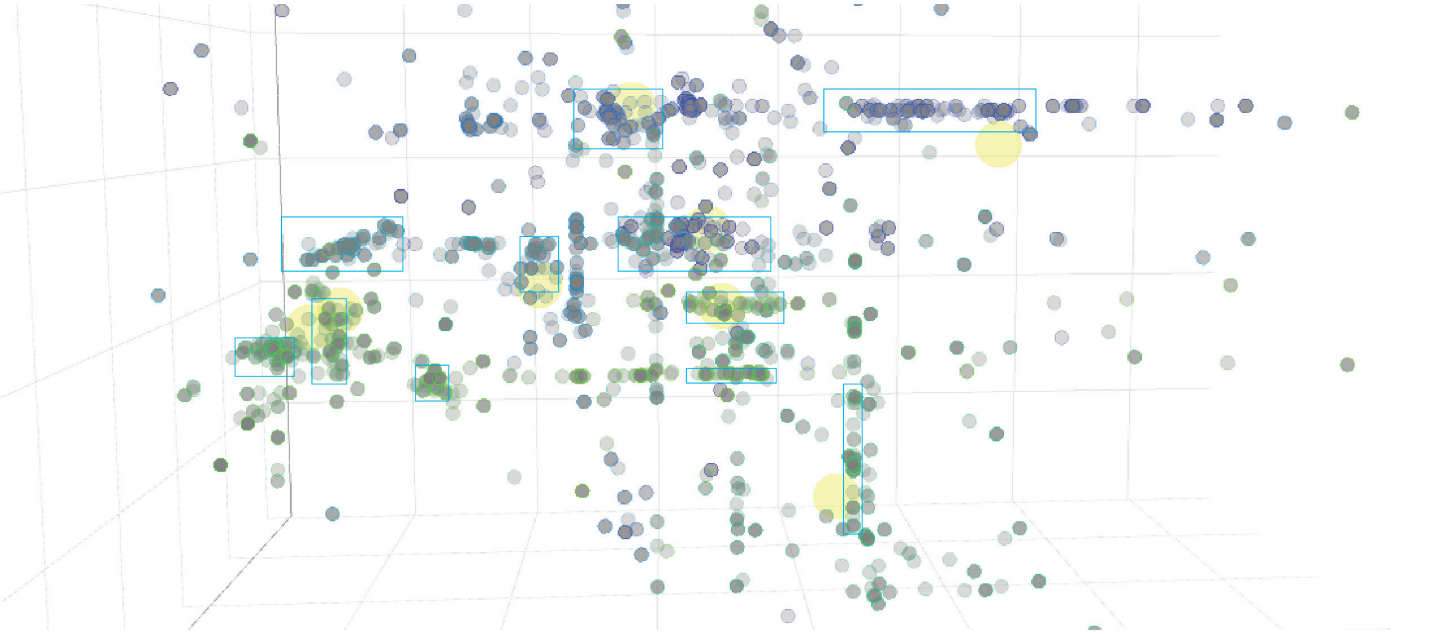
Emergent Syntax

Machine Learning for the Curation of Design Solution Space

Christian Sjoberg
UNC Charlotte

Christopher Beorkrem
UNC Charlotte

Jefferson Ellinger
UNC Charlotte

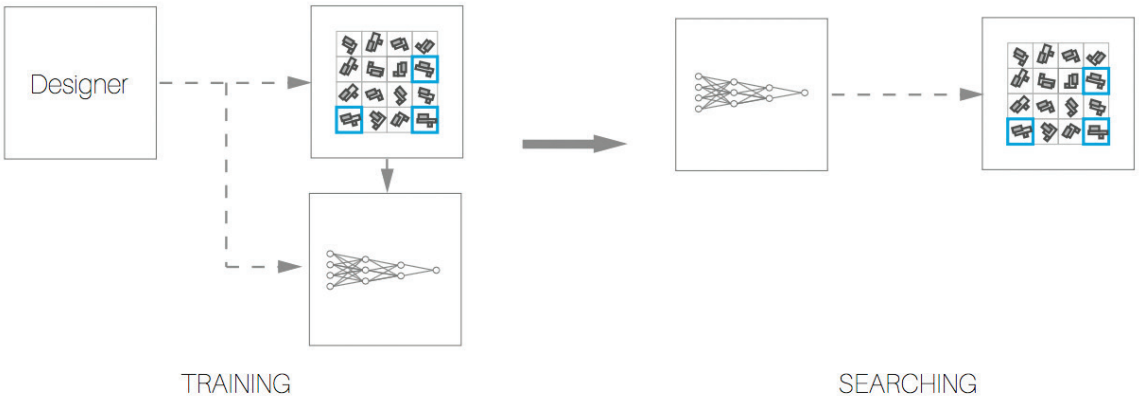


1

ABSTRACT

The expanding role of computational models in the process of design is producing exponential growth in parameter spaces. As designers, we must create and implement new methods for searching these parameter spaces, considering not only quantitative optimization metrics but also qualitative features. This paper proposes a methodology that leverages the pattern modeling properties of artificial neural networks to capture designers' inexplicit selection criteria and create user-selection-based fitness functions for a genetic solver. Through emulation of learned selection patterns, fitness functions based on trained networks provide a method for qualitative evaluation of designs in the context of a given population. The application of genetic solvers for the generation of new populations based on the trained network selections creates emergent high-density clusters in the parameter space, allowing for the identification of solutions that satisfy the designer's inexplicit criteria. The results of an initial user study show that even with small numbers of training objects, a search tool with this configuration can begin to emulate the design criteria of the user who trained it.

1 Identification of clusters within a design solution space. Clusters emerge in regions of relatively high fitness based on the user-trained model.



2 Diagram representing the training and search processes. 2

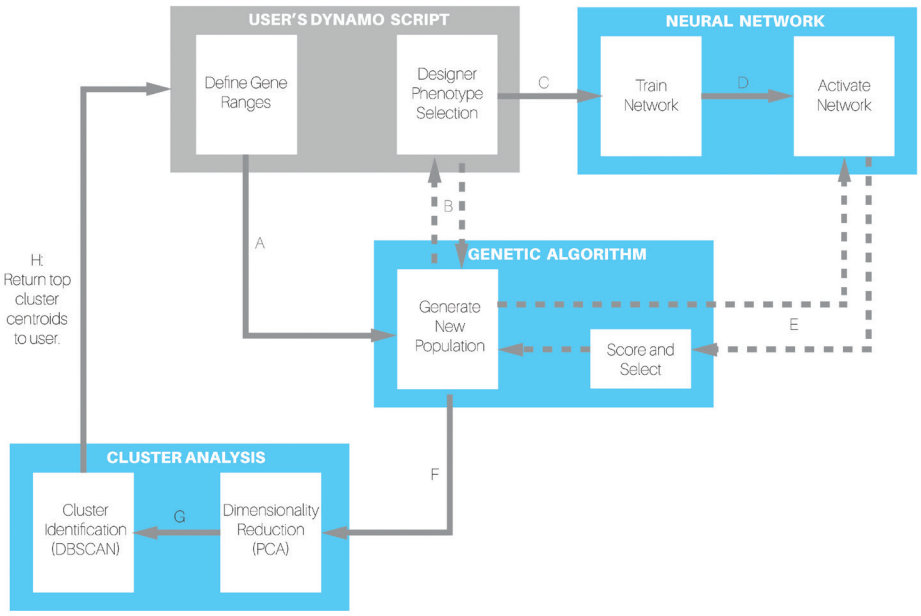
INTRODUCTION

As the influence of computation on design continues to grow, designers have the ability to optimize and account for high numbers of variables. This results in a high-dimension solution space and an exponentially increasing number of possible variations. Optimization strategies enable designers to navigate toward solutions which are quantitatively superior in pre-defined tests, but that limit the designer's ability to search the solution space for qualitative aspects. Fully accepting a quantitatively optimized solution poses an issue for designers. Calculating a top performing design based on limited or standardized criteria reduces the role of the designer. This has the potential to situate design as a secondary layer on calculated models. This workflow limits the ability of designers to holistically consider the aspects of design problems that are abstract or qualitative. To counteract this side effect, designers may include quantitative descriptions of qualitative criteria into their computational models. In doing so, they constrain the solution space based on their pre-conceived expectations of the possible outcomes. In either situation, the designer is limited in their exploration of the full solution space, which reduces opportunities to discover emergent properties of designs resulting from unexpected combinations of parameters.

If we are to avoid this, we must re-evaluate the possible methods for navigating vast solution spaces and create models that can evaluate solutions based on both quantitative evaluations and qualitative properties. In the context of this project, the term qualitative is used to describe features of a computationally generated design that are visibly recognizable as favorable to the designer, but are not measured by the designer's script as an evaluation of the design's performance. These features can be understood to be the emergent result of the combined state of many parameters of the design. Due to the abstract and subjective nature of qualitative criteria, designers can benefit from a process of selecting solutions based on their own expertise and intuition, rather than the simplification of a problem and constraint of possible solutions.

If a tool used by the designer requires them to explicitly define each of the criteria at the onset of the search for a solution, the process of exploration, discovery, and redirection is lost. To create tools that more naturally resemble the designer's process, we can begin to develop a bottom-up approach to the creation of constraints for the solution space. The collection of a designer's choices in their own process of curating possibilities to a design problem contains patterns that can be used to form a model of their inexplicit criteria.

This project seeks to develop and test a method for searching vast solution spaces based on inexplicit evaluation criteria demonstrated by designers using the system. The pairing of artificial neural networks with genetic solvers provides a means of searching solution spaces for learned selection criteria. Machine learning methods allow for the formation of evaluative criteria based on patterns learned from the user's selection of design options. As designers are iteratively presented with design choices, they will choose the solutions that meet their personal aesthetic, abstract, or qualitative criteria for design. The encoded design parameters for the possible selections, as well as the actual user selections, are recorded as a training set for an artificial neural network. A network trained on this selection data acts as a model of the user's evaluation of design options. Using this model as the fitness function for an evolutionary solver provides a mechanism for searching solution spaces that contain more design possibilities than could possibly be individually evaluated by a human designer. Learning from the designer's evaluation of a sample of the possible solutions, this system demonstrates an ability to identify additional high performing regions of the solution space based on the user's inexplicit qualitative criteria.



3 System logic diagram representing the roles of each algorithm used.

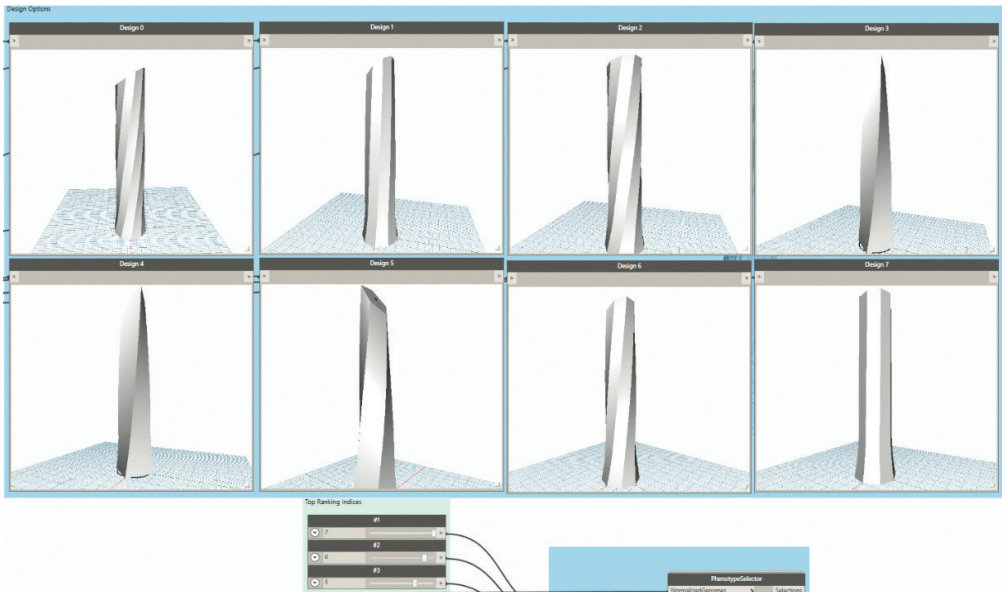
BACKGROUND

The use of machine learning as a method of solving complex problems has been a reality of computer science since the 1950s. Early models of machine learning, such as Arthur Samuel's 1952 model of the game of Checkers, proved for the first time that a machine could learn to play a game better than its creator in a short period of time (Samuel 1957). This is considered to be the first developed machine learning algorithm. In the realm of design, it is conceivable that such a moment might also occur, where a system could eclipse the ability of human designers to consider and respond to the vast numbers of variables and relationships that influence design.

The writings and work of Nicholas Negroponte of MIT illustrate the recognition of this possibility within the design field. His work contemplated authorship in the era of machine-aided design. Posing critical questions regarding the role that smart machines would play in design, he hypothesized that computing systems that he deemed "Architecture Machines" would one day go beyond the automation and repetition of tasks that are tedious to the architect. He also proposed that such systems would not only learn about architecture, but would "learn about learning about architecture" (Negroponte 1967). This concept of tools learning from our design processes serves as a driver of this project.

The use of genetic algorithms for design purposes, along with the further exploration of the concepts of tools that learn, were further explored for the purposes of design throughout the second half of the 20th century. Machine learning using genetic algorithms was explored as a means of developing shape grammars and altering the "state space" of a design (Gero, Louis, and Kundu 1994). The conception of the design as a state being encoded as a sequence of parameters pushed the development of further algorithmic design methodologies.

Searching solution spaces for desired metrics has become one of the most prominent methods of solving complex design problems in recent years. Black Box Studio at Skidmore, Owings, and Merrill (SOM) began exploring the ability to apply search algorithms to architectural optimization problems in the early 2000s. Their work focused on the use of genetic algorithms to search for optimized solutions to daylighting performance in tower massing (Besserud 2008). This work laid out the process for applying evolutionary genetic solvers to design problems. By identifying the parameters that could be manipulated in the form of the tower, they provided the algorithm with the genomes to manipulate. The fitness function or evaluation criteria for genomes in this study was based on the geometry's performance in a daylighting analysis. Using these inputs, the genetic algorithm iteratively creates and tests sets of individual design genomes and evaluates their performance. Although genetic algorithms are very effective at finding the single highest performing solution, they identify the goal of their work as the ability to visualize and evaluate a range of many well-performing design solutions (Besserud 2008). This work shows the increasing interest of designers and architects in the curation of possible design solutions through algorithmic methods. Recent projects by research groups at Autodesk have further explored the use of



4 Example representation of generated design options in Autodesk Dynamo using standard Watch3D components.

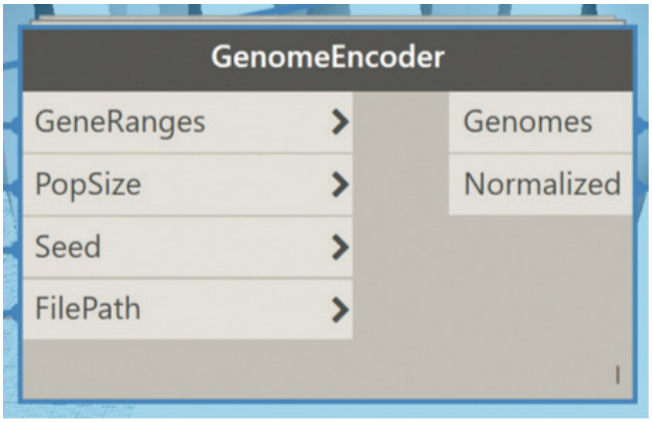
genetic algorithms as part of a "generative design" process. The application of multi-objective optimization algorithms to the task of space planning allows their system to balance six different criteria in the formation of a plan for an office space (Nagy et al. 2017).

Several tools have become available for designers seeking to incorporate genetic algorithms in the architectural design process, most notably Galapagos for Rhino Grasshopper (Rutten 2013) and Optimo for Revit Dynamo (Asl et al. 2015). These tools allow designers using visual scripting platforms to use genetic algorithms without needing to code the entire process. The increasing complexity of models of design problems in architecture has furthered the exploration of methods for navigating vast solution spaces for pre-defined metrics, but current methods lack the ability to search for more abstract qualities that represent the designer's expertise or intuition.

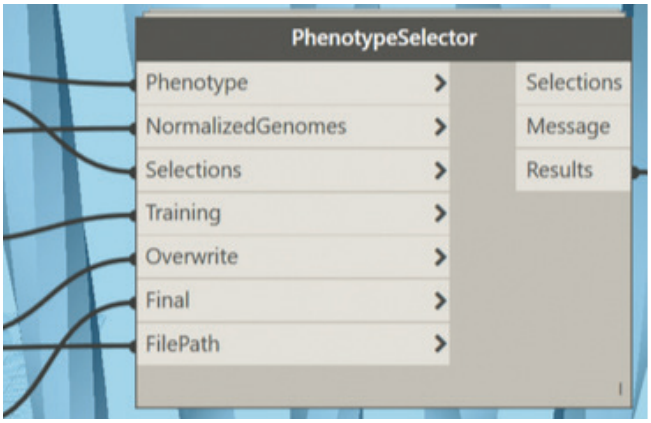
The application of machine learning methods for the formation of a model of the inexplicit evaluation heuristics of designers could enable searches of solution spaces to return design options representing the qualitative evaluation of the designer or designers making a selection.

METHODOLOGY

The implementation of this project is intended to accomplish two overarching goals: training a neural network based on user selection, and performing a search of the remaining parameter space of the user-created script. To better integrate this functionality into the workflow of architectural computational designers, this system is designed to be added to scripts created by Autodesk Dynamo users. As shown in Figure 4, the Dynamo environment acts as a front-end of the system, and an external Python program executes the machine learning, evolutionary search, and cluster identification tasks. Results of the search of the parameter



5 Custom component used to provide the system with ranges for all included parameters.



6 Custom component used to create training pairs based on user selection.

space are returned to the Dynamo interface to be presented to the user.

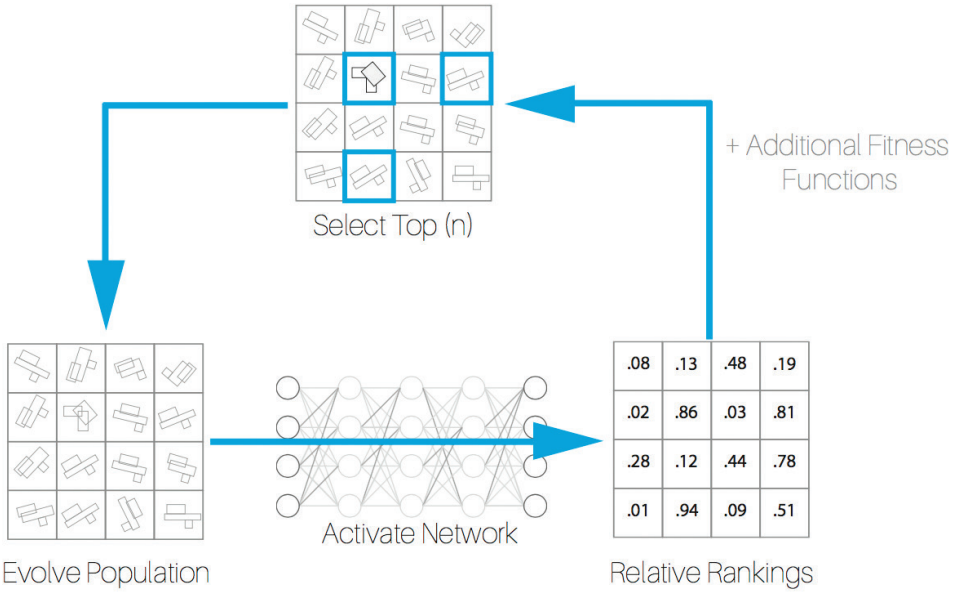
Working within the Dynamo environment allows users to append this system to the scripts, also known as graphs, that they have already developed as a means of searching for design solutions. The user defines all the parameters and geometric constraints they wish to include in their design during the creation of their original Dynamo graph. Once the user has created their graph, they will add two additional components created for this project to aid in the training task.

The first component, called GenomeEncoder, takes as input the range values for each parameter the designer wishes to include in the system (Figure 5). It is responsible for creating the random sample populations from which the user will make training selections, and recording their normalized parameters as a relative percentage of the range for each given parameter. This is accomplished by generating random parameter sequences within the range provided for each parameter of each design. The resulting two-dimensional list is then passed to the user-created portion of the graph, which generates the geometric representation of the design options.

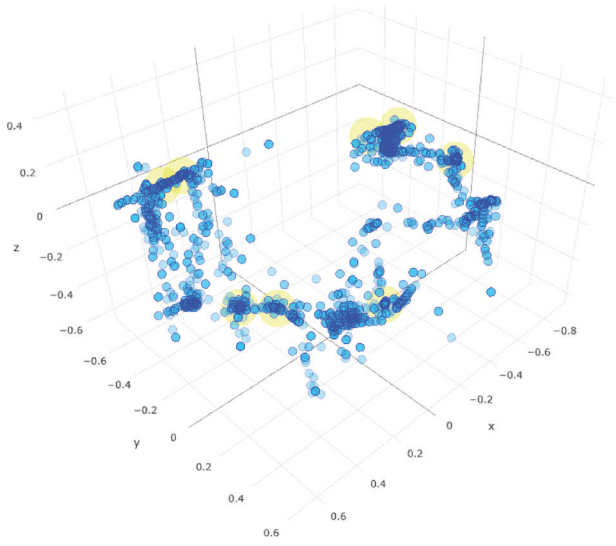
The second component the user adds to their script is the PhenotypeSelector. This component takes in the normalized population parameter set from the GenomeEncoder, as well as the selection ranking values from the user, and assembles population-ranking pairs to be passed to the Python program for training. User selections from the sample population are provided as a list of the index numbers of the designs they identify as their top choices based on their personal criteria. Once the user has made the desired number of selections, the PhenotypeSelector will export the training data to the Python program, where it will later be used to train the machine learning algorithm and search the parameter space.

This project uses a supervised neural network to create the predictive scoring model used for design evaluation. The Python program created for this project relies heavily on the machine learning libraries PyBrain (Schaul et al. 2010) and scikit-Learn (Pedregosa et al. 2011) to perform machine learning and data analysis tasks. The training set file generated by the PhenotypeSelector component serves as the primary input for the Python script. The raw population and ranking data in this file must be encoded properly before being fed to the network for training. Input-output pairs, or "training objects," must be assembled from each selection generation to create the final training set for the supervised network.

The input side of each training object contains the sequence of parameter values that created the designs in the selection population. All inputs are normalized to a range between zero and one, representing their relative percentage along the parameter's value range. This normalization prevents unequal weighting of parameters in the neural network. The output side of each training object contains the relative scores for the designs. Scores are calculated relative to the ranking of the design. The highest performing design receives a score of 1.00, the second receives a score of 0.66, the third receives a score of 0.33, and



7 Search logic diagram showing the cycle of population evolution and scoring.



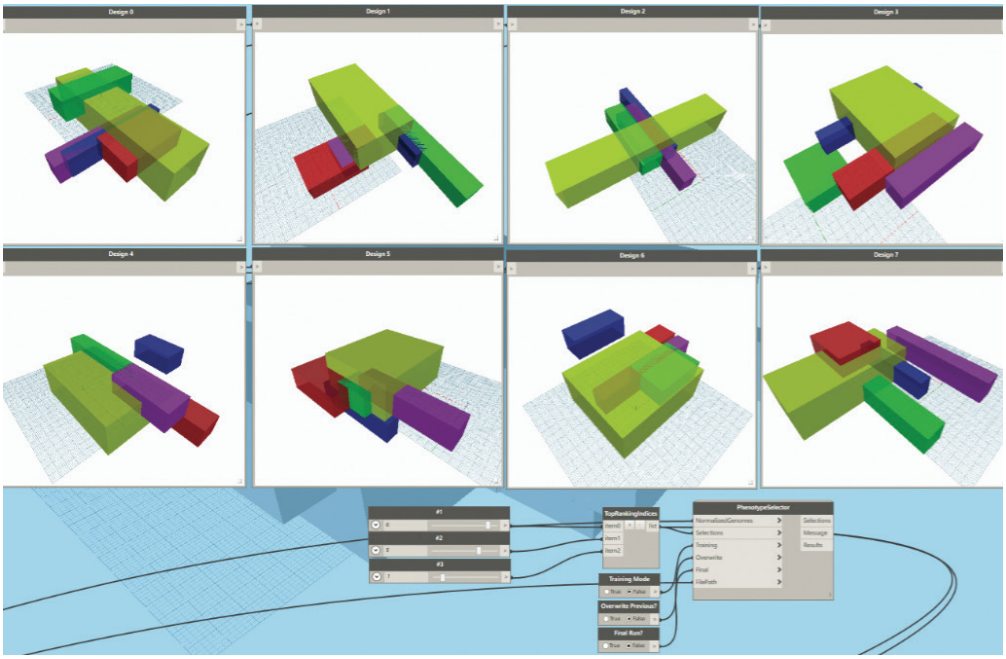
8 Identification of cluster centroids in 3D principal component analysis space using DBSCAN.

all other designs receive a score of zero. This creates a training object of input dimension equal to the product of the number of designs per generation and the number of parameters per design. The output dimension is equal to the number of designs per generation, allowing for one score value per design. The network is then configured with the same input and output dimensions as the training objects and two hidden layers by default.

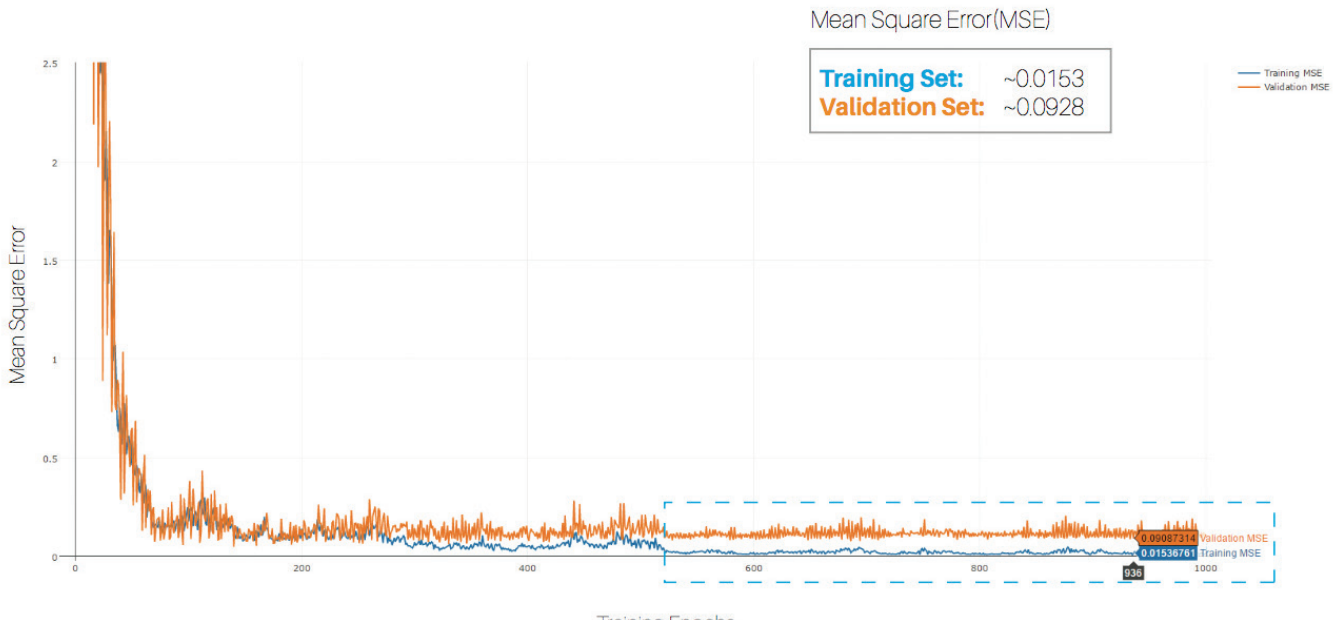
The network is being trained to evaluate designs in the context of the presented options. This enables the network to later act as a fitness function for the genetic algorithm, iteratively evaluating the design options presented to it as inputs and returning their fitness scores relative to the population as outputs. This relationship is key to the searching capability of the system and its versatility in scoring regions of the parameter space outside of the initial training set based on its model of the user's scoring heuristics.

Backpropagation training is used to train the network for multiple cycles, or "epochs," through the entire training set. Each epoch reinforces the relationships in the network by allowing the continued correction of connection weights in the network (Hecht-Nielsen 1988). The trained network is capable of being provided with the parameter values of all individuals in a population, and returning an array of score values representing the anticipated selection score for each individual. This network, when activated, is intended to represent the user's selection on the context of the input population.

To initiate the search, random sample population is formatted and provided to the trained neural network for scoring, and serves as the first generation of the genetic algorithm. The scores returned by activating the network with the sample generation provide the "fitness" for each of the individual designs in the population. Using these fitness scores, the genetic algorithm evolves the population toward the highest performing scores (Figure 7). The parameters of the top scoring individuals in the population are crossed to create the next population. As with any evolutionary solver, variation is introduced into the population through chance mutation of the genes, in this case design parameters, of the



9 Random sample population of massing designs as seen by the user during the selection process in Dynamo.



10 Training convergence MSE (Y) over training epochs (x).

individuals. Mutation ensures that new regions of the parameter space are being sampled as new generations are evolved from the previous. As random mutations create higher performing designs, they are selected and the next generation will contain more of their traits.

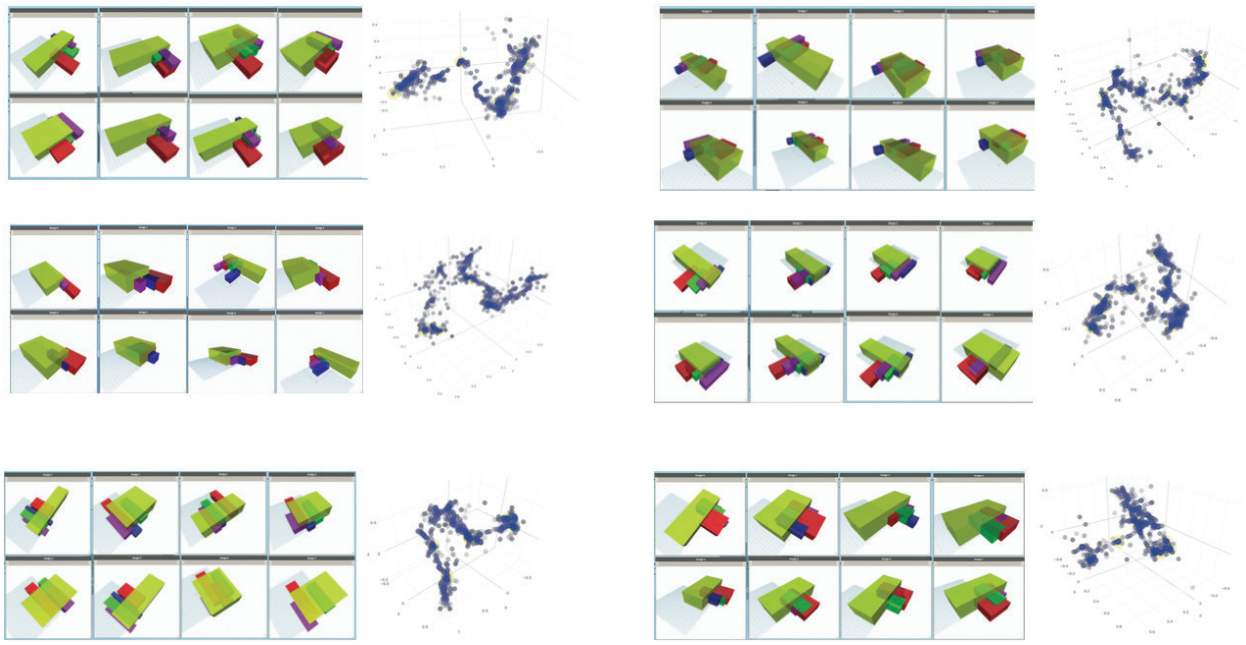
Individuals returned from all evolutions of the population begin to form high-density structures in the parameter space. The concentration of individuals in a region of the parameter space is the result of high performance in the region. This is due to the ability of high performing individuals to pass their genes on to the next generation. Identification of the parameter set at the most dense regions of the parameter space determines the highest performing individuals found by the search.

To evaluate clusters of designs with high numbers of parameters, principal component analysis (PCA) is employed for dimensionality reduction. PCA is used to reduce higher dimension parameter spaces to a specified number of perpendicular summary dimensions (Abdi et al. 2010). A three-dimensional PCA space not only allows for effective measurement of the similarity of design options, but also the ability to visualize the results of the search using a conventional scatter plot. This reduction in dimensionality makes clustering effective with higher numbers of design parameters (Jolliffe 2002).

Density-based spatial clustering of applications with noise (DBSCAN), developed by Martin Ester et al. (1996), is used for identifying the high-density clusters within the PCA space. The centroid of each cluster can then be calculated as the average of each parameter value for all individuals in the cluster. This centroid is used to represent the average design found in the cluster (Figure 8). These centroids must be converted back to the original parameter space from the three-dimensional PCA space before being returned to the user's Dynamo graph. Once this conversion is complete, the set of design parameters are returned to the user's Dynamo script, where the parameter values can be used to recreate the geometric representations of the search results.

Participants in the verification study for this system were asked to make selections of the top performing options from a set of color-coded programmatic masses representing the schematic diagram of a building (Figure 9). Participants were not asked to develop the Dynamo script that produces this geometry. This parameter space was chosen to provide the complexity necessary to test the effectiveness of the PCA analysis, while still producing geometric representations with recognizable characteristics that could be visually correlated with parameter values.

Users were provided with three explicit base selection criteria to provide a means of gauging the system's ability to replicate the criteria. Any additional features they wished to consider could be included in addition to these criteria. The first group of participants were asked to rank designs based on how well they met the following design criteria:



11 Massing designs returned by system after user training and search.

• Program C (blue) must be adjacent to or near program D (purple).

• Program A (yellow) must be on the ground level.

• Programs should have minimum overlap.

The first criterion is intended to allow for the evaluation of the system's ability to recognize criteria resulting from a limited set of relative relationships between design parameters. In this case, the five parameters of both the blue and purple masses. The second criterion is intended to allow for the analysis of the system's ability to reproduce selections that have an absolute parameter value that is favored by the user. In this case, the z parameter of the yellow mass should have a value of zero when on the ground plane. The third criterion tests the system's ability to produce results that satisfy a global relationship resulting from a combination of all parameters in the system. This requires the returned designs to satisfy different complex sets of possible combinations of parameters that result in this criterion being met. Providing participants with well-defined criteria for selection is necessary in order to evaluate whether or not the system is able to emulate these criteria in its search process.

Training sets generated through this process were used to gauge the algorithm's predictive capabilities. Quantitative evaluation of the system's predictive capabilities was conducted using cross validation. Cross-validation performance was measured by the mean square error (MSE) of the system when predicting score values for populations in both the training and validation sets. Qualitative evaluation was performed through visual assessment of the system's returned designs for adherence to the design objectives provided to the users.

RESULTS AND DISCUSSION

Figure 10 shows the MSE convergence during network training. MSE in the training set levels out at an average value of ~0.0153, while the validation set averages ~0.0928. This variance does not account for the fact that selections are made based on the relative rankings of designs to others in the population. This correlation does, however, show that even when trained with only 138 examples, the system can begin to emulate the designer's scoring.

Visual review of search results shown in Figure 11 suggests that the system can begin to emulate a generalized set of desired relationships between parameters in the design, however, it did not perform as well at recreating exact values for parameters that might be recognized by the user. For example, in most result populations the blue and purple masses are directly adjacent to each other, but the yellow mass does not always have an exact Z location of zero to place it on the ground plane. However, the yellow mass does tend to maintain the relationship to the other masses, which results from being on the ground plane, meaning that it is almost always the lowest volume in the set. The third design objective, to minimize overlap between masses, seems

to also have some representation in the results returned, as most designs show a clustering of masses along an edge of the largest mass. This would suggest that the system has not been successful at meeting this criterion in the context of the other criteria.

In comparison to the randomly generated starting conditions created by the system (Figure 8), the results (Figure 10) returned do more closely meet the design criteria. These results show that this prototype system is capable of learning inexplicit criteria. Visual comparison of the PCA space graphs generated for each user show strong similarity in their relative regions of density (Figure 11). This suggests that the similarity in the user's criteria for selection is also represented in the system's evaluation of the solution space.

One of the challenges in the development of this system is the limitation of user selection through active input. Users of the system are not likely to spend long periods of time actively making selections. This results in a dataset that is extremely small relative to most applications of neural networks and machine learning. Small training sets cause several problems for the system, including the propensity for overfitting and even greater limitations on the ability to validate predictive capabilities, due to very small numbers of validation samples as a fraction of the training set. These limitations will be addressed by the future development of more passive methods of collecting designer selection criteria. The architecture of this software is such that the means of generating the training set is left open, allowing for the possibility of scaling or modifying the user input process. The creation of training sets has the potential to be a team or even crowdsourced task. Collaborative training by a group of designers or clients could also provide a means of generating much larger and perhaps more meaningful training sets for accomplishing design tasks.

CONCLUSION

This project implements a method for capturing and emulating a designer's inexplicit qualitative selection criteria as a means of searching vast solution spaces for design options. The iterative search power of the machine, when paired with user-trained neural network models for evaluation, leverages the strengths of both the designer and the machine. This is a potentially powerful method of discovering designs favorable to the designer in solution spaces larger than the designer could ever thoroughly search. This work shows that even with small numbers of training objects, a search tool with this configuration is capable of forming a model that begins to emulate the design criteria of the user who trained it.

ACKNOWLEDGEMENTS

Thank you to Eric Sauda, John S. Gero, Rachel Dickey, and Mirsad Hazikadik for their guidance in the development of this project.

REFERENCES

Abdi, Hervé, and Lynne J. Williams. 2010. "Principal Component Analysis." *Wiley Interdisciplinary Reviews: Computational Statistics* 2 (4): 433–459.

Asl, Mohammad Rahmani, Saied Zarrinmehr, Michael Bergin, and Wei Yan. 2015. "BPOpt: A Framework for BIM-Based Performance Optimization." *Energy and Buildings* 108: 401–412.

Besserud, Keith, and Joshua Cotten. 2008. "Architectural Genomics." In *Silicon + Skin: Biological Processes and Computation*, *Proceedings of the 28th Annual Conference of the Association for Computer Aided Design in Architecture*, 238-245. Minneapolis, MN: ACADIA.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 226–231. Portland, OR: KDD.

Gero, John S., Sushil J. Louis, and Sourav Kundu. 1994. "Evolutionary Learning of Novel Grammars for Design Improvement." *AI EDAM* 8 (2): 83–94.

Hecht-Nielsen, Robert. 1988. "Theory of the Backpropagation Neural Network." *Neural Networks* 1: 445–448.

Jolliffe, Ian. 2002. *Principal Component Analysis*, 2nd ed. New York: Springer.

Nagy, Danil, Damon Lau, John Locke, James Stoddart, Lorenzo Villaggi, Ray Wang, Dale Zhao, and David Benjamin. 2017. "Project Discover: An Application of Generative Design for Architectural Space Planning." In *Proceedings of the Symposium on Simulation for Architecture and Urban Design*. Toronto, ON: SimAUD.

Negroponte, Nicholas. 1969. "Toward a Theory of Architecture Machines." *Journal of Architectural Education* 23 (2): 9–12.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

Rutten, David. 2013. "Galapagos: On the Logic and Limitations of Generic Solvers." *Architectural Design* 83 (2): 132–135.

Samuel, Arthur L. 1959. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development* 3 (3): 210–229.

Schaul, Tom, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank

Sehnke, Thomas Ruckstieß, and Jurgen Schmidhuber. 2010. "PyBrain." *Journal of Machine Learning Research* 11: 743–746.

IMAGE CREDITS

All drawings and images by the authors.

Christian Sjoberg is a Computational Designer at SGH Boston. He holds a Master of Science in Architecture and a Master of Science in Information Technology from The University of North Carolina at Charlotte. His research focuses on the implications of machine learning on the process of design.

Chris Beorkrem is an Associate Professor in the School of Architecture at the University of North Carolina at Charlotte. He currently serves as coordinator of the DesComp Dual Masters Degree in Architecture and Computer Science, and as Co-Director of the Digital Arts Center . He teaches studios and workshops in computational design and fabrication. The second edition of his book, “Material Strategies in Digital Fabrication” came out in 2017.

Jefferson Ellinger is founding partner of Ellinger/Yehia Design (E/Ye Design) and has built several projects throughout North America and won several international competitions. His work has been featured in multiple international Thal publications, exhibited at the Museum of Modern Art in New York and was highlighted in his TEDx Talk. He holds a Master of Architecture from Columbia University in New York City and a B.S. in Architecture from Ohio State University.