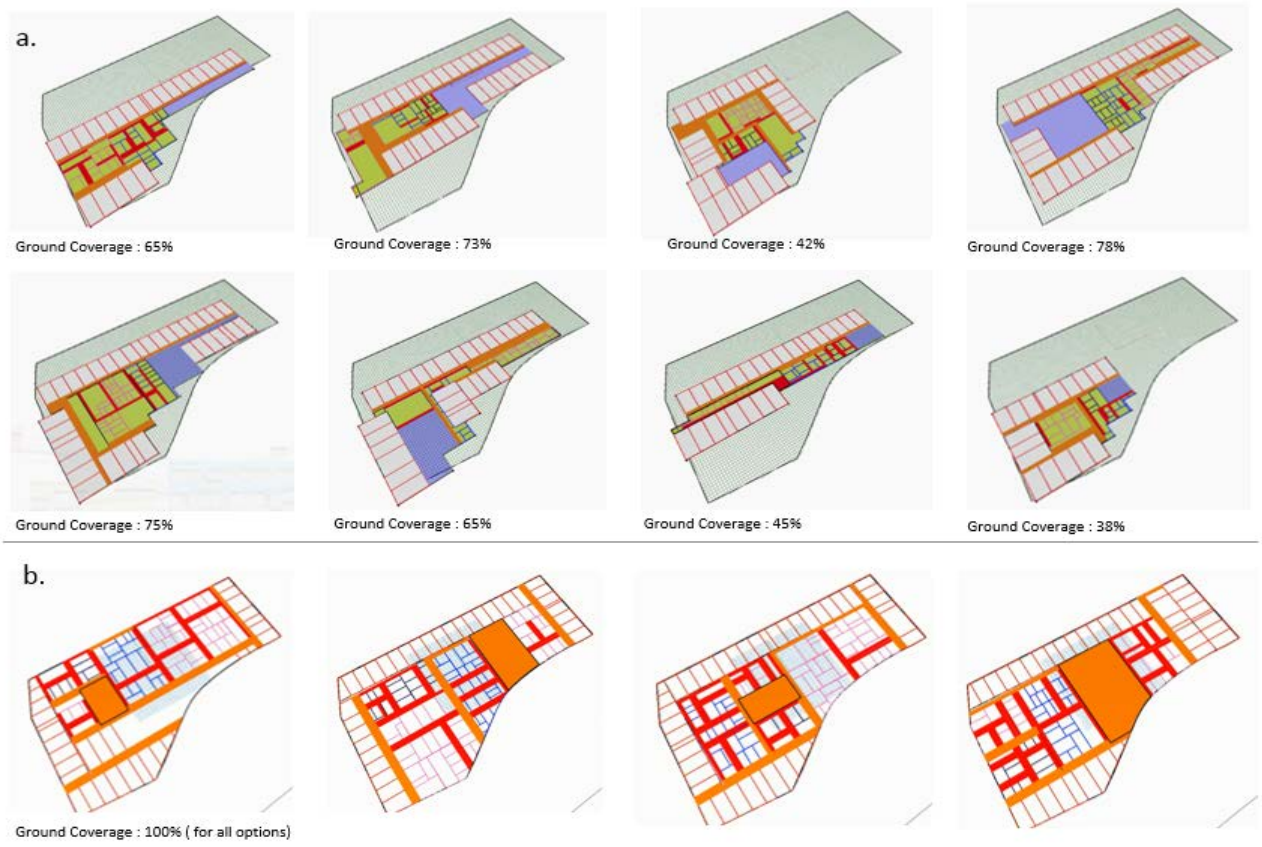


Figure 26 Snippet depicting comparison by % surface area covered of most favorable arrangements based on program requirements and user assigned weights

Step 10: Storing and Scoring Generated Designs

Generated designs are stored as a collection of cells, where each cell stores information about its assigned department or zone. Cells store space plan analytic information, such as distance to entrance/exit, distance from circulation areas, etc., and these metrics help determine the overall scored success of a space plan. Scoring conveys to the user the success of design options relative to input goals and constraints. A dynamic dashboard was created in Microsoft Power BI, and hence provide interactive visualizations with business intelligence capabilities to create reports and aid in decision making, The dashboard (refer Fig. 27) created for the current study is fed data from the room schedule through an excel spreadsheet.

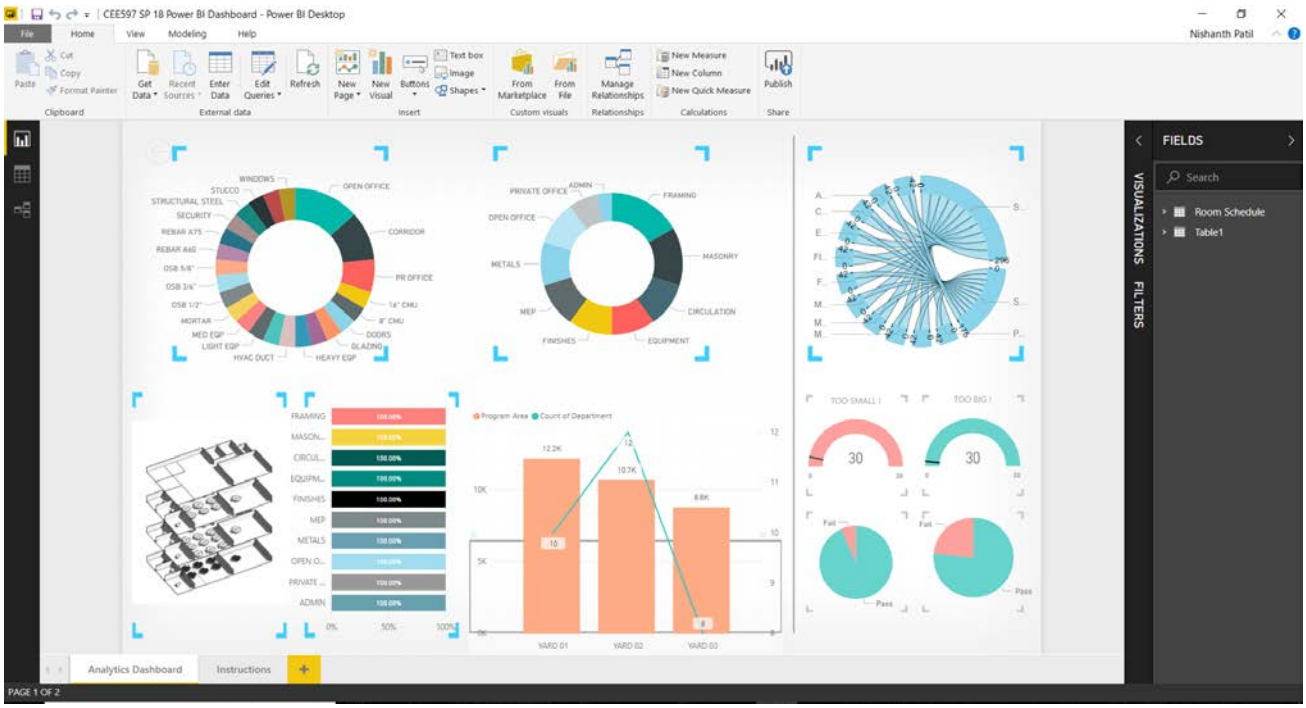


Figure 27 Microsoft PowerBI Dashboard for analyzing output of the laydown area configurator

Specifically, if one wants to analyze the distribution of space in yard 01 on a particular day, then they simply need to click on yard 01 (refer Fig. 28) to understand a host of data ranging from distribution of space categorized on the basis of material category, room or size and also be capable of understanding the adjacency logic and performance of the design in terms of number of spaces which are too small or large compared to their ideal requirement.

Similarly, if one wants to analyze the distribution of space in allocated to a material class, say metals across yards on a particular day, then they simply need to click on metals (refer Fig. 29) to understand a host of data ranging from distribution of space categorized on the basis of location of yard, room or size and also be capable of understanding the adjacency logic and performance of the design in terms of number of spaces which are too small or large compared to their ideal requirement.

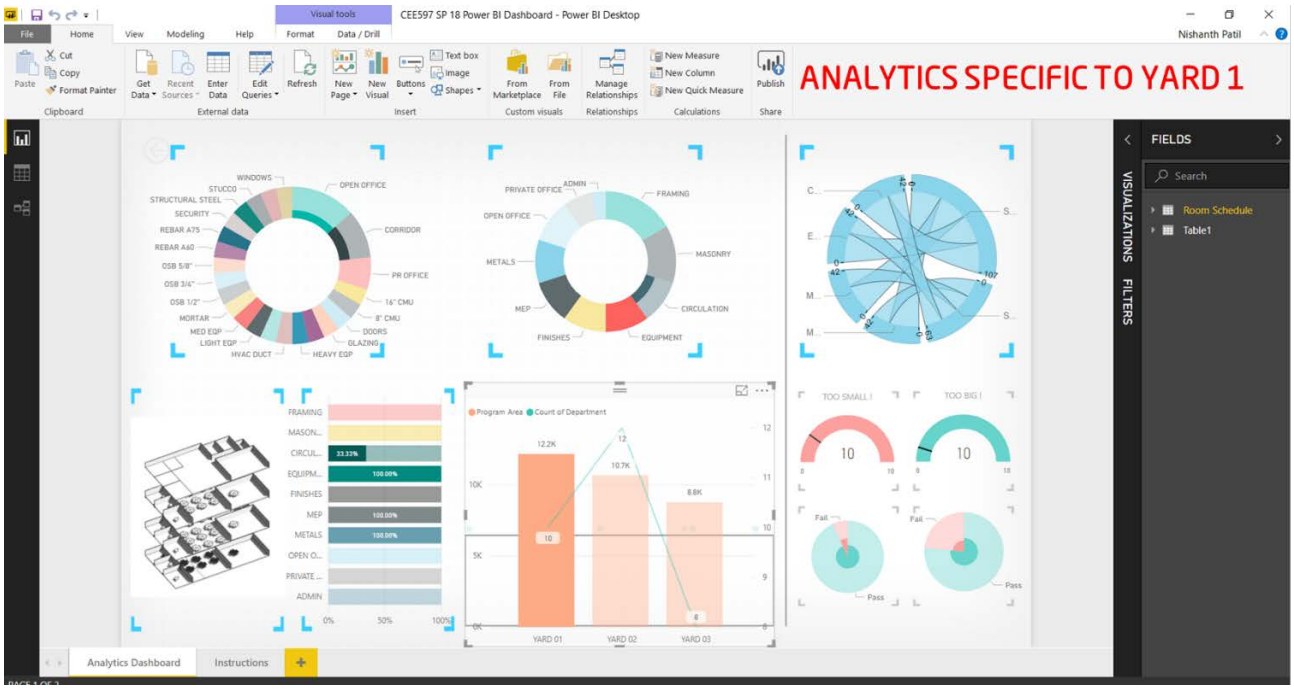


Figure 28 Microsoft PowerBI Dashboard for analyzing output of the laydown area configurator specific to Yard 01

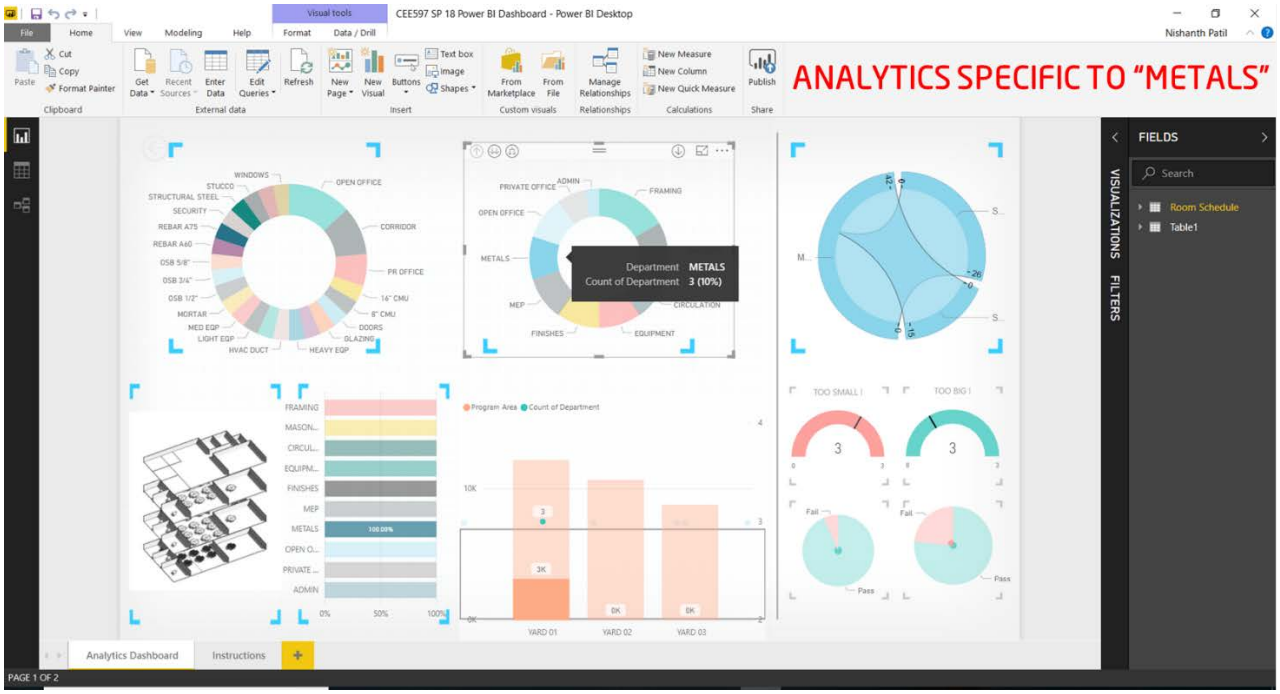


Figure 29 Microsoft PowerBI Dashboard for analyzing output of the laydown area configurator specific to Metals

USE CASE AND RESULTS: IRREGULAR SHAPED BOUNDARIES

The robustness of the algorithm was tested on an irregularly shaped land boundary. This is important as most laydown areas are often created in a part of the project site which is created as the remainder of the space allocated for important and necessary program requirement and hence the probability of the laydown area being irregularly shaped is very high. In essence, laydown areas are always subject to dynamic and reactive planning and seldom is a pre-defined space allocated towards storage. With site constraints and specific client goals, such as maximizing ground coverage, minimizing travel distance (Rechel, James, and Martin 2009), maximizing connectivity to the area under construction, and minimizing view impedance from fluctuating supply-demand curves, this simulation is a valuable test case to understand how a goal-driven design workflow can be automated using generative design strategies. The results of running the laydown area configurator on such a site is depicted via Fig. 30 through Fig 31

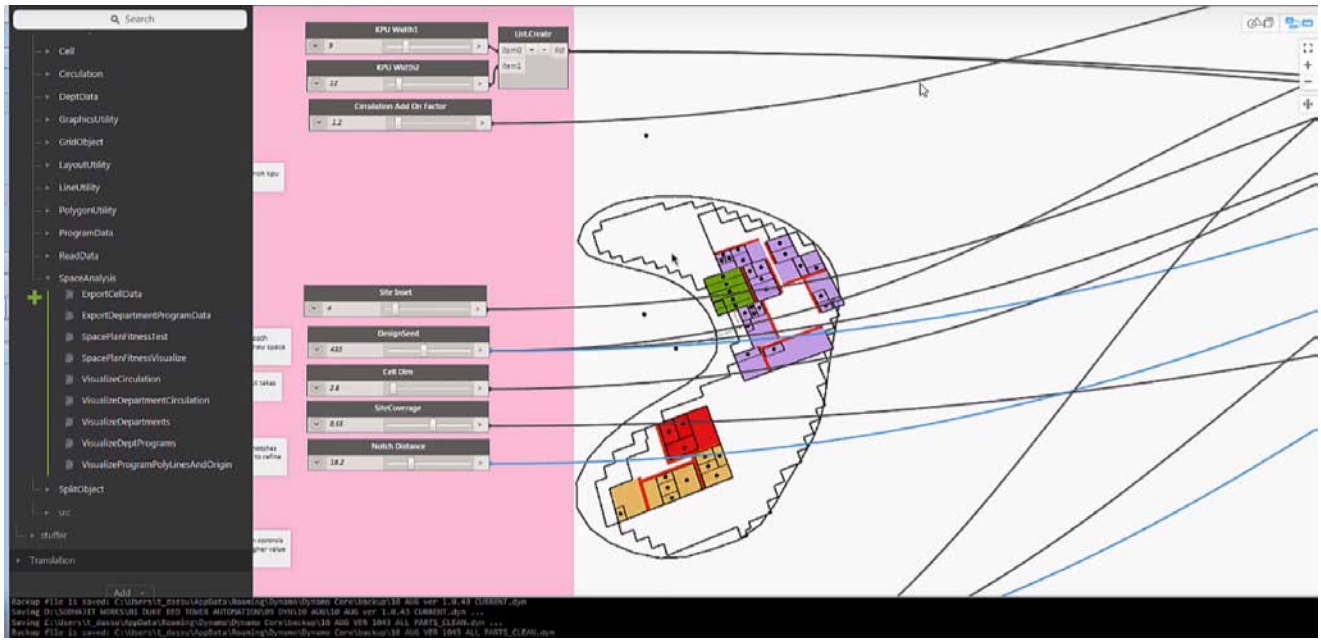


Figure 30 Possible arrangement of laydown area with 55% site coverage, 50m travel distance and highly variable material supply-demand curve

Figure 31 Possible arrangement of laydown area with 70% site coverage, 50m travel distance and highly variable material supply-demand curve

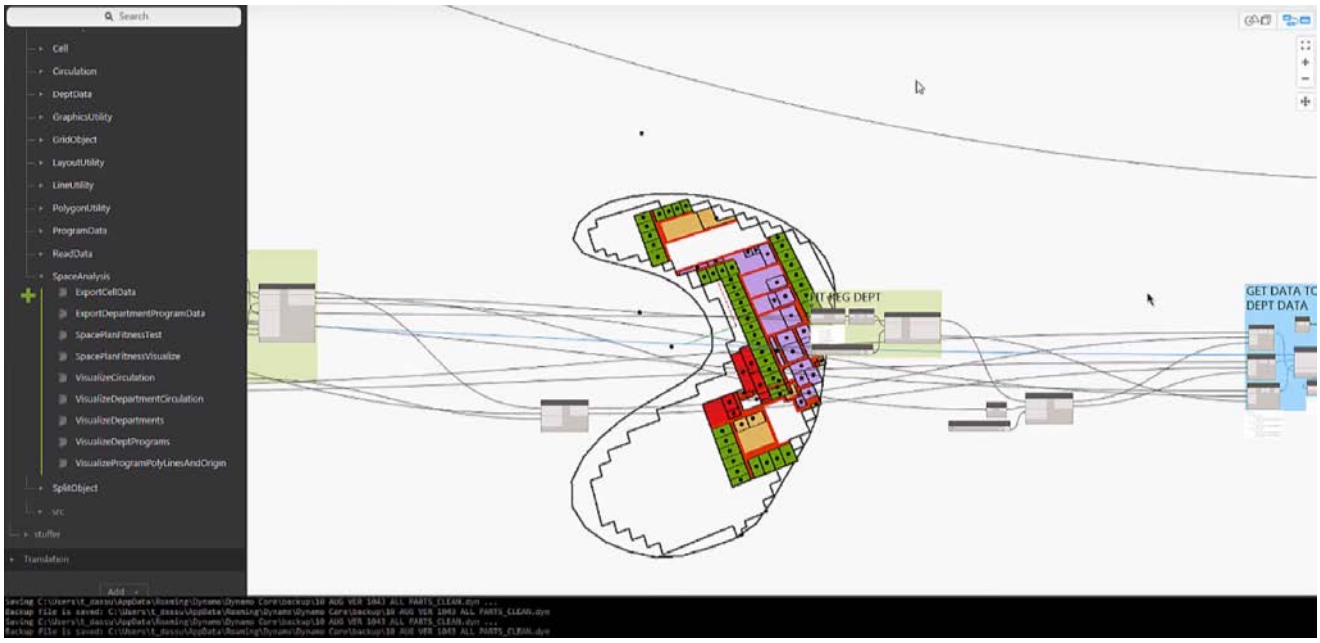


Figure 32 Possible arrangement of laydown area with 65% site coverage, 50m travel distance and highly variable material supply-demand curve

CONCLUSION

Though this is not the first attempt to use generative design strategies to develop space plans, this research leverages efficient data structures, coupled with robust, scalable algorithms from computational geometry and generative design, to deliver rational space plans for laydown areas. Building the system on Autodesk Dynamo allows many users to benefit from the system. Separating geometry and computation significantly improved system performance, allowing iterative solution searches to arrive at satisfactory results. Nevertheless, we recognize current system limitations such as an inability to handle non-orthogonal or curved spaces, as well as an inability to distribute spaces on multiple levels, but we plan to address these shortcomings. Currently, the system generates design options without learning from its previous iterations, sometimes leading to architecturally inadequate proposals. We envision surpassing this

limitation when the Configurator is coupled with genetic algorithm optimization in future work.

At present, the system can generate, score, and analyze design options each time the user adjusts a slider within the graph. Currently, each space plan is scored with respect to the percentage of program elements placed in comparison to program document requirements, the number of key planning units with access to circulation space, travel distance to nearest hoisting station and traffic handling capacity based on a normalized material supply-demand curve for a period of 15 days ensuring minimal change in subsequent layouts. Metrics are user-weighted, allowing the PM, CM or superintendent to change metrics to cater to any exigency.

We plan create a semantic tool linked with the project schedule which learns from generated options by Machine Learning and re-deploy options via a deep neural network to suggest best laydown area options. This will also require a IFC file transfer capability in a CDE which is currently lacking on most widely adopted platforms like Autodesk Navisworks (.nwd, .nwc), Synchro PRO and Synchro Scheduler (.sp) or Bentley ConstructSim(.xm) etc.

REFERENCES

Abourizk, S., Halpin, D., Mohamed, Y., and Hermann, U. (2011). “Research in modeling and simulation for improving construction engineering operation.” *Journal of Construction Engineering and Management*, 137(10), 843-852.

Akinci, B., Fischer, M., and Zabelle., T. (1998). “A proactive approach for reducing non-value adding activities due to time-space conflicts.” Proc. Sixth Annual Conference of the International Group for Lean Construction (IGLC-6), Lean Construction Institute, Arlington, VA, 1-18.

Azadivar, F., and Wang, J. (2000). “Facility layout optimization using simulation and genetic algorithms.” *International Journal of Production Research*, 4369- 4383.

Becker, S., M. Peter, D. Fritsch, D. Phillip, P. Baier, and C. Dibak. 2013. “Combined Grammar for the Modeling of Building Interiors.” ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences II–4 W1: 1–6.

Bentley, Jon Louis, and Jerome H. Friedman. 1979. “Data Structures for Range Searching.” ACM Computing Surveys 11 (4): 397–409.

Bentley, Jon Louis. 1975. “Multidimensional Binary Search Trees Used for Associative Searching.” Communications of the ACM 18 (9): 509–517.

1990. “K-d trees for semidynamic point sets.” In Proceedings of the Sixth Annual Symposium on Computational Geometry. Berkeley, CA: SOCG. 187–197.

Boon, Christopher, Corey Griffin, Nicholas Papaefthimious, Jonah Ross, and Kip Storey. 2015. “Optimizing Spatial Adjacencies using Evolutionary Parametric Tools: Using Grasshopper and Galapagos to Analyze, Visualize, and Improve Complex Architectural Programming.” Perkins + Will Research Journal 7 (2): 25–

37.

Das, Subhajit, John Haymaker, and Chuck Eastman. 2015. “Data Model and Processes for Building Programming.” Atlanta: Georgia Tech Digital Building Lab. http://www.dbl.gatech.edu/node/15402

Eastman, Charles. 1970. “Automated Space Planning.” Communications of the ACM 13 (4): 242–250.

Elbeltagi, E., and Hegazy, T. (2001). “A hybrid AI-based system for site layout planning in construction.” *Computer-Aided Civil and Infrastructure Engineering,* 16, 79-93.

El-Rayes, K., and Khalafallah, A. (2005). “Trade-off between safety and cost in planning construction site layouts.” *Journal of Construction Engineering and Management,* 131(11), 1186-1195.

El-Rayes, K., and Said, H. (2009). “Dynamic site layout planning using approximate dynamic programming.” *Journal of Construction Engineering and Management,* 23(2), 119-127.

Helsberg, Anders, Mads Torgersen, Scott Wiltamuth, and Peter Golde. 2010. C# Programming Language. Boston, MA: Addison-Wesley Professional.

Hicks, Chris, Tom McGovern, Gary Prior, and Iain Smith. 2015. “Applying Lean Principles to the Design of Healthcare Facilities.” International Journal of Production Economics 170 (Part B): 677–686.

Jo, Jun H., and John S. Gero. 1998. “Space Layout Planning Using an Evolutionary Approach.” Artificial Intelligence in Engineering 12 (3): 149–162.

Knecht, Katja, and Reinhard König. 2010. “Generating Floor Plan Layouts with K-d Trees and Evolutionary Algorithms.” Proceedings of the 13th International

Conference on Generative Art. Milan: GA. 238–253.

Liggett, Robert S. 2000. “Automated Facilities Layout: Past, Present and Future.” Automation in Construction 9 (2): 197–215.

Lopes, Ricardo, Tim Tutenel, Ruben M. Smelik, Klaas Jan de Kraker, and Rafael Bidarra. 2010. “A Constrained Growth Method for Procedural Floor Plan Generation.” In Proceedings of GAMEON. Leicester, UK: EUROSIS. 13–22.

Marson, Fernando, and Soraia Raupp Musse. 2010. “Automatic Real- Time Generation of Floor Plans Based on Squarified Treemaps Algorithm.” International Journal of Computer Games Technology 2010 (7): 10.

Michaleka, Jeremy, Ruchi Choudhury, and Panos Papalambrosa. 2002. “Architectural Layout Design Optimization.” Engineering Optimization 34 (5): 461–484.

Mirahmadi, Maysam, and Abdallah Shami. 2012. “A Novel Algorithm for Real-time Procedural Generation of Building Floor Plans.” arXiv:1211.5842v1

Nassar, Khaled. 2010. “New Advances in the Automated Architectural Space Plan Layout Problem.” In Proceedings of the International Conference In Computing in Civil and Building Engineering, edited by Walid Tizani. Nottingham, UK: ICCBE.

Rechel, Bernd, Buchen James, and Mckee Martin. 2009. “The Impact of Health Facilities on Healthcare Workers’ Well-Being and Performance.” International Journal of Nursing Studies 46 (7): 1025–1034.

Shneiderman, Ben. 1992. “Tree visualization with tree maps: 2-d space- filling approach.” ACM Transactions on Graphics (TOG) 11 (1): 92–99.

Wang, Shu-Xi. 2012. “The Improved Dijkstra’s Shortest Path Algorithm and Its Application.” Procedia Engineering 29: 1186–1190.

Zhang, H., and Wang, J. Y. (2008). “Particle swarm optimization for construction site unequal-area layout.” *Journal of Construction Engineering and Management,* 134(9), 739-748.

Zhou, F., AbouRizk, S. M., and AL-Battaineh, H. (2009). “Optimisation of construction site layout using a hybrid simulation-based system.” *Simulation Modelling Practice and Theory,* 17, 348-363.

Zouein, P. P., and Tommelein, I. D. (1994). “Automating dynamic layout construction.” Proc. International Symposium on Automation and Robotics in Construction, IAARC, 409- 416.

Zouein, P. P., and Tommelein, I. D. (1999). “Dynamic layout planning using a hybrid incremental solution method.” *Journal of Construction Engineering and Management,* 125(6), 400-408.

Zouein, P.P., and Tommelein, I. D. (2001). “Improvement algorithm for limited space scheduling.” *Journal of Construction Engineering and Management*, 127(2), 116–124.