PERKINS+WILL

Research Journal

2015 / VOL 07.02

www.perkinswill.com

SPECIAL ISSUE: FUTURE OF ARCHITECTURAL RESEARCH

ARCHITECTURAL RESEARCH CENTERS CONSORTIUM 2015 CONFERENCE

02. OPTIMIZING SPATIAL ADJACENCIES USING EVOLUTIONARY PARAMETRIC TOOLS: *Using Grasshopper and Galapagos to Analyze, Visualize, and Improve Complex Architectural Programming*

Christopher Boon, Portland State University, *christopher.a.boon@gmail.com*
Corey Griffin, Assoc. AIA, Portland State University, *cgriffin@pdx.edu*
Nicholas Papaefthimious, AIA, LEED AP BD+C,
ZGF Architects LLP, *nicholas.papaefthimiou@zgf.com*
Jonah Ross, ZGF Architects LLP, *jonah.ross@zgf.com*
Kip Storey, ZGF Architects LLP, *kip.storey@zgf.com*

ABSTRACT
This article documents the use of Grasshopper and Galapagos (Rhino plugins) as analytical tools to graphically represent and optimize the adjacency requirements in programmatic spaces. The resulting three-dimensional spatial diagrams are evaluated based on evolutionary fitness, which within this research context is defined as minimizing the numerical value of the total distance of all interconnected programmatic elements. This approach offers a unique system of analysis that can create an extensive range of otherwise unexplored solutions to space planning when faced with complex adjacency requirements and a large number of programmatic elements. The Grasshopper script is a significant leap forward over existing research in this area, with the ability for users to define an irregular shaped site boundary, input multiple stories, relate program elements to external adjacencies (views, parking, etc.) and handle an unlimited number of program elements. Consequently, the resulting script can be used as an aid in the schematic design of buildings that have inherently complex programmatic relationships. This article uses a three-story hospital on a sloping site with fifty program elements to demonstrate the efficacy of this approach.

KEYWORDS: space layout planning, parametric design, genetic algorithms

1.0 INTRODUCTION

Parametric tools for architectural design have advanced significantly since the advent of "blob architecture" at Columbia University and as seen in the work of Greg Lynn, Michael McInturf, and Douglas Garofalo in the mid-1990s[1]. The early computer software tools used by architects during this time including Maya and CATIA, allowed for the design and construction of complex forms not previously possible. Due to the perception that these new forms were little more than willful sculptures that performed poorly[2], the pejorative term "blobitecture" was coined[3].

Despite these inauspicious beginnings, academics and practicing architects realized that the implications for parametric design are larger than generating unusual forms. In the past two decades, parametric tools have been used to enhance the design process, allowing architects to iterate more quickly and focus on optimizing the performance of a building. These optimizations have focused on topics as varied as reducing energy use[4], improving the flow of passengers in an airport[5], and the topic of this article – space layout planning[6]. Before discussing the use of evolutionary design tools in space planning in more depth, we will first outline the development of parametric design and genetic algorithms.

1.1 Parametric Design

Scholarly articles on parametric design first appear in the 1950s and 60s, focusing on the design of rockets and airplanes[7,8], and therefore it should be no surprise that the early proponents of parametric design in architecture used software originally developed for airplane design. In one of the first articles to discuss the impact of parametric modeling on architecture, Roller (1991) discusses the way specific information storage - parametric information - in combination with the construction process has the potential to capture a true version of design intent, while the benefits of iterative variation in the design process are examined[9]. Motta (1999) focuses on computer-aided, intelligent modeling systems, illustrating their widespread application and value as tools that have the potential for reuse. A series of precedents in parametric design highlight the importance of establishing a workflow that produces useful results in this field[10]. More recently, Woodbury (2010) summarizes the role of parametric modeling in architectural design, "rather than the designer creating the design solution (by direct manipulation) as in conventional design tool, the idea is that the designer first establishes the relationships by which parts connect, builds up a design using these relationships and modifies the relationships by observing and selecting from the results produced"[11]. He breaks down the essential components of parametric modeling, including geometry, programming, and patterns.

During conceptual and schematic design stages, much of what is explored involves theoretical concepts. The work is expressed diagrammatically. If the concept can be distilled into numerical parameters, then it is possible to use parametric modeling. This type of analysis allows designers to explore a much wider range of options in a much shorter timeframe. This article explores the ability of evolutionary parametric modeling as a space planning diagram-building tool. For a building to be efficient, it must meet the needs of its occupants. The needs of the occupants are determined during the design process by a complex relationship of programmatic and physical adjacencies. These spaces and their adjacencies can be measured and converted as numeric values, which in turn allow them to be turned into parameters to be manipulated.

1.2 Genetic Algorithms and Evolutionary Design

The field of biology, and more specifically evolutionary developmental biology, has studied the relationship between evolution and form for living organisms for decades. This work is best summarized by Thompson (1942) and more recently by Carroll (2005)[12,13] and highlights the breadth and depth of research in evolutionary "design" in biology over architecture. Fogel (1966) and Holland (1975) were two of the first authors to apply evolutionary design to artificial systems, introducing the concept of genetic algorithms to model evolution[14,15]. Goldberg (1989) popularized the use of genetic algorithms as general function optimizers for other sciences and engineering[16]. Lenski et al. (1999) explore the ability of digital models to evolve like organisms in nature[17]. Through experimentation, they attempt to find the best possible way to create a digital genome as well as how influences in nature, such as mutation, can affect the population. The authors use diagrams to help explain the methods that can achieve the fittest results.

Frazer (1995) provides an overview of early work applying genetic algorithms to architecture, adopting nature's processes of evolution to the design of buildings[18]. Applying genetic algorithms for generative design and structural form finding, Hensel et al. (2010) and Weinstock (2010) focus on the concept of emergence as the main process for evolutionary design, "emergence is… a central concept of biomimetics, in which biological structures are analyzed and understood as material hierarchies self-organized into structures that are achieved by a bottom up process of self-assembly from which their properties and performances emerge"[19, 20]. Frazer, Hensel, and Weinstock all emphasize the role of biomimicry in their work over using genetic algorithms as optimization engines to solve complex, multi-objective problems as seen in computer science and engineering. Weng et al. (2014) summarize the history and use of genetic algorithms in optimizing the form of buildings with an emphasis on reducing energy use[4].

1.3 Previous Visual, Parametric Tools for Space Planning

Several academic researchers have looked at generating algorithms for computer optimized space planning over the last fifteen years[21-28]. Dutta and Sarthak (2011) provide a summary of this line of research[6]. This academic research focuses largely on the computer programming and mathematics behind the genetic algorithms and optimizations developed. However, more practical applications are needed, especially in visualizations for these space planning algorithms or "tools". Visual tools are necessary in order to communicate the results of space planning with a variety of stakeholders and allows for architects to quickly assess the merits of a given scheme.

Architecture firms have been developing space planning visualization and optimization tools using Grasshopper. LMN has used detailed sets of spreadsheets to create architectural geometry based on programmatic relationships[29,30]. The tool arranges program elements based on relationships marked within the information on the spreadsheets and translates this information into individual spaces that organize themselves according to user defined hierarchies. This parametric tool is limited in that it is primarily used for visualizing program spaces with no algorithm for optimizing adjacencies or even generating a spatial layout. NBBJ developed a tool that allows designers to quickly organize and visual architectural programs in three dimensions[31]. The project uses spring physics to create gravity between the various elements, which are represented by spherical volumes, to optimize adjacencies. This tool is very limited and requires significant input on the part of the user to transform the results of the spring physics into an architectural space layout plan. However, similar research using Newtonian gravitational models has been recently published by academic researchers at the Vienna University of Technology[28]. Unfortunately, neither LMN or NBBJ have written peer-reviewed articles about their work in this field. This is likely due to the desire to keep research confidential that gives the firms an edge when competing with other firms for projects or simply the lack of resources to pursue peer-reviewed research.

1.4 Goals for a New Optimization Tool

This article explores the utilization of Grasshopper (a Rhino plugin) as an analytical tool to graphically represent a three-dimensional analysis of adjacency requirements in space layout planning. As much space planning in architecture firms is based on past experience with similar building types and a limited number of iterations, this research explores new ways of using evolutionary design to improve floor-plan layouts for complex programs, such as hospitals, and provide potential solutions that the architect would not have discovered otherwise. Using an adjacency matrix and list of program spaces as the input, the Grasshopper script developed in this research uses the "evolutionary solver" Galapagos to minimize the distance between programmatic elements that need to be adjacent and creates three-dimensional diagrams. As a brute force iterative process, the genetic algorithm can work away at a space layout problem without further user input until a certain amount of time as elapsed or the fitness plateaus. This type of approach offers a unique system of analysis that can create an extensive range of unique and otherwise unexplored solutions when faced with problems in developing complex order for spatial relationships.

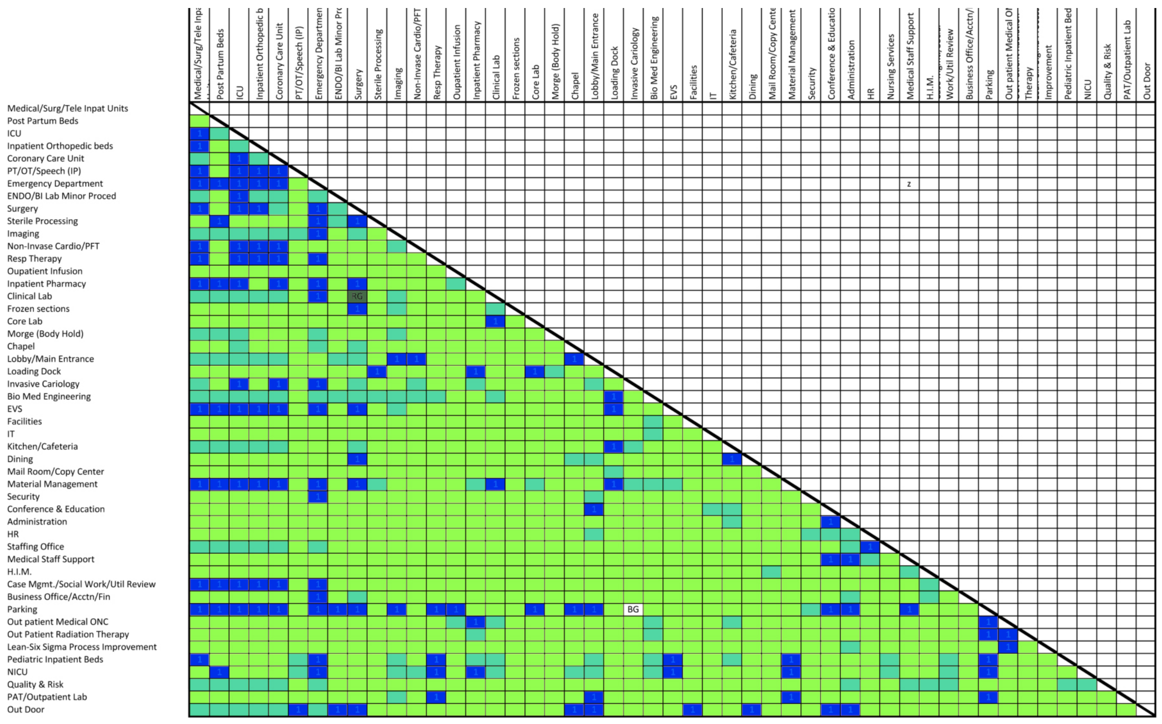
2.0 METHODOLOGY

2.1. Design Directive

Programming for complex buildings, such as a health care facility, is a difficult job. This is due to the high level of complexity in spatial relationships and high number of programmatic elements. A hospital also requires a high level of efficiency, due to the nature of its function in society.

ZGF Architects LLC (ZGF) has significant experience designing hospitals. With this expertise, one can rely on client interviews and intuition to achieve efficiency in spatial relationships – grouping elements that need to be adjacent and minimizing the travel distance between them. At the beginning of this research project, ZGF was in process of designing a hospital and there were program adjacency charts generated at client meetings and these resources were made available as sample test figures (Table 1). The numbers on the charts represent the various needs of the client. At meetings, the design team and clients develop a matrix of all the discrete programmatic elements. The matrix forms a hierarchy of space in terms of priority and ranks the adjacency requirement of each element in relation to all other elements in the chart. A space like the emergency room would take priority over the gift shop for example, and the two have no adjacency need.

Table 1: This matrix shows the various programmatic elements for a hospital and the relationship with others. There are three different levels of importance indicated by tones.



2.2. Tool Development and Geometry

The definition begins by input of some simple geometric parameters, thereby establishing the programmatic volumes that will be used in the iterative process. These first steps are the customizable portions of the tool. Each unique architectural situation requires some unique combination of factors that this research attempts to address to some extent in terms of spatial proxemics. The potential floor plan space is all that is actually drawn by the user. The ground floor of the building or site outline is first drawn and then additional floors can be added of any shape. Floor to ceiling height is adjustable. Any number of floors is allowed and there can be multiple sites separated from one another. Any shape can be used to describe the boundary conditions. The tool will automatically fill in the shapes with a three dimensional, rectilinear grid that can also be adjusted as desired.

The geometry all occurs within a three dimensional grid. This begins to describe a geometric system. All of the cells in the grid are marked with a single point. These points are all contained in a cloud. Each programmatic element is also built around a single point (Figure 1). The programmatic central points are here referred to as “origins”. Within the list of programs, there exists an explicit hierarchy. That is to say, the first program on the list dominates the second, which dominates the third and so on. The first program is placed in the grid by selecting one of the points in the cloud. The selected point becomes the center of the mass, which forms as a pixelated sphere around that point (Figure 2). Once a program occupies a space, all points within it are subtracted from the cloud. The next program repeats the process until all the programs are massed out. The proximity is determined by measuring the distance between origin points. The relationships are weighted in terms of their needs by multiplying the distances by various factors, the higher priority the adjacency need, the higher the multiplication factor. This multiplication factor can also be negative to repel program elements that need to be isolated from one another. The algorithm calculates the total value of all relationships by performing mass addition. This figure is the fitness score. The evolutionary process tries to minimize this number as much as possible. The idea is that a lower score creates a closer proximity and a more effectively adjacent situation.

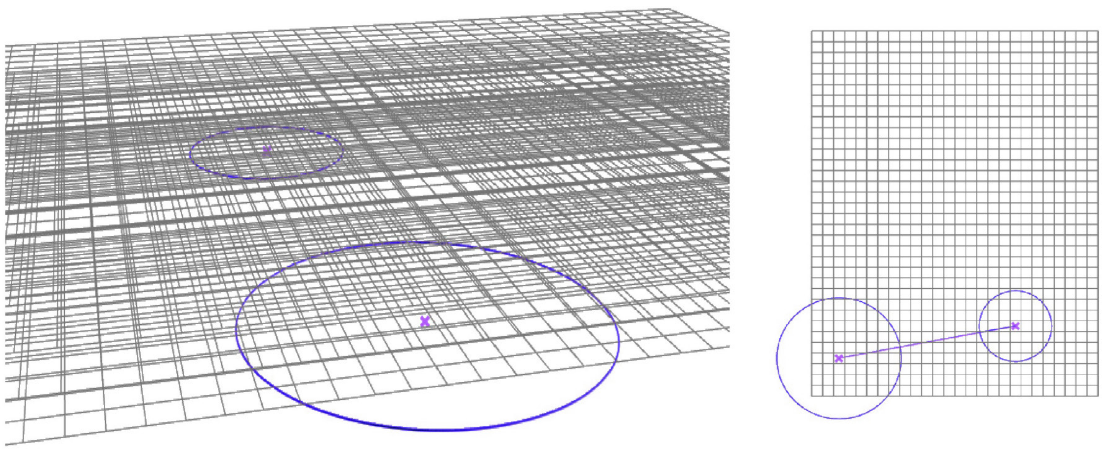


Figure 1: Screenshots showing the origin points selected in the grid and the relative floor area.

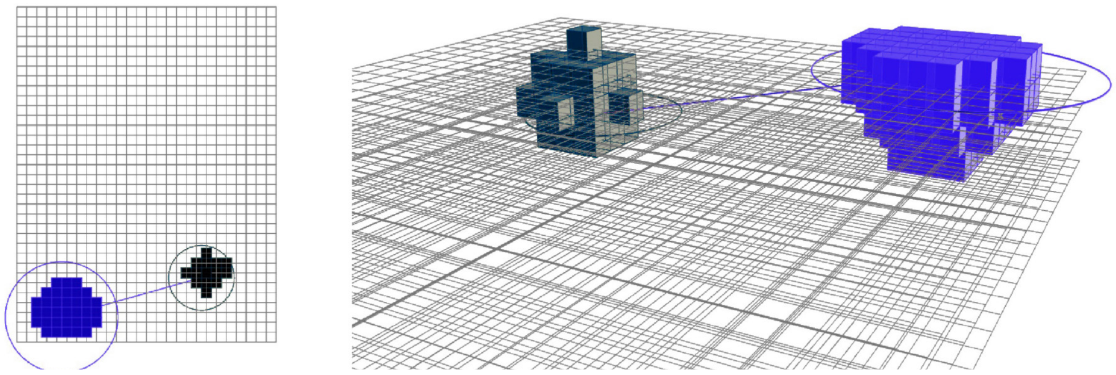


Figure 2: Areas are translated into "pixelated spheres".

To create the evolutionary element in the process, the computer is allowed to manipulate all the origin points to create iterations. The point cloud within the shell is a list, and Galapagos can iterate using the sliders connected to that list.

2.3. Tool Demonstration and Steps

A series of generic tests help to demonstrate how the system works and where it needs refinement. This test was run with nine programmatic elements. Those are broken into three groups represented by different colors (yellow, magenta, and cyan). Each group member wants to be adjacent to the others in the same group. The site outline for this experiment is for square blocks. There are four floors to be occupied. The Grasshopper definition (Figure 3) is implemented using the steps outlined in Table 2 and described in more detail below.

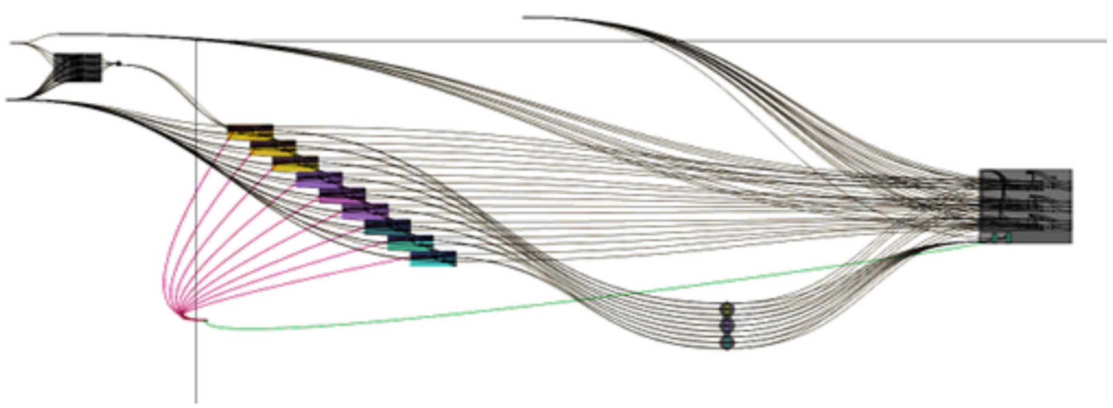
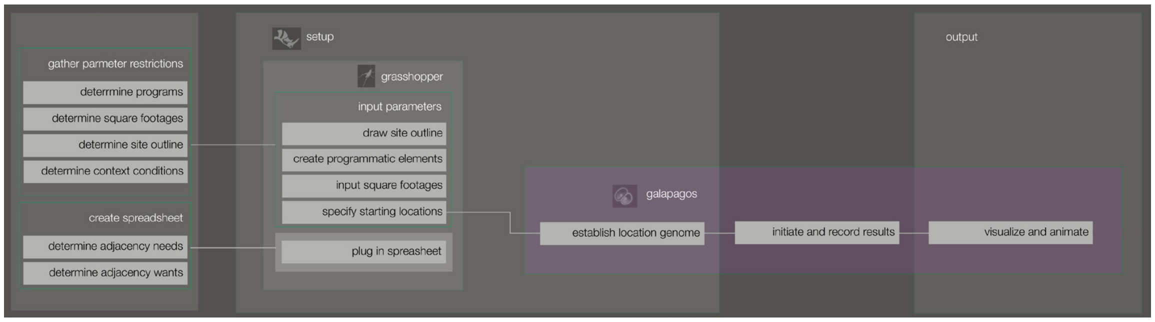


Figure 3: Grasshopper definition, screenshot where color represent program groupings.

Table 2: Diagram outlining the software programs and steps used to generate adjacency diagrams.


gather parmeter restrictions
determine programs
determine square footages
determine site outline
determine context conditions
create spreadsheet
determine adjacency needs
determine adjacency wants
setup
grasshopper
input parameters
draw site outline
create programmatic elements
input square footages
specify starting locations
plug in spreasheet
galapagos
establish location genome
initiate and record results
visualize and animate
output

Step 1: Establish site-specific initial geometry and offset levels
The starting parameters are customizable. The user sets up a grid that the spaces are built within. The number of cells and size of cells can be set to a specific coarseness. The program can also build the grid inside of a two-dimensional site outline to accommodate non-rectilinear sites. To account for multiple levels the grid can be offset vertically any number of times and the floor-to-floor height is adjustable (Figure 4).

Step 2: Assign hierarchy of spaces with list order
When the program activates, the order of the list of program elements is the order of importance. The first element in the list is ranked as the most important. This hierarchy is applied when spaces are divided and re-assigned giving priority to the most important program elements.

Step 3: Input square footage and set initial locations for each program element
The square footage for each space is set using a slider in the Grasshopper definition. The various program elements are assigned an initial origin point in the grid for a starting location. This point is center of the pixelated sphere, which represents a program element in the definition.

Step 4: Generate adjacency lines between program elements
The definition creates a line connecting the origin points for every adjacency requirement. A lower value indicates a closer proximity and more optimized adjacency. The definition adds up the total distance of all adjacency lines, which can be weighted to create greater attraction or repulsion in order to prioritize certain adjacencies over others or ensure two program elements are isolated from one another. The Galapagos evolutionary solver will try to minimize this total distance to improve fitness.

Step 5: Run Galapagos and record visualization of every iteration
The Galapagos "genome" is solely made of the origin location sliders, which allows it to rearrange the location of the program elements within the grid in order to improve fitness by creating the lowest total adjacency line distance (Figure 5). The areas inside the program circles are translated into pixels on the grid and extruded into three-dimensional volumes by the Grasshopper definition. All iterations are recorded in sequence along with their fitness score. The results (Figure 6) can be exported as an animation.

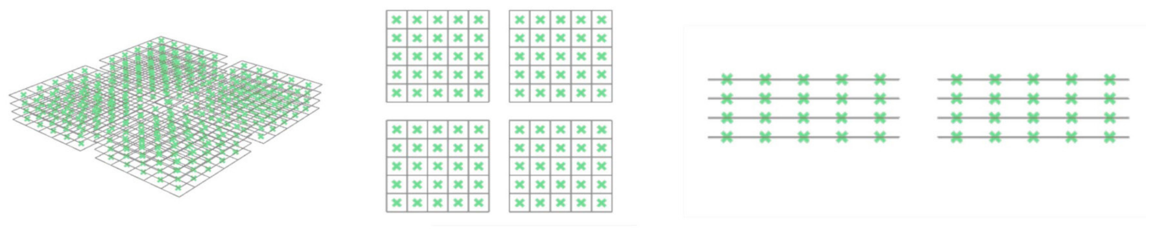
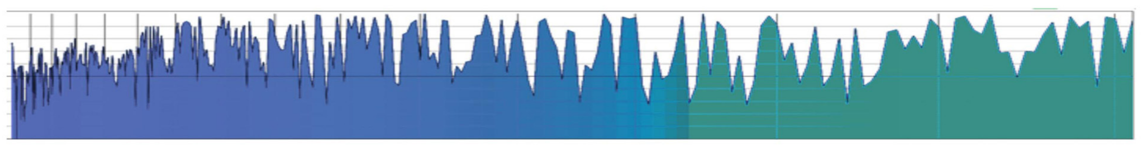


Figure 4: Point cloud formation, derived from initial geometry: site outlines and offset floors.



20 minutes
306 iterations

Figure 4: Point cloud formation, derived from initial geometry: site outlines and offset floors.

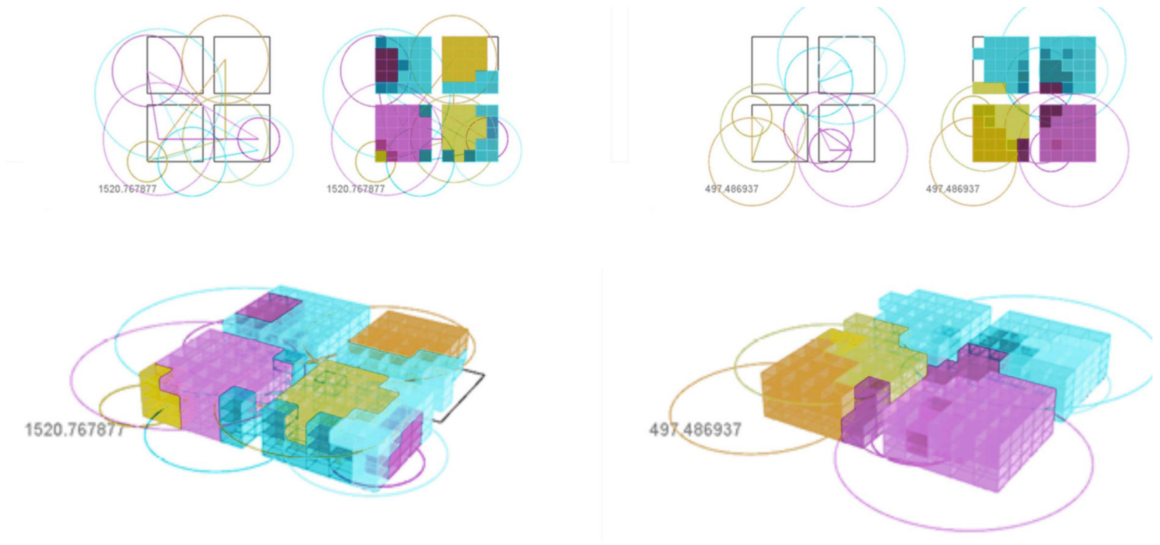


Figure 6: Least fit result (left) and most fit result (right).

3.0 CASE STUDY

Using thirty-six program elements (Figure 7) and the adjacency matrix for an actual hospital project (Table 1), a hypothetical sloping site with an irregular outline and a three story building height was chosen to test the Evolutionary Parametric Program Adjacency Script (EP-PAS).

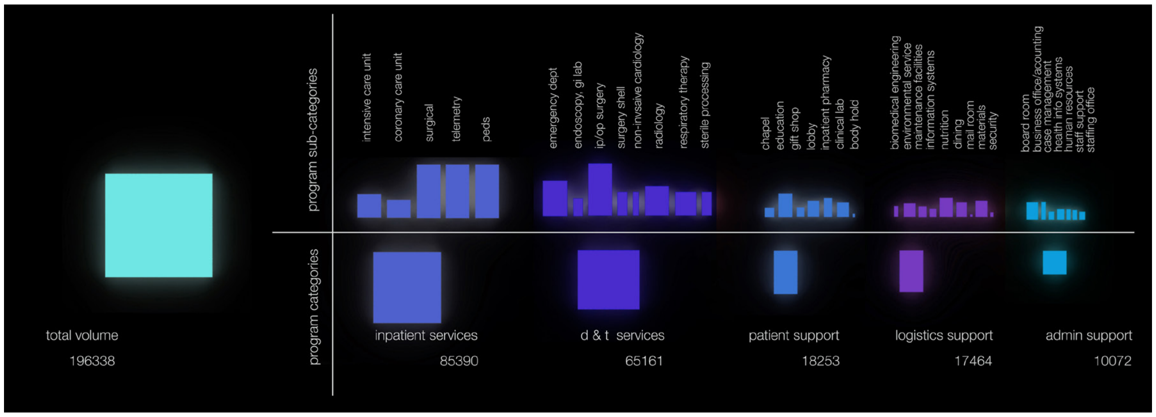


Figure 7: Program elements and relative square footages for a hospital.

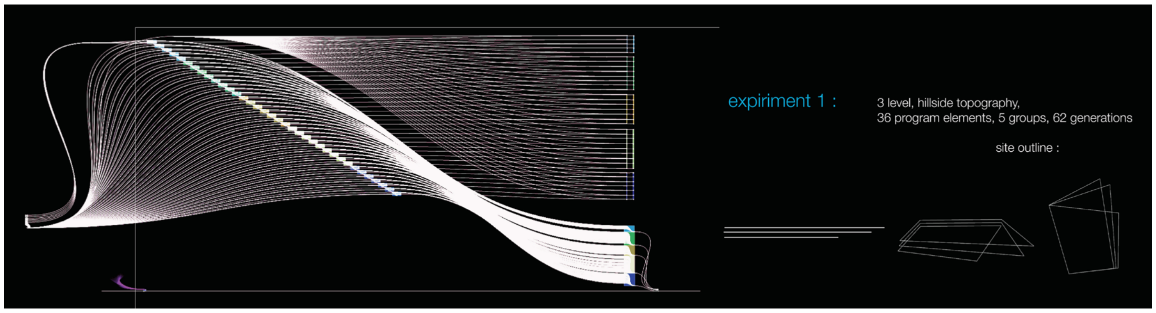

expiriment 1 :
3 level, hillside topography,
36 program elements, 5 groups, 62 generations
site outline :

Figure 8: Grasshopper definition (left) and site constraints (right).

3.1. Script Deployed

Figures 8-11 depict the Grasshopper definition for this specific set of programs and adjacencies as well as the resulting Galapagos generated iterations, using the steps outlined in Section 2.2.

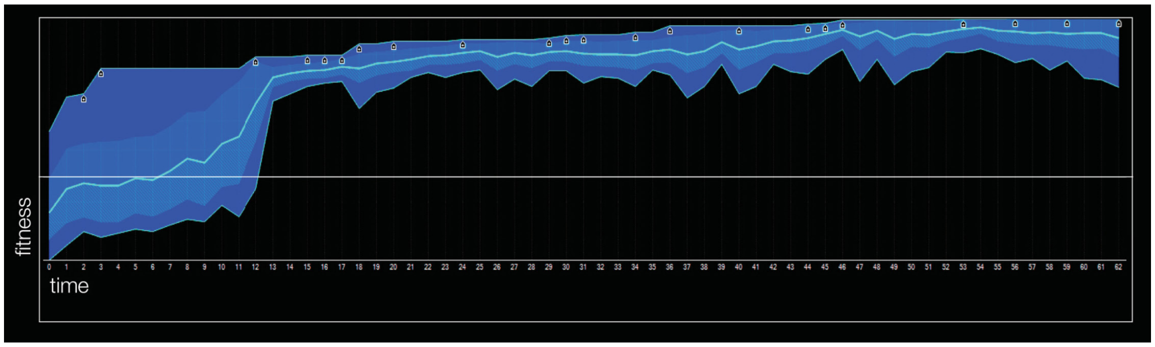


Figure 9: Graph depicts fitness variation (y) over generation number (x), where 62 generations took eight hours to compute.

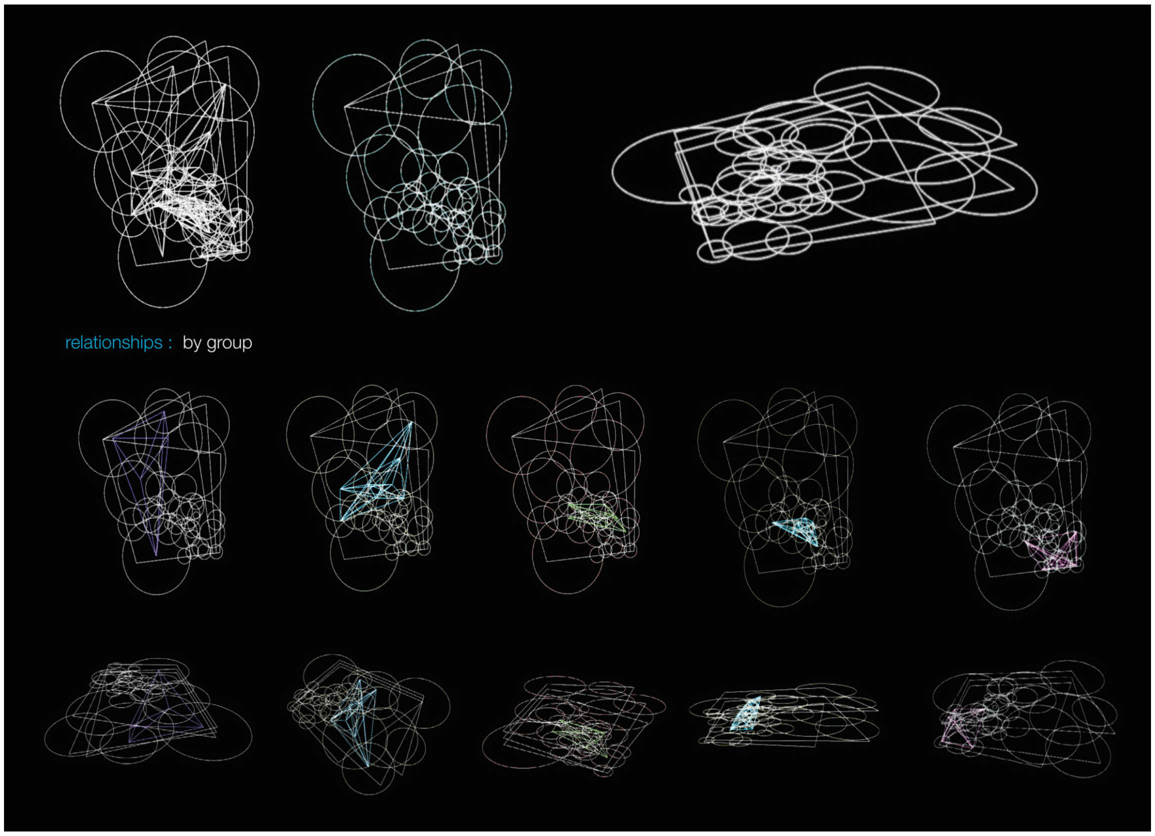
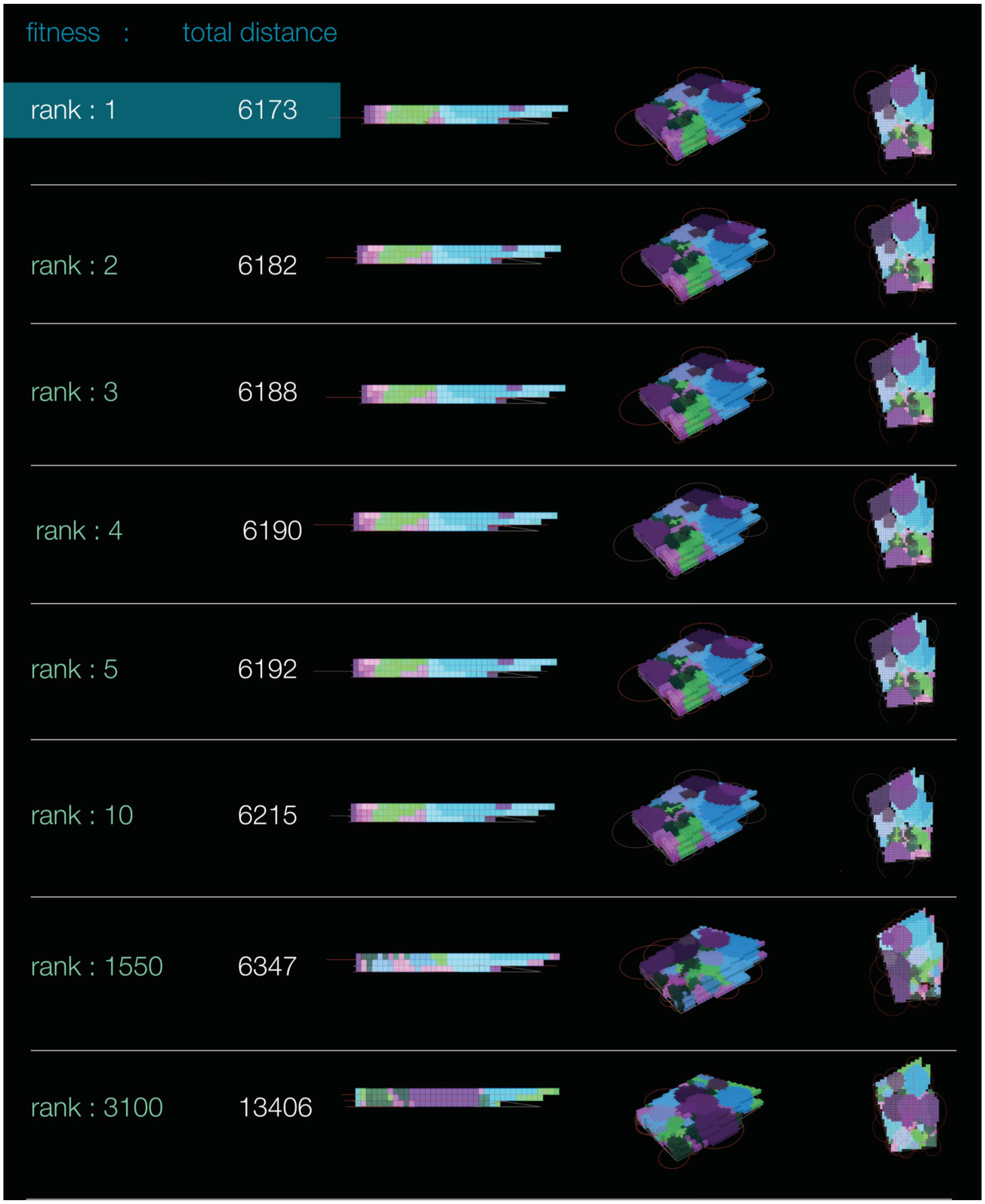


Figure 10: Generations, highlighting the geometries and distances used to optimize adjacencies and relationships by program type ("group").



fitness :	total distance
rank : 1	6173
rank : 2	6182
rank : 3	6188
rank : 4	6190
rank : 5	6192
rank : 10	6215
rank : 1550	6347
rank : 3100	13406

Figure 11: Least fit result (bottom) and most fit result (top) in section, isometric and plan.

3.2. Case Study Discussion

The script can run indefinitely, although it reaches a point where the results show no serious improvement in fitness. At these points it is best to either stop the process and take best result, or continue by restarting the process and hope for a different type of mutation to take hold of the evolutionary direction. There are clearly many ways to achieve similar results within the structure of the parametric tools, so this is just one solution to a problem. Better definitions are ones that can accomplish complex moves within a few commands, this makes the load on the computer exponentially lighter and therefore it produces more results and is easier to use. In this particular example, the most significant improvement in fitness occurred by the 13th iteration (Figure 9), with only slight improvements in fitness over the next 50 iterations. Despite these small improvements in fitness, these last 50 iterations could still be valuable to the architect as they highlight a variety of solutions that are similar in performance rather than one "right" or best answer.

There are several limitations to this script. This script currently treats program elements as three dimensional and does spread program elements over multiple floors during the optimization process. For many program elements, this situation is likely not ideal. Further development of the script would be necessary to put boundaries on programs that need to be located on the same story – slowing down each iteration. As outlined in the Methodology section, this script uses a cumulative total distance taken between each program element and attempts to minimize it as the definition of fitness or optimization. This could lead to iterations where there is one long and inefficient relationship compensated by other tightly packed elements. This possible iteration could rank better than one where all elements are equidistant from each other. It is necessary and - given the graphic outputs of the script – possible for the user to manually check for these types of situations and rule out iterations that have obvious inefficiencies. The authors would encourage users of this script to look at iterations with similar fitness scores to see which result would work best with other architectural constraints not currently accommodated in this research.

The advantages of this script, particularly with respect to healthcare, are that the program elements must be prioritized when the script is generated. This allows critical spatial relationships to be built into the script to avoid impractical solutions. For example, the emergency room can be placed near a particular parking lot or driveway to ensure quick access rather than allowing the script to place the emergency room on the third floor even though that might be the most "fit" location for it in terms of adjacencies. Other anchors can be programmed to ensure particular views or orientations as needed. Another advantage is that programs that cannot be adjacent to one another for reasons of noise or contamination can be given a negative value for distance in the algorithm to reward the script for placing those elements further apart. The major advantage in using this script for space planning layout in healthcare is the fact that a large number of program elements and adjacencies are required in contemporary healthcare. It would be challenging during conceptual design to iterate space planning layouts more than a few times manually, given this complexity and potentially better layouts could be missed.

4.0 CONCLUSION

The Grasshopper/Galapagos script presented in this article is best applied to projects with complex programs with multiple adjacency requirements in early design phases. It can give designers data to inform their process and ultimately justify their decisions to clients who want to see a more data-driven or evidence-based approached to architectural design. The adaptability of the tool is critical to its usefulness. It has been simplified in terms of the scripting to a large degree, but could perhaps still be refined to produce faster results. One observation from the experiments, however, is that the best way to save time is by sacrificing certain amounts of flexibility. The more customizable the tool becomes, the more complex the calculation. A downside to this flexibility is that the user must use the Grasshopper interface to input the program elements. It would be ideal for non-Grasshopper users to be able to use the script as a tool by inputting an excel spreadsheet of program elements and their required adjacencies. This is one future line of development for this project.

Creating the range for the results is the task of the designer, and the narrower the range, the more meaningful results are. In other words, narrowing the field where variables can occur creates more pertinent results. The end results of this definition are diagrammatic, so they serve as a source of information that can be observed as a metric then incorporated into design. It could be expanded into a more literal space, with doors and windows, and the results could become more directly applicable to building design in the later stages. There will always be limitations in using genetic algorithms to optimize space planning layouts or any other criteria. In the case of using Grasshopper with Galapagos, this comes in the necessary simplification of complex relationships into a single number to be maximized or minimized. For example, instead of total cumulative distance as a measure of fitness and proxy for optimized adjacency, this research could have used an average distance or ratio of shortest to longest distance. The authors specifically chose cumulative distance to allow for the repulsion of elements that need separation that would be more difficult to account for in averages and ratios. Still, this single number as an adjacency proxy can be misleading when not taken into consideration with the other architectural criteria that must be considered when organizing spaces in a building.

The introduction of the evolutionary process is extremely useful in creating iterative design tools. It should be noted these are tools, not an artificial intelligence created to replace architects or their decision making. These evolutionary design tools will only be as good as the user who programs the scripts and adjacency information generated in collaboration with the building's users. Consequently, computer programming should already be a skill in demand in architectural practice. Thankfully, visual programming tools that work in tandem with drafting programs make the learning and integration of parametric design and genetic algorithms more rapid. After being skillfully designed and programmed by the user, genetic algorithms are a way in which computer can independently process time consuming iterative work. The ideal situation for the tool involves a combination of parametric operations with evolutionary solving. The evolutionary modeling methods discussed in this article could be used on a much wider range of situations. This research explored adjacency, but any parameter or combination of parameters could be driving the model process including daylighting, structural efficiency, thermal comfort, and energy use, and many academics and professionals are already working on these types of tools. As a diagramming tool, parametric engines allow designers to start their process from an informed standpoint. The authors believe these methods are sound, but it will take further tests that at the very least are compared to the results of humans attempting the same optimization without the assistance of genetic algorithms.

Acknowledgments

The authors would like to thank the Van Evera and Janet M. Bailey Fund of the Oregon Community Foundation for supporting the graduate seminars in which this research was conducted. The authors would also like to thank ZGF Architects for their collaboration on this research.

REFERENCES

[1] Sterling, B., (2009). "Blobitecture and Parametrics", *Wired*, July 23.

[2] Silber, J., (2007). *Architecture of the Absurd: How "Genius" Disfigured a Practical Art*, New York, NY: Quantuck Lane Press.

[3] Safire, W., (2002). "On Language: Defenestration", *New York Times*, December 1.

[4] Weng, Z., Ramallo-González, A. P., and Coley, D. A., (2014). "The Practical Optimization of Complex Architectural Forms", *Building Simulation*, Vol. 8, No. 3, pp. 307-322.

[5] Sitek, M., and Masły, D., (2014). "Parametric Design of Airport Passenger Service Areas", *Advances in Human Factors and Sustainable Infrastructure: Proceedings of the AHFE 2014 Conference*, pp.138-146.

[6] Dutta, K., and Sarthak, S., (2011). "Architectural Space Planning Using Evolutionary Computing Approaches: A Review", *Artificial Intelligence Review*, Vol. 36, No. 4, pp. 311-321.

[7] Harriman, T. J., (1958). "Technical Management of Missile Systems", *Engineering Management, IRE Transactions*, Vol. 4, pp. 133-135.

[8] Swihart, J., and Wimpress, J., (1964). "Influence of Aerodynamic Research on the Performance of Supersonic Airplanes", *Journal of Aircraft*, Vol. 1, No. 2, pp. 71-76.

[9] Roller, D., (1991). "An Approach to Computer-Aided Parametric Design", *Computer-Aided Design*, Vol. 23, No. 5, pp. 385-391.

[10] Motta, E., (1999). *Reusable Components for Knowledge Modeling: Case Studies in Parametric Design Problem Solving*, Amsterdam, The Netherlands: IOS Press.

[11] Woodbury, R., (2010). *Elements of Parametric Design*, New York, NY: Routledge.

[12] Thompson, D., (1942). *On Growth and Form*, Cambridge, UK: The University Press.

[13] Carroll, S., (2005). *Endless Forms Most Beautiful: The New Science of Evo Devo and the Making of the Animal Kingdom*, New York, NY: Norton.

[14] Fogel, L., (1966). *Artificial Intelligence through Simulated Evolution*, New York, NY: Wiley.

[15] Holland, J., (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, Ann Arbor, MI: The University of Michigan Press.

[16] Goldberg, D., (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning, Reading*, UK: Addison-Wesley.

[17] Lenski, R.E., Ofira, C., Collier T.C., and Adami, C., (1999). "Genome Complexity, Robustness and Genetic Interaction in Digital Organisms", *Nature*, Vol 400, pp. 661-664.

[18] Frazer, J., (1995). *An Evolutionary Architecture*, London, UK: Architectural Association.

[19] Hensel, M., Menges, A., and Weinstock, M., (2010). *Emergent Technologies and Design*, New York, NY: Routledge.

[20] Weinstock, M., (2010). *The Architecture of Emergence: The Evolution of Form in Nature and Civilisation*, New York, NY: Wiley.

[21] Gero, J. S., and Kazakov, V. A., (1998). "Evolving Design Genes in Space Layout Planning Problems", *Artificial Intelligence in Engineering*, Vol. 12, No. 3, pp. 163-176.

[22] Jo, J. H., and Gero, J. S., (1998). "Space Layout Planning Using an Evolutionary Approach", *Artificial Intelligence in Engineering*, Vol. 12, No. 3, pp. 149-162.

[23] Michalek, J., Choudhary, R., and Papalambros, P., (2002). "Architectural Layout Design Optimization", *Engineering Optimization*, Vol. 34, No. 5, pp. 461-484.

[24] Azadivar, F., and Wang, J., (2000). "Facility Layout Optimization Using Simulation and Genetic Algorithms", *International Journal of Production Research*, Vol. 38, No. 17, pp. 4369-4383.

[25] Medjdoub, B., and Yannou, B., (2000). "Separating Topology and Geometry in Space Planning", *Computer-Aided Design*, Vol. 32, No. 1, pp. 39-61.

[26] Wong, S. S., and Chan, K. C., (2009). "EvoArch: An Evolutionary Algorithm for Architectural Layout Design", *Computer-Aided Design*, Vol. 41, No. 9, pp. 649-667.

[27] Zawidzki, M., Tateyama, K., and Nishikawa, I., (2011). "The Constraints Satisfaction Problem Approach in the Design of an Architectural Functional Layout", *Engineering Optimization*, Vol. 43, No. 9, pp. 943-966.

[28] Lorenz, W. E., Bicher, M., and Wurzer, G., (2015). "Adjacency in Hospital Planning", *Mathematical Modeling*, Vol. 8, No. 1, pp. 862-867.

[29] Scrawford, S., (2010). "Grasshopper Space Planner", Retrieved from http://lmnts.lmnarchitects.com/parametrics/grasshopper-space-planner/ on 1/18/2015.

[30] Scrawford, S., (2013). "Update to GH Graphic Programming Tool", Retrieved from http://lmnarchitects.com/tech-studio/parametrics/graphic-programming-round3/ on 7/24/2015.

[31] Syp, M., (2010). "Architecture: Realtime Physics for Space Planning", Retrieved from http://vimeo.com/15563685 on 3/3/2015.