Space Plan Generator

Rapid Generation & Evaluation of Floor Plan Design Options to Inform Decision Making

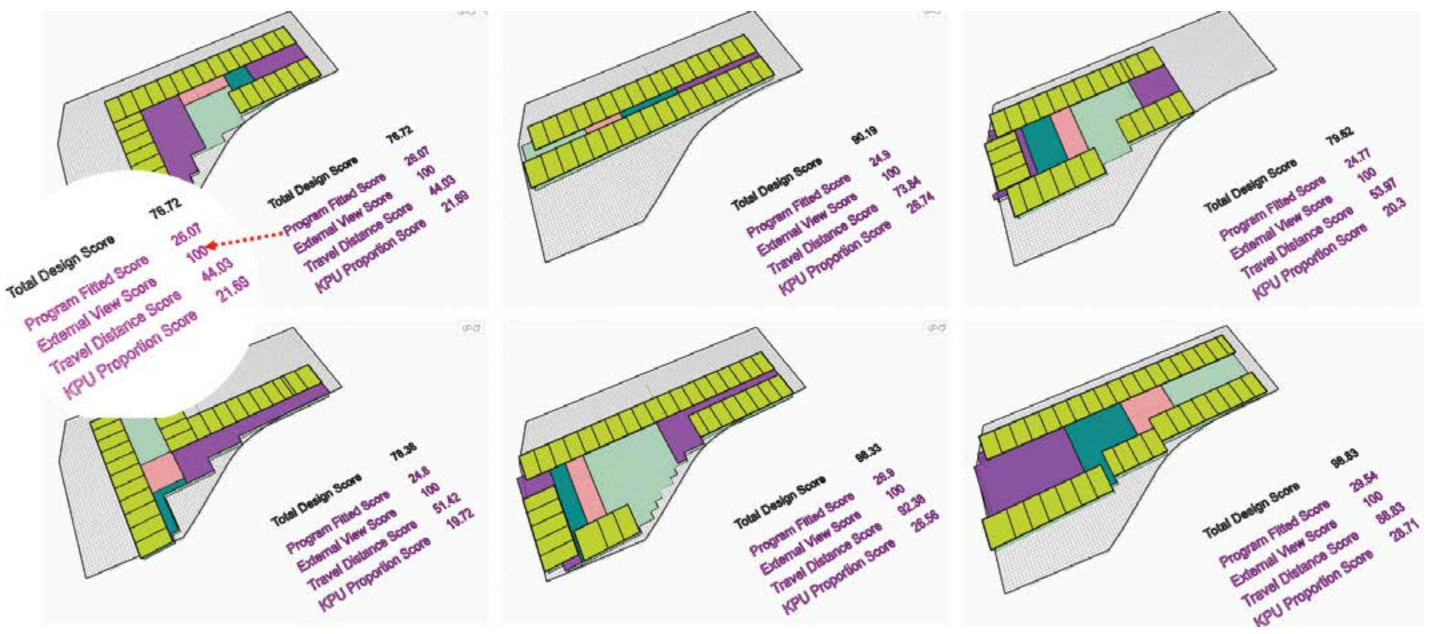
Subhajit Das
Autodesk / Georgia Tech

Colin Day

Anthony Hauck
Autodesk

John Haymaker

Diana Davis
Perkins+Will


Total Design Score 76.72
Program Fitted Score 26.07
External View Score 100
Travel Distance Score 44.03
KPU Proportion Score 21.69
Total Design Score 90.19
Program Fitted Score 24.9
External View Score 100
Travel Distance Score 73.84
KPU Proportion Score 26.74
Total Design Score 79.62
Program Fitted Score 24.77
External View Score 100
Travel Distance Score 53.97
KPU Proportion Score 20.3
Total Design Score 78.36
Program Fitted Score 24.8
External View Score 100
Travel Distance Score 51.42
KPU Proportion Score 19.72
Total Design Score 96.33
Program Fitted Score 28.9
External View Score 100
Travel Distance Score 92.38
KPU Proportion Score 26.56
Total Design Score 98.83
Program Fitted Score 29.54
External View Score 100
Travel Distance Score 98.83
KPU Proportion Score 28.71

1

1 Generated space plan layout options with design score for reference.

ABSTRACT

Design exploration in architectural space planning is often constrained by tight deadlines and a need to apply necessary expertise at the right time. We hypothesize that a system that can computationally generate vast numbers of design options, respect project constraints, and analyze for client goals, can assist the design team and client to make better decisions. This paper explains a research venture built from insights into space planning from senior planners, architects, and experts in the field, 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 described below automatically generates hundreds of design options from inputs typically provided by an architect, including a site outline and program document with desired spaces, areas, quantities, and adjacencies to be satisfied. We envision that this workflow can clarify project goals early in the design process, save time, enable better resource allocation, and assist key stakeholders to make informed decisions and deliver better designs. Further, the system is tested on a case study healthcare design project with set goals and objectives.

INTRODUCTION

Design teams work under schedule and resource constraints that limit the range of design solutions they can generate and analyze, impeding informed design and optimal decision making. We have surveyed building guidelines, codes, facility-planning rules, literature, and case studies, and conducted interviews with senior professional medical planners and architects to understand industry best practices and acceptable design methodologies (Das, Haymaker, and Eastman 2015). This work clarified the implicit and explicit domain knowledge and processes in space planning and identified opportunities to leverage computation for repetitive design generation and analysis tasks, liberating architects to invest time in problem formulation and decision making.

In this paper, we present Space Plan Generator (SPG), an emerging methodology and tool to automate aspects of architectural design. We first briefly describe foundational work in data structures and space-planning methodologies, then explain the working methodology of SPG (see Figure 2) 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 testing of its the validity and efficacy within a healthcare facility case study from a large architectural practice.

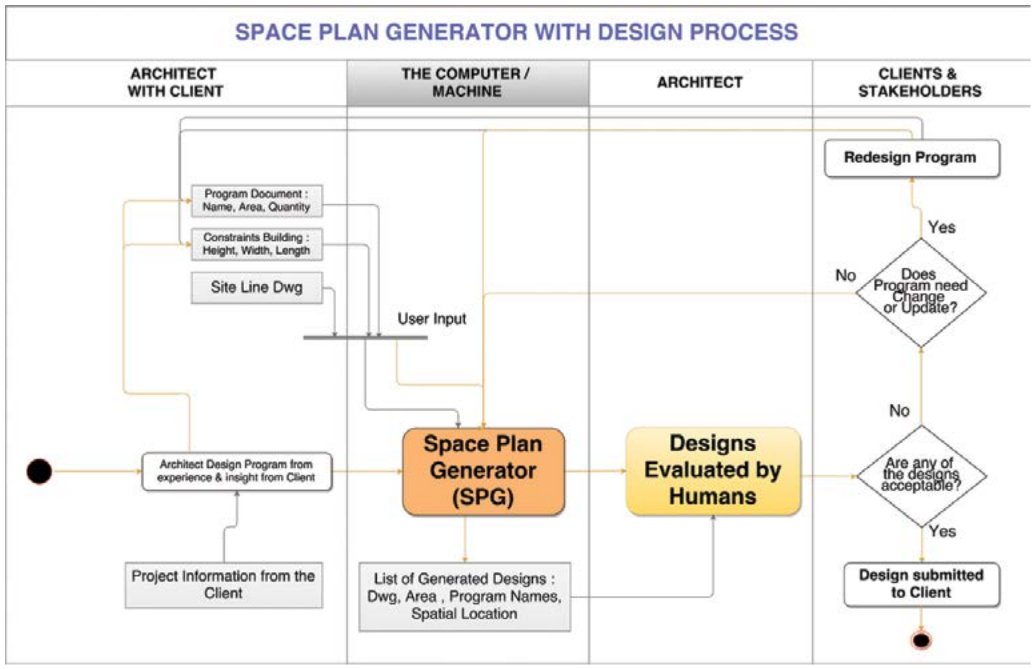
RELATED WORKS

TreeMap Data Structure

Architectural design is a process necessitating 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).


SPACE PLAN GENERATOR WITH DESIGN PROCESS
ARCHITECT WITH CLIENT
THE COMPUTER / MACHINE
ARCHITECT
CLIENTS & STAKEHOLDERS
Redesign Program
Program Document : Name, Area, Quantity
Constraints Building : Height, Width, Length
Site Line Dwg
User Input
Yes
No
Does Program need Change or Update?
No
Architect Design Program from experience & insight from Client
Space Plan Generator (SPG)
Designs Evaluated by Humans
Are any of the designs acceptable?
Yes
Project Information from the Client
List of Generated Designs : Dwg, Area , Program Names, Spatial Location
Design submitted to Client

2 Activity diagram showing SPG and architect together in a design process.

Space Plan Generation

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

The Space Plan Generator 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.

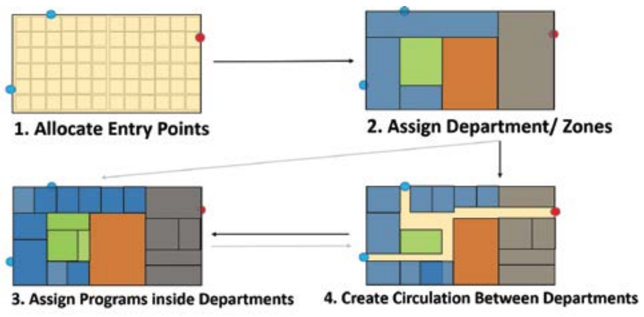
Hierarchical Space Assignment

Our approach is hierarchical (see Figure 3). 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.

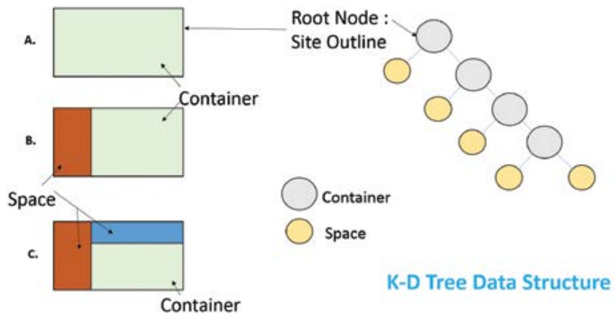
K-d Tree Data Structure

Our space-assignment algorithm uses a K-d data structure dividing the K-d space by partition planes perpendicular to one of the coordinate axes (see Figure 4). 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. Each generated node is assigned a space or region, each of which can contain child nodes (Bentley 1990). K-d Trees efficiently search and traverse data sets to create spatial plan partitions (Bentley 1975).

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 particular 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


1. Allocate Entry Points
2. Assign Department/ Zones
3. Assign Programs inside Departments
4. Create Circulation Between Departments

3 Hierarchical space planning approach.



4 Space Data Tree construction based on K-d data structure binary tree.

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 slice and dice techniques (Shneiderman 1992), 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 (see Figure 4).

Benefits of using Space Data Tree:

- Provides fast and efficient data storage to represent spatial data.
- Supports nearest neighbor searches, which help build department topology maps and appraise the efficiency of a layout by representing department neighbors and adjacency.
- Finds neighbors for departments and program elements and shared edges between them, with each Space and Container Node storing the splitting line. Thus allows the computation of circulation networks between and within departments.
- 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.

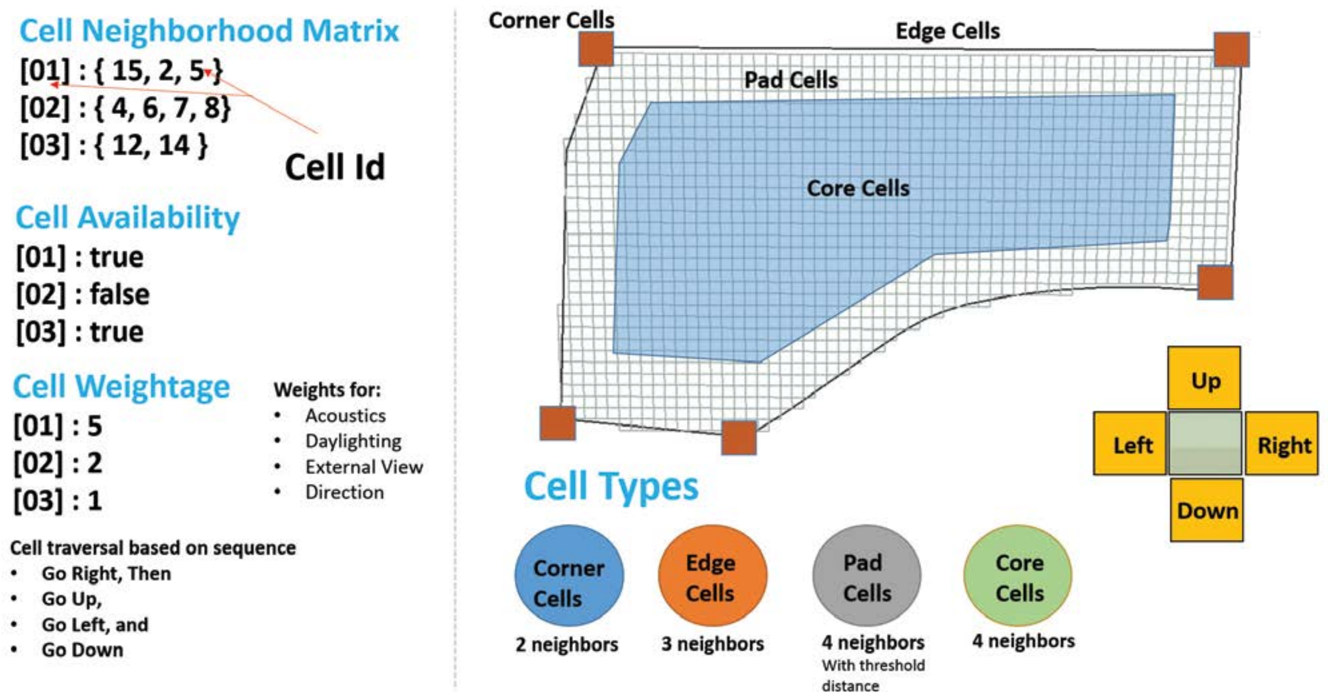
Splitting Strategies

We 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 acceptable 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 (see Figure 5b). The algorithm



5 (a) Shows Cell storage based on ID, Cell neighbor matrix, Cell weights to account for specific objectives. (b) Shows types of cells based on the number of neighbors they have.

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.

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 (see Figure 5a).

Cell Weighting for Specific Metrics

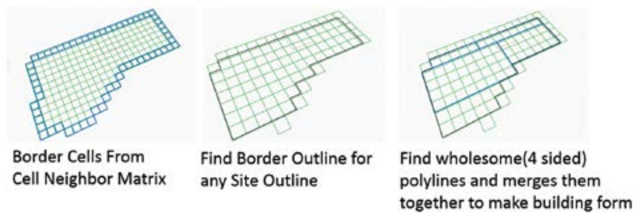
Each cell is given a custom weight to gauge different aspects of the project, such as acoustic performance, daylighting, and site constraints. Weighting influences the allocation of spatial locations for a certain program. For example, if the northwest side of the site is conducive for daylighting, then 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 (see Figure 6). 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 traversesg 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.


Border Cells From Cell Neighbor Matrix
Find Border Outline for any Site Outline
Find wholesome(4 sided) polylines and merges them together to make building form

6 Polygon Boundary algorithm showing the orthogonal boundary for any input arbitrary site outline.

Analyzer

In addition to the design generator, an analyzer component validates the efficacy of the generated space plan with respect to the original project 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. Please see Figure 7 to understand the working methodology of SPG 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 generator as explained below (see Figure 8). The Dynamo graph is composed of two components, a 'Generator' and an 'Analyzer.' After every graph evaluation, the 'Generator' yields distinct space plan layouts rendered as a list of polyline geometries. The 'Analyzer' gauges the fitness of each design option to input goals and constraints. The prototype proceeds as follows:

Step 1: User Inputs

The graph needs requirements in the form of a site outline '.sat' file and program document '.csv' file (see Figure 9). 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, quantity, area or dimension, program, preference value, and adjacency list.

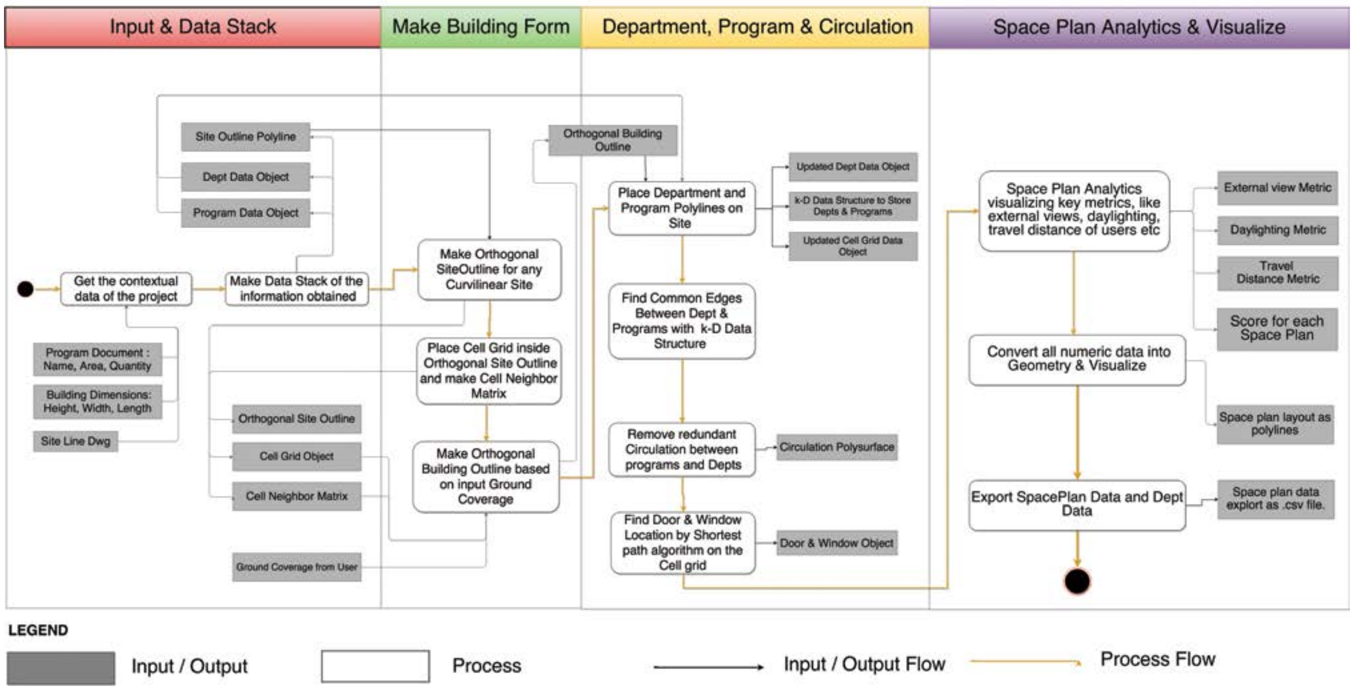
Preference value for each program element specifies the order in which the element will be placed onto the site. The adjacency list is a comparison chart to which the program and department topology map will adhere while allocating program elements on the site. Other user inputs include corridor width, the aspect ratio for circulation and program elements, grid cell dimension, etc.

Step 2: Cell Grid Information

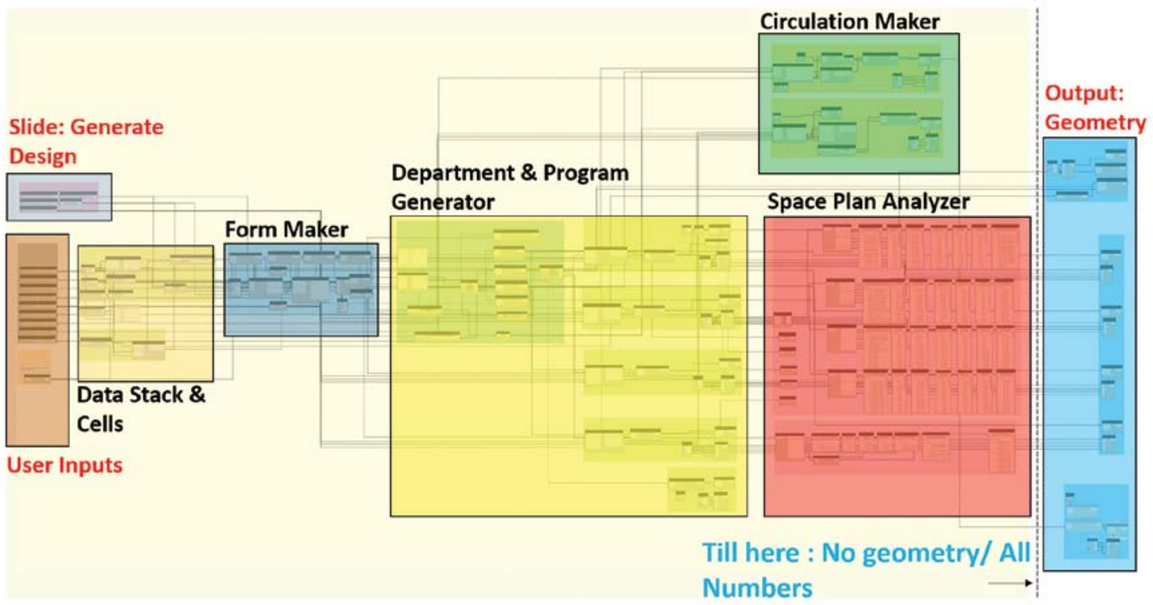
The graph builds the cell grid on the site and the initial cell neighbor matrix, which are used repeatedly in the workflow explained above.

Step 3: Data Stack

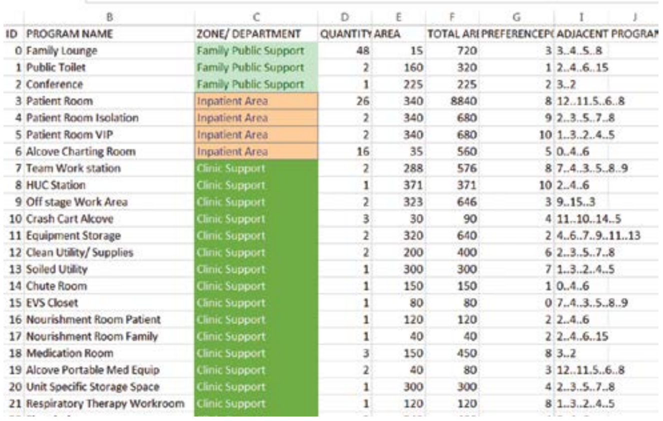
This step extracts input information from the program document and builds a Department Data and Program Data class object. The data stack node sorts and stores the department and program data object based on the preference values from the


Input & Data Stack
Make Building Form
Department, Program & Circulation
Space Plan Analytics & Visualize
Site Outline Polyline
Dept Data Object
Program Data Object
Get the contextual data of the project
Make Data Stack of the information obtained
Program Document : Name, Area, Quantity
Building Dimensions: Height, Width, Length
Site Line Dwg
Orthogonal Site Outline
Cell Grid Object
Cell Neighbor Matrix
Ground Coverage from User
Make Orthogonal SiteOutline for any Curvilinear Site
Place Cell Grid inside Orthogonal Site Outline and make Cell Neighbor Matrix
Make Orthogonal Building Outline based on input Ground Coverage
Orthogonal Building Outline
Place Department and Program Polylines on Site
Updated Dept Data Object
k-D Data Structure to Store Depts & Programs
Updated Cell Grid Data Object
Find Common Edges Between Dept & Programs with k-D Data Structure
Remove redundant Circulation between programs and Depts
Circulation Polysurface
Find Door & Window Location by Shortest path algorithm on the Cell grid
Door & Window Object
Space Plan Analytics visualizing key metrics, like external views, daylighting, travel distance of users etc
External view Metric
Daylighting Metric
Travel Distance Metric
Score for each Space Plan
Convert all numeric data into Geometry & Visualize
Space plan layout as polylines
Export SpacePlan Data and Dept Data
Space plan data explort as .csv file.
LEGEND
Input / Output
Process
Input / Output Flow
Process Flow

7 Activity diagram showing Space Plan Generator's working methodology.

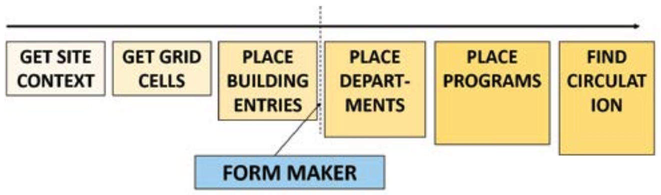


8



	B	C	D	E	F	G	I	J
ID	PROGRAM NAME	ZONE/ DEPARTMENT	QUANTITY	AREA	TOTAL ARI	PREFERENCEP(	ADJACENT PROGRAM	
0	Family Lounge	Family Public Support	48	15	720	3	3..4..5..8	
1	Public Toilet	Family Public Support	2	160	320	1	2..4..6..15	
2	Conference	Family Public Support	1	225	225	2	3..2	
3	Patient Room	Inpatient Area	26	340	8840	8	12..11.5..6..8	
4	Patient Room Isolation	Inpatient Area	2	340	680	9	2..3..5..7..8	
5	Patient Room VIP	Inpatient Area	2	340	680	10	1..3..2..4..5	
6	Alcove Charting Room	Inpatient Area	16	35	560	5	0..4..6	
7	Team Work station	Clinic Support	2	288	576	8	7..4..3..5..8..9	
8	HUC Station	Clinic Support	1	371	371	10	2..4..6	
9	Off stage Work Area	Clinic Support	2	323	646	3	9..15..3	
10	Crash Cart Alcove	Clinic Support	3	30	90	4	11..10..14..5	
11	Equipment Storage	Clinic Support	2	320	640	2	4..6..7..9..11..13	
12	Clean Utility/ Supplies	Clinic Support	2	200	400	6	2..3..5..7..8	
13	Soiled Utility	Clinic Support	1	300	300	7	1..3..2..4..5	
14	Chute Room	Clinic Support	1	150	150	1	0..4..6	
15	EVS Closet	Clinic Support	1	80	80	0	7..4..3..5..8..9	
16	Nourishment Room Patient	Clinic Support	1	120	120	2	2..4..6	
17	Nourishment Room Family	Clinic Support	1	40	40	2	2..4..6..15	
18	Medication Room	Clinic Support	3	150	450	8	3..2	
19	Alcove Portable Med Equip	Clinic Support	2	40	80	3	12..11.5..6..8	
20	Unit Specific Storage Space	Clinic Support	1	300	300	4	2..3..5..7..8	
21	Respiratory Therapy Workroom	Clinic Support	1	120	120	8	1..3..2..4..5	

9



10

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 building 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 building 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 building form. One advantage of this approach is that the user can set a percentage of total site area for the building to occupy, resulting in building perimeters of specific site coverage. The merge operation employs the Polyline Boundary algorithm as explained above.

Strategy 2: Form Follows Function

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

8 Current State of the Autodesk Dynamo Graph. Every node is a custom node, written in C#.

9 Requirements supplied to the graph via .csv program document.

10 'Form Maker' in the sequence of hierarchical space planning approach to generate architecturally rationale space plans.

11 Form maker in action placing colored poly surfaces as individual departments.

12 (a) Shows the Dept. Analytics node highlighting area, cells, and programs for all four departments. (b) Shows the area percentage desired and achieved in relation to other departments for all four departments.

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 building outline using minimum edge distance, thus refining the building form and increasing the number of design choices for the user.

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.

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.

Step 7: Circulation Computation
Circulation computation discovers circulation networks between departments 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 available 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).

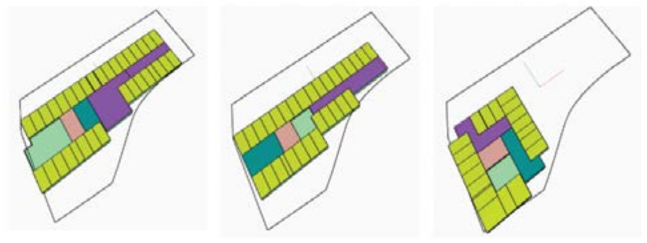
Step 8: Space Plan Analytics
Evaluates the generated design based on metrics as summarized above (see Figure 12).

Step 9: Space Plan Output / Geometry Rendering
One of the salient features of the SPG is that it maintains a distinction between geometry and computation, with geometry only rendered at the process conclusion for visualization (see Figures 13a and 13b). 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.

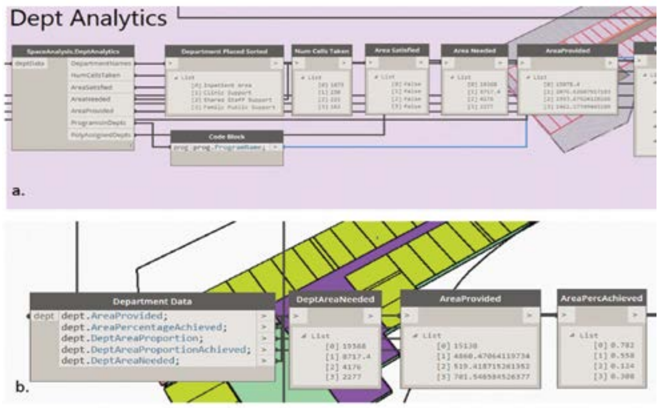
Step 10: Storing and Scoring Generated Designs
Generated designs are stored as a collection of cells, where each cell stores information about its assigned program or department. Cells store space plan analytic information, such as distance to external windows, visibility of the cell from circulation 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 (see Figure 1).

USE CASE AND RESULTS

The prototype is being tested on a new hospital bed tower to be built on an existing healthcare facility site which includes an existing bed tower and hospital facility (see Figure 14). With site constraints and specific client goals, such as maximizing patient beds per floor, employing new hospital facility design guidelines and building codes, minimizing nurse travel distance (Rechel, James, and Martin 2009), maximizing connectivity to the existing hospital, and minimizing view impedance from the existing bed tower, this project is a valuable test case to understand how a goal-driven design workflow can be automated using generative design strategies. The current state of the Dynamo package



11



```
prog  prog.ProgramName;
```

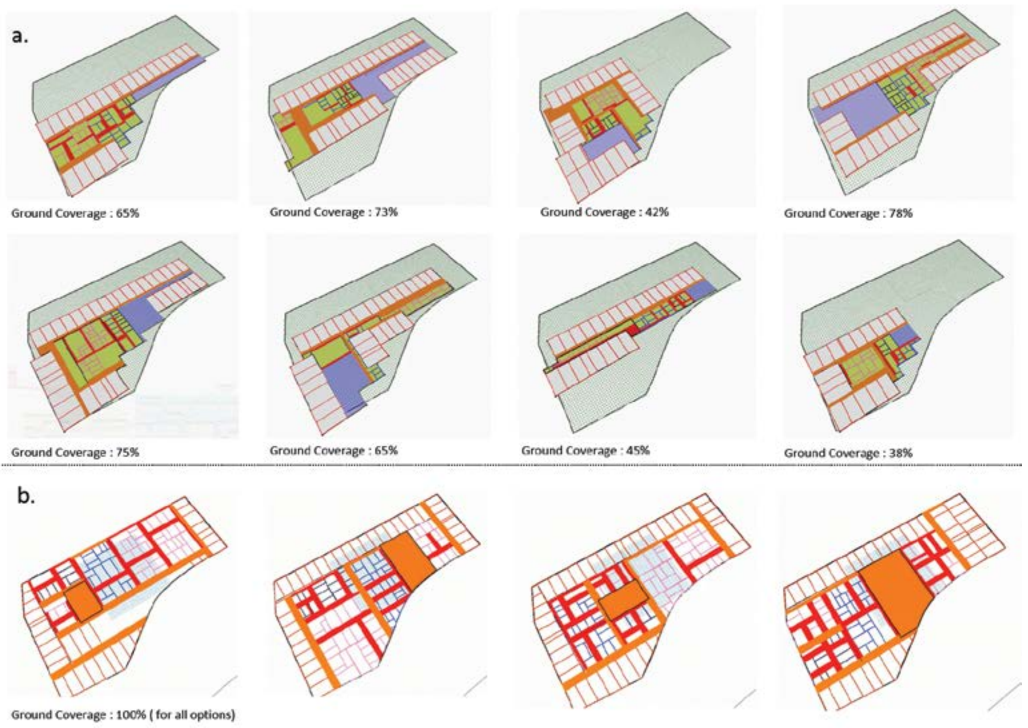
```
dept  dept.AreaProvided;
      dept.AreaPercentageAchieved;
      dept.DeptAreaProportion;
      dept.DeptAreaProportionAchieved;
      dept.DeptAreaNeeded;
```

```
List
  [0] 19368
  [1] 8717.4
  [2] 4176
  [3] 2277
```

```
List
  [0] 15138
  [1] 4860.47064119734
  [2] 519.418715261352
  [3] 701.546584526377
```

```
List
  [0] 0.782
  [1] 0.558
  [2] 0.124
  [3] 0.308
```

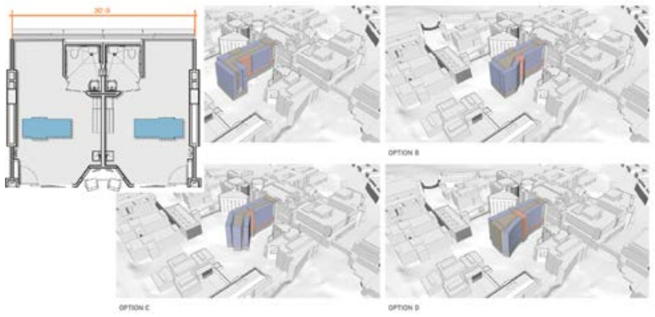
12



13 (a) Generated Space Plan Layouts after using 'Form Maker' based on set ground coverage. (b) Space Plan Layouts without using 'Form Maker' filling the whole site space with program elements.

restricts designs to a single floor as a limited case to ensure stability and reliability before being implemented for multiple floors.

At present, the system is capable of generating, scoring, and analyzing 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 patient rooms with access to external views and daylighting, nurse travel distance to all patient rooms, and percentage number of Key Planning Units (inpatient patient beds in this case). Metrics are user-weighted, allowing architects to seek design solutions for project goals. We plan to couple the generator with Genetic Algorithm (GA) optimization to improve design candidates by learning from each iteration to reach optimal floor layouts, as driven by project goals.



14 Concept forms generated manually by the design team. Top left shows modular patient room for the case study Hospital Bed Tower.

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 architectural space plans. Building the system on Autodesk Dynamo allows a large number of 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 generator is coupled with genetic algorithm optimization in future work.

REFERENCES

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.

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.

Subhajit Das has a diverse background in Architecture and Computer Science, with degrees in M Arch Design Computing (University of Pennsylvania) and MS in Computer Science (Georgia Tech). Being part of the Autodesk Generative Design group, Subhajit was instrumental to collaborating with Perkins+Will Healthcare Design team to steer this research forward. Currently, he is pursuing a PhD in Computer Science at Georgia Tech with core interests in Machine Learning, Graphics, and Visual Analytics.

Colin Day is a Principal Engineer in the Autodesk Generative Design Group, with years of experience in the development and implementation of algorithms in Computational Design and Computer Graphics.

Anthony Hauck joined the Autodesk Revit team in 2007, holding a succession of product management positions in the group until joining Autodesk AEC Generative Design in 2015 as its Director of Product Strategy, where he is responsible for helping define the next generation of building software products and services for the AEC industry.

John Haymaker, PhD, AIA, LEED AP serves as Perkins+Will’s Director of Research, overseeing areas of inquiry including materials, design process, building technology, and healthcare practices. Previously a Professor of Civil Engineering at Stanford University, and at Georgia Tech, John has contributed over 80 articles to professional literature in the areas of design process communication, optimization, and decision-making.

Diana Davis has over 18 years of experience in the design, planning, and delivery of healthcare projects. She brings a special interest in Lean planning and evidence-based design to her work, paired with a commitment to improving the healthcare environment for caregivers, patients, and families. She is mentoring graduate level researchers at the Georgia Institute of Technology and Autodesk in the development of new digital tools to aid healthcare planning and design.