

# Aplysia: A Neuroevolutionary Generative Design Tool

David Gerber<sup>1</sup>, Natalie Sham<sup>2</sup>, Edith Chow<sup>2</sup>, Farzad Ghaznavi<sup>2</sup>, and Jenessa Man<sup>2</sup>

<sup>1</sup>University of Southern California, Los Angeles CA 90007, USA

<sup>2</sup>Arup, Toronto, ON M4W 3M5, Canada

djgerber@post.harvard.edu, natalie.sham@arup.com,  
edith.chow@arup.com, farzad.ghaznavi@arup.com,  
jenessa.man@arup.com

**Abstract.** This work presents the development of a proof-of-concept generative design tool for the AEC industry, named Aplysia. Aplysia has the capability to provide the designer with the ability to produce emergent design solutions from multi-objective criteria without the tradeoff between number of objectives and computational resources. easily and rapidly produce varied, performance-oriented geometries suited for concept design. The current inefficiencies with existing generative design tools are primarily due to the underlying algorithms, such as evolutionary algorithms which require significant computational resources due to large search spaces, and inconsistencies between industry requirements and provided features, such as requiring domain expert input to use these tools which make it inaccessible to many users. We present the novel use of a compositional pattern-producing network (CPPN) and the Neuro-Evolution of Augmenting Topologies (NEAT) algorithm for a building-scale structure. This paper details the software development methodology to build the tool, driven by a user-centric approach. Requirements gathering, which framed the scope of Aplysia, was completed through a use case study. The user and technical requirements were translated into a modular system architecture and user-friendly GUI. Aplysia was experimentally tested for the design of a lightweight, free-standing canopy. Our initial findings show that Aplysia improves the generative design workflow for the test case, which we argue is more adaptable to real-world AEC design problems and outline further improvements in the continual development of Aplysia.

**Keywords:** Parametric Design, Generative Design, Multi-Objective Optimization, Artificial Neural Networks, CPPN-NEAT.

## 1 Introduction

The rate of growth of different parts of an organism, however small, is fundamental to morphological shifts in physiology, as stated in “Growth and Form” by D’Arcy Thompson [1]. The generation of physical forms is a dynamic system of parametric interplay that is more accessible to designers today due to the increasing availability of higher-performing computers and advances in computational approaches. These

technologies can foster a designer’s ability to explore a plethora of new forms. Generative design is comprised of three factors: a design schema, an ability to create variations, and a means of selecting desired outcomes. The main intentions of generative design are automating exploration of the solution space and producing design in concert with some defined optimization. Specifically, In the context of Architecture, Engineering, and Construction (AEC) projects, generative design is defined as exploring and/or optimizing the design space and then reporting to the user which options can be further analyzed based on defined geometric parameters and fitness goals [2]. Typically, a model is set up with quantitative goals and an initial geometry. The model is then connected to an algorithm, usually a single or multi-objective optimization algorithm, that receives the input parameters and searches through the design space to find “high-quality designs” which are evaluated against and driven by the fitness goals. The algorithms within existing tools significantly affect the solutions generated, and they are investigated in-depth in this paper.

### 1.1 Algorithms in Generative Design Tools

Existing generative design tools primarily implement two main classes of optimization methods: iterative and metaheuristic. Iterative optimization generates a single point in every iteration, such as local direct search algorithms [3] like RBFOpt [4]. Metaheuristic optimization methods uses a set of rules to produce near-optimal solutions during multiple iterations with less computational effort, such as simulated annealing [5], particle swarm optimization [6], and genetic algorithms [7] [8].

The two types of genetic algorithms used in existing generative design tools are single-objective and multi-objective (MOGAs). MOGAs are well-suited for generative design applications in AEC since they can provide a more realistic representation of the complexity of a real-world design problem. A common approach to optimizing a multi-objective problem is to search for a subset of solutions in the Pareto-optimal set, or the Pareto front [9], which are non-dominated solutions (non-dominated relative to each other). The Pareto front is a boundary where all solutions are optimal but will have trade-offs between two or more objectives [10]. An example of a MOGA which uses this approach includes the Strength Pareto Evolutionary Algorithm (SPEA), introduced by Zitzler and Thiele in 1998 [11], as a technique for finding the Pareto-optimal set for multi-objective optimization problems. SPEA-2 was introduced in 2001 [11] and enhances SPEA by incorporating a fine-grained fitness assignment strategy, a density estimation technique, and an enhanced archive truncation method [11]. The Non-dominated Sorting Genetic Algorithm (NSGA), introduced by Srinivas and Deb, focuses on distributing the population of solutions over the entire Pareto-optimal regions [12]. NSGA-2 [13] is an improved version of NSGA, addressing the high computational complexity of nondominated sorting, lack of elitism, and the need for specifying a sharing parameter [13]. Evolutionary algorithms, however, do not guarantee convergence to optimal solutions [14], and as more objectives are involved, encodings become more complex and require domain expert input [15]. This requires more computational resources, human expertise, and makes it harder to find best fit solutions. Primarily Grasshopper and Dynamo plugins

were investigated as they are the most used in the AEC industry, including Galapagos, Octopus, Wallacei, Biomorpher, Opossum, Silvereye, GOAT, and Refinery. The algorithms analyzed were: SPEA-2 [11], SOGA [9], NSGA-2 [16], COGA [17], MSRSM [3], Guttman [3], PSO [18], BP [19], GANs [20], K-means [17], SVM [21], HyperNEAT [10], and various nonlinear optimization algorithms [22].

**Table 1.** Existing generative design tools for Rhino and Revit and their associated algorithms.

Tool	Algorithm	Description
Octopus	SPEA-2, SVM, BP, ES-HyperNEAT	<i>SPEA-2</i> : Strength pareto evolutionary algorithm; finding the Pareto-optimal for multi-objective problems
		<i>SVM</i> : supervised learning models with learning algorithms, for classification of data/regression problems
		<i>BP</i> : backpropagation; find loss function's gradient
		<i>ES-HyperNEAT</i> : extension of HyperNEAT to evolve large-scale ANNs
Galapagos	SOGA	<i>SOGA</i> : Single-objective genetic algorithm
Wallacei	NSGA-2, K-means	<i>NSGA-2</i> : Non-dominated sorting genetic algorithm; distribute population of solutions over Pareto-optimal regions
		<i>K-means</i> : clustering method
Biomorpher	COGA with K-means	<i>COGA</i> : Cluster-Oriented Genetic Algorithms <i>K-means</i> : clustering method
Opossum	MSRSM, Guttman	<i>MSRSM</i> : searches the model for points that balance improving the model's accuracy, using a genetic algorithm/random sampling/mathematical solvers <i>Guttman</i> : evaluates the surrogate model's point of largest curvature
Silvereye	PSO	<i>Particle Swarm Optimization</i> : seek good local minimum of the similarity measure, conjugate gradient used to find local minimum accurately, not suited for multiple object tracking
GOAT	Various nonlinear optimization algorithms	Gradient-free optimization algorithms
Refinery	NSGA-2	<i>Non-dominated sorting genetic algorithm</i> : distribute population of solutions over Pareto-optimal regions

Architectural projects are complex with many functional, contextual, material, economic, code, and client/stakeholder requirements. With the addition of coupled parameter design tasks from other disciplines, the number of variables and objectives will increase [23]. There is an interest in exploring the use of state-of-the-art optimization methods using machine learning approaches. These approaches can be

better suited to search through large search spaces, to ensure convergence to optimal solutions with advanced filtering techniques [24].

## 1.2 Neuroevolution

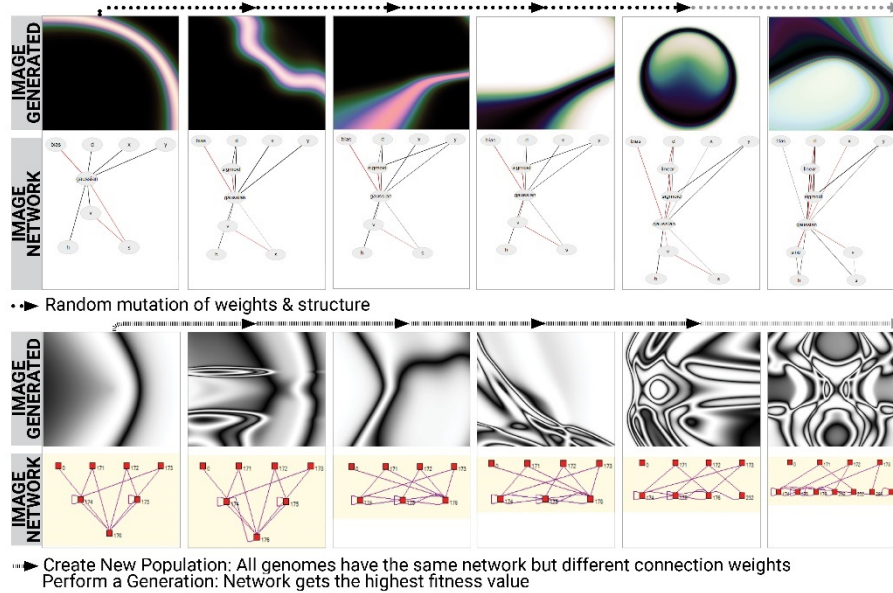
Current research on machine learning methods are focused on deep learning, where neural networks weights are trained through backpropagation or stochastic gradient descent. An alternate approach, neuroevolution (NE), is used to train neural networks with evolutionary algorithms. Research has shown that NE algorithms produce more diverse solutions as it is able to circumvent being stuck in local optima, and scales well with model size and a large amount of data [25] [26]. Another distinction between traditional machine learning approaches to training models and the use of NE for training, is that it evolves an optimal “brain” for the problem instead of mapping. NE approaches address some of the limitations common to many of the algorithms used in existing generative design tools discussed earlier. Particularly in the AEC design context, generative design using NE algorithms is yet to be fully explored.

An alternate approach, neuroevolution (NE), is used to train neural networks with evolutionary algorithms. Research has shown that NE algorithms produce more diverse solutions as it is able to circumvent being stuck in local optima, and scales well with model size and a large amount of data [25] [26]. Another distinction between traditional machine learning approaches to training models and the use of NE for training, is that it evolves an optimal “brain” for the problem instead of mapping. NE approaches address some of the limitations common to many of the algorithms used in existing generative design tools discussed earlier. Particularly in the AEC design context, generative design using NE algorithms is yet to be fully explored.

Neuroevolution algorithms generate solutions that are encodings of the structures that are evaluated for fitness, commonly known as phenotypes, that are typically neural networks. There are two types of encoding that maps a neuroevolution solution to the structure itself: direct or indirect. A direct encoding maps the solution exactly to the structure such that every node and connection is explicitly stated, whereas indirect encoding specifies rules to how the structure should be formed. An indirect encoding can compactly capture regularities such as symmetries in the network structure [25].

In 2007 Stanley [27] proposed a new type of indirect encoding called Compositional Pattern-Producing Networks (CPPNs), which are structurally like neural networks. CPPNs allow the use of a wider variety of activation functions (other than Sigmoid and Gaussian functions used in neural networks) and its input nodes are defined as coordinate system inputs, such as cartesian or polar coordinates [27]. The combination of various activation functions such as sine waves and triangle functions can allow for mathematical abstraction of common structural motifs such as symmetry. These structural motifs can then be used to capture complex patterns. CPPNs can be evolved using any neuroevolutionary algorithms. Neuro-Evolution of Augmenting Topologies (NEAT) demonstrates the possibility for evolution to optimize solutions and become increasingly complex over each generation [28]. NEAT is used to evolve CPPNs (CPPN-NEAT) into architectures of increasing

complexity from a simple form and the results show that patterns and regularities can be discovered with this combination [25].



**Fig. 1.** (A) Evolving 2D images encoded by CPPNs using a weight-space tour where every progression is a random mutation of weights and structure of a single network. All simulations were performed using the CPPNX tool [29]. (B) Evolving 2D images with the NEAT algorithm, every progression both creates a new population of different connection weights, and performs a generation where networks with the highest fitness value survive. Simulations were performed using GeneticArt [30], an implementation of SharpNEAT (used in Aplysia).

Some variants of NEAT and CPPN have been used in several architectural, structural, and urban design projects. An implementation of CPPN and NEAT was used to map the network directly to the physical nodes and vectors of trusses for digital fabrication to evolve efficient truss structures [14]. Richards and Amos [31] used CPPNs to manipulate material properties and shell thicknesses of shell structures, in response to automated finite element analysis and projected design intentions. CPPNs evolved shell textures based on a specific loading case to stiffen the shell structure. Vierlinger [32] used a variation of CPPN-NEAT in Grasshopper, a plugin for Rhino, on agent behaviour based on position and distance to building elements (cores, envelope, slabs, columns), with agents being capable of altering direction and branching behaviour. Janssen [33] used CPPN-NEAT to generate and evolve urban massing models. A developmental procedure was set up to generate variations of urban models, using CPPN to find the four parameter fields: parks, residential/commercial plot ratios, and rotation of blocks. The parameters were mapped to each parcel, and the models were evaluated on predefined performance

criteria, quality of vista and location, with a single overall score assigned to each model generated [33].

## 2 Aplysia

The design of the Aplysia generative design tool is centered on four components: use of a novel algorithm in the generative design space; requirements gathering to understand users' needs; developing a user interface which provides engaging, high-quality user interaction and experience; and an improved generative design workflow. The tool is named "Aplysia", a sea slug/sea hare, inspired by Kandel's Nobel Prize work on the organism's biological mechanisms of storing memory and learning.

The research question was how to develop a tool that provides a user-friendly interface for all skill levels and aid the AEC user to generate emergent design solutions from an unlimited number of parameters, without a tradeoff between computational resources and degrees of freedom. The reduced computation time will not affect the production of scalable, performance-oriented, and buildable geometries. A literature review was conducted on the intersection and application of generative design and neural networks in architecture, structural engineering [15] [18], and urban design, leading to neuroevolution methods [34], and generative tools and algorithms [12] [35] [9] [17] with a focus on Grasshopper and Dynamo plugins. Following the findings of the review, extensive research on CPPN and NEAT was conducted to understand the connection between the algorithms, the problem space, and where the algorithm fits into the optimization process. Ken Stanley, who developed the algorithms, and Daniel Richards, who implemented them for evolving 2D and 3D geometries, were contacted to better understand the algorithms.

### 2.1 Requirements Gathering

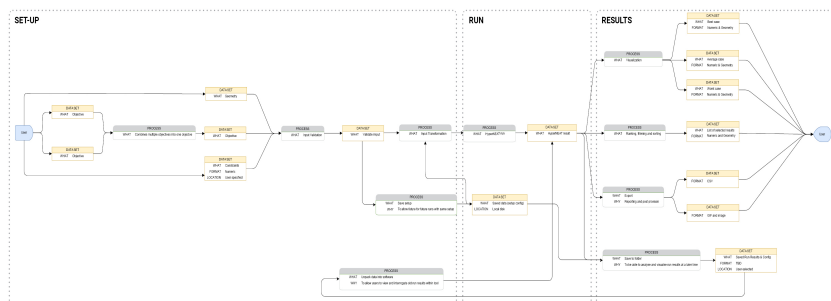
Requirements gathering is a tool used in the development of software projects that can capture multidisciplinary views [36]. Aplysia aims to be an interactive system that can support users in the generative design field in achieving their goals. The scope and requirements of Aplysia was established using user-centered design methodologies in data gathering and analysis for software development [36]. Use case analysis was selected for identifying requirements due to its emphasis on user-system interaction [37] [38], and . It is a method used in software development and is very effective at defining the current and future actions of a product [39]. Semi-structured interviews were conducted for gathering the respondents' data in defining use cases and to gain insight for the requirements. There were 26 questions, ranging from detailing each project worked on using generative design processes, to specific questions about the advantages and disadvantages of each software used. Interviewees were selected from various disciplines (33% architects, 42% structural engineers, 16% digital specialists, 8% urban designers) globally with different levels of expertise in the generative design field within the design/engineering firm.

Data was extracted from the documented interviews based on the type of work, basic or advanced, per discipline. It was further dissected into activities for each user, and preferred user requirements. The user requirements were translated into technical requirements using the MoSCoW prioritization technique to understand and manage core functionality and features, illustrated in table 2. Technical feasibility was considered when prioritizing the user requirements into ‘must-haves’ for a proof of concept to be completed within the allotted time frame.

**Table 2.** Results from the requirements gathering phase

User Requirements		Technical Requirements	
<i>MUST HAVE</i>		<i>SHOULD HAVE</i>	<i>NICE TO HAVE</i>
Low learning barrier	Good software	Detailed	Collect, compare,
Can consider many	UX/UI	visualization of a	and visualize history
objectives	Feed inputs to	specific solution	of runs
Has data visualization	CPPN-NEAT	Grouping of design	Input check and
options	Run CPPN-NEAT	goals	validation
Can sort/filter solutions	Sort, filter, ranking		
Can export solutions to	Export function		
usable formats	Allow design goals		
Can group objectives	and constraints to be		
Display/	turned on and off		
visualization of search	Ability to set design		
error	goals to min or max		

A workflow mapping exercise was conducted for how a user typically interacts with a generative design program. It was then modified to include Aplysia’s user based technical requirements. A user typically begins by setting up the constraints, objectives, etc. needed for the generative design. At this stage, a user may also want to bring up saved settings or load up past data. Once the set-up is done, users will run the algorithm and wait for the results. When the results are ready the users will interact with some visualization of the results and may export them for further analysis outside of the application.



**Fig. 2.** Aplysia workflow mapping diagram.

## 2.2 Distributed System Design

Aplysia is a system modularized such that a failure in one component can be contained. As a result, algorithm failures will not cause UI crashes and run times are independent of users' computer capabilities. Figure 3 illustrates the system architecture comprised of frontend and backend components.

Another notable aspect of a distributed system is the ability to make updates without taking down other parts of the system. Users will be able to run optimizations on their models while the dashboard is down for updates or maintenance and the data will still be stored properly. Access to past data is possible through the dashboard through a web browser even if another component like the engine is going through an update. The Grasshopper Component, seen in Figure 3, which will be detailed in Section 2.3, is the GUI that Aplysia users interact with and it acts as an access point to all the other Aplysia components. Analytical features or algorithm updates can be added without users having to perform any updates to their Grasshopper.

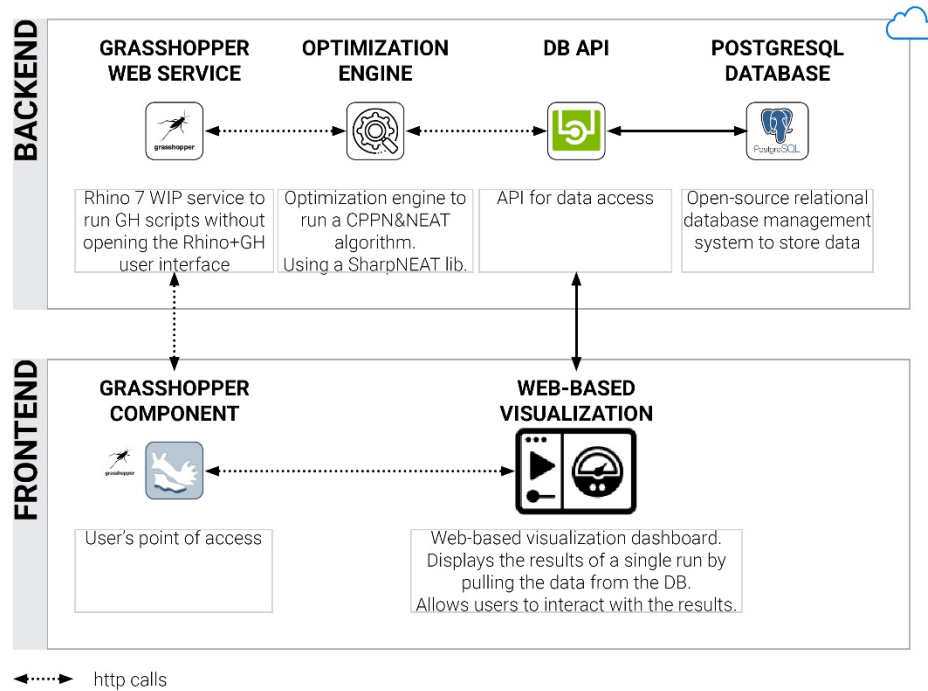


Fig. 3. Aplysia system architecture

### Grasshopper Web Service

This backend component (C#) was created to run specified grasshopper models with sets of constraint values and return the resulting objective values. This is critical to running an optimization algorithm on a grasshopper model. The current implementation uses the Rhino 7 SDK and supported Grasshopper models are limited to analytical tools compatible with this version.



### **Engine**

The “Optimization Engine” (C#) currently contains an implementation of CPPN-NEAT, it can theoretically optimize any model but in this case the fitness evaluation part has been set to call the grasshopper web service to open run a specific grasshopper file with a set of constraint values specified by the optimization algorithm. The engine can be easily updated to improve the CPPN-NEAT algorithm or include other optimization algorithms

### **Database**

A PostgreSQL database was set up to store all optimization run data in a schema designed to allow specific filters on results to be done quickly and easily. To protect against direct access to the database, an API (Node JS) was created as a front for DB access and exposes specific queries that is necessary for the system. Adding new queries in for future features is relatively straightforward and therefore scalable.

### **Dashboard**

A dashboard API (Node JS) was created to visualize the results. The current visualization is limited to a table of all results, graphs of the objective values for all the results, and the ability to visualize details of a single result in a radar plot. Updating the dashboard to include other visualization and analytical tools can be done without having to take down the rest of the system.

### **Aplysia GUI**

The design process for the Aplysia UX/UI as a Grasshopper plugin focused on ease-of-use with minimal button clicks and clear portrayal of information, with a focus on content presentation, easy navigation, simple and responsive interface, consistent UI elements (font, colour scheme), and quick feedback mechanisms. The UI was initially designed in Adobe InDesign and Illustrator, and an interactive mock-up created in Balsamiq. Combining the visuals and functionality of both led to the final proof-of-concept system that was reviewed by an internal UX expert.

### **Set Up**

The first set-up tab shows a summary of the number sliders connected to “Constraints”. The second tab shows the “Objectives”, to be minimized/maximized. Since certain objectives correspond to different disciplines, an additional option for selecting colour categories of each objective is available. The third set-up tab shows options for algorithm settings which are currently under development and will aim to simplify the settings in practical terms, so the user does not need prior knowledge of the algorithm. The final set-up tab displays a summary of the previous tabs for a final check.

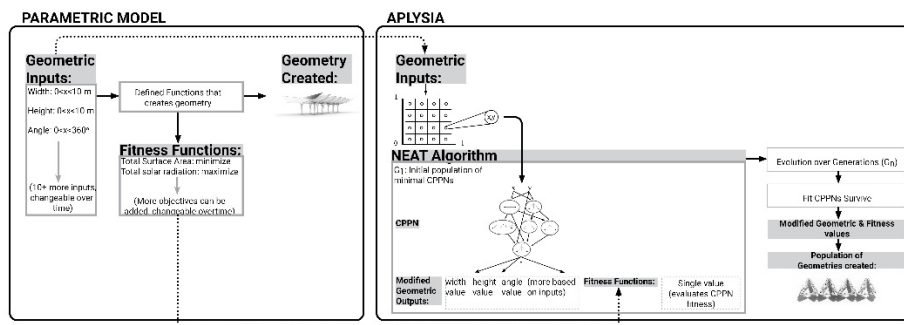
### Run

When the user clicks ‘Run’ once they have confirmed the settings, a message will appear indicating that the backend is running the algorithm. Once the run is complete, a message will show up indicating this along with a link to an external dashboard for the user to view later. The results will also be displayed through the GUI simultaneously.

### Algorithm Implementation

There are two primary categories of topologies that a CPPN can represent: abstract and physical. If a parametric model is represented using abstract CPPN topologies, the geometric characteristics of the model (e.g. the distance between columns in a canopy) will be encoded as independent inputs or nodes, which are not explicitly defined in relation to other geometric parameters. In a physical topology representation, the geometric characteristics of the model are defined relative to each other. These characteristics are encoded in terms of its connectivity with other components in the model (e.g. the position of one column is defined relative to all other elements in the model). This complexity associated with encoding the problem turned the focus onto using CPPN&NEAT with abstract networks, an extension of the original implementation of CPPN-NEAT [27]. By mapping the geometry to an abstract topology, the problem is shifted away from dimensionality and degrees of freedom to the underlying problem structure. Aplysia’s optimization algorithm uses SharpNEAT, an implementation of NEAT written in C# by Colin Green.

Aplysia implemented CPPN&NEAT by abstracting the parametric model with multiple objectives into a simple topological relationship using a “state-space sandwich” substrate configuration [40]. The sandwich is comprised of two layers: one layer sends connections in a single direction to another layer. As the three-dimensional structure is restricted, it becomes a four-dimensional CPPN ( $x_1, y_1, x_2, y_2$ ). For instance ( $x_1, y_1$ ) is on a layer/plane, and ( $x_2, y_2$ ) is on another target layer/plane. This structure allows CPPNs to find patterns within state-space sandwich substrates [40].



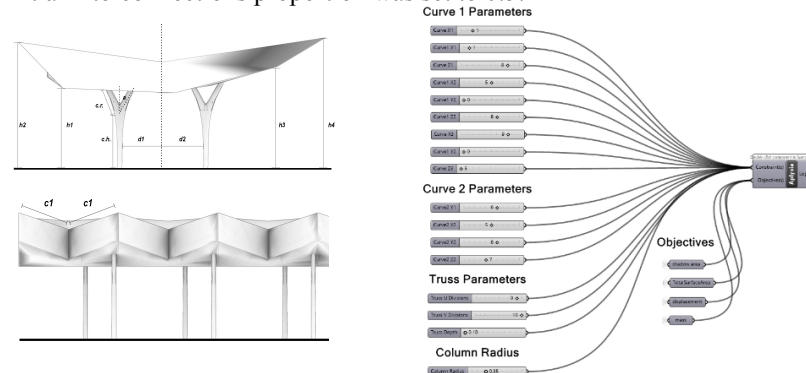
**Fig. 4.** Typical set-up of a parametric model and how CPPN&NEAT works with the model

## Results

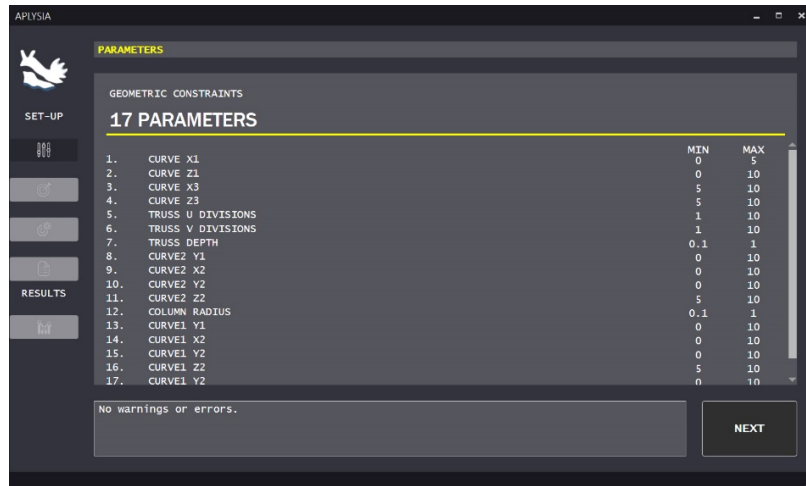
The results tab, shown in Figure 4, displays a table of all the solutions. When a solution is selected, the results for that individual are displayed with radar chart and graphs. Due to the project's time frame, features in the visualization component were not fully implemented but will be included in the next phase of development.

### 2.3 Experimental Results

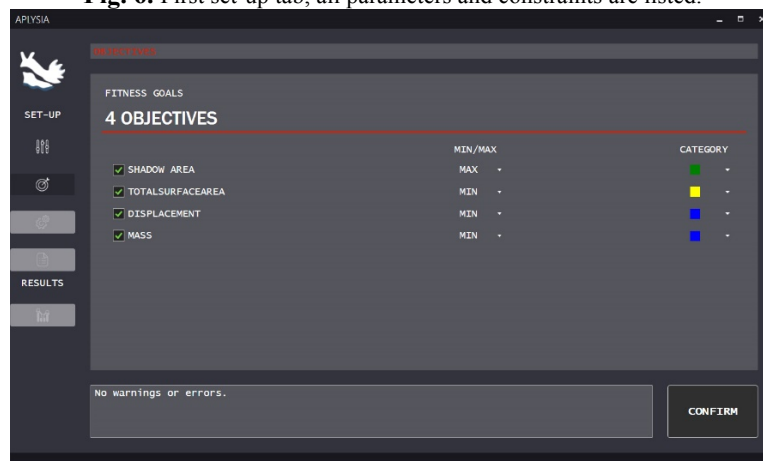
Aplysia was tested on a design problem in the conceptual design phase of a lightweight, free-standing canopy for a transport hub. The problem is defined by 17 geometric parameters. The inputs are the position of the end and mid points for two curves that generate one modular roof; the U, V divisions and depth of the space truss structure; and the column radius. The canopy is modular for constructability and cost effectiveness, where two parallel columns form one module that can be repeated. The Four fitness objectives measure the entire canopy structure (1) to minimize the total surface area of the membrane, (2) maximize shadow area under the canopy, (3) to minimize displacement and (4) minimize self-weight. Solutions were generated over 5 generations, generating 35,000 individuals on an Intel Xeon(R) W-2145 CPU @ 3.70GHz, with a 64.0 GB RAM, the optimum results are shown in Figure 8. For the NEAT algorithm parameters, the default settings were used, where the initial species count was set to 10, elitism and selection proportion was set to 0.2, offspring asexual and sexual proportion was set to 0.5, and interspecies mating proportion was set to 0.01. The initial interconnections proportion was set to 0.5.



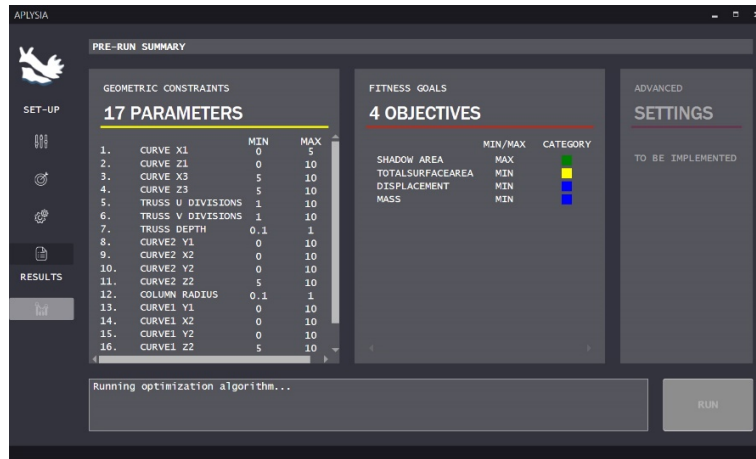
**Fig. 5.** Initial canopy parameters corresponding to the sliders plugged into Aplysia.



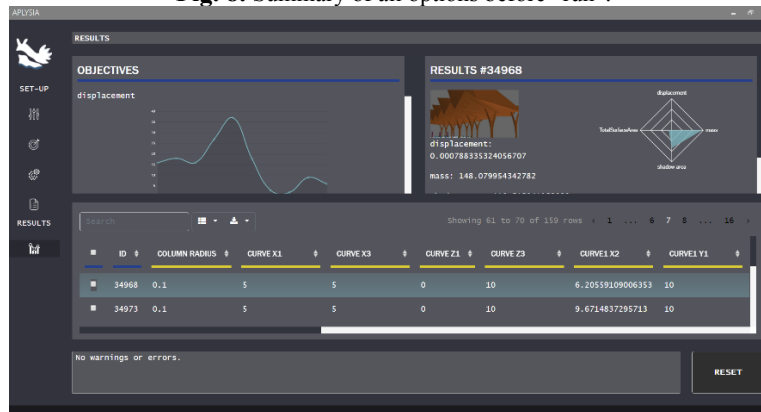
**Fig. 6.** First set-up tab, all parameters and constraints are listed.



**Fig. 7.** Objectives are listed with options to select/deselect objectives; minimize or maximize objectives; colour categories correspond to different disciplines.

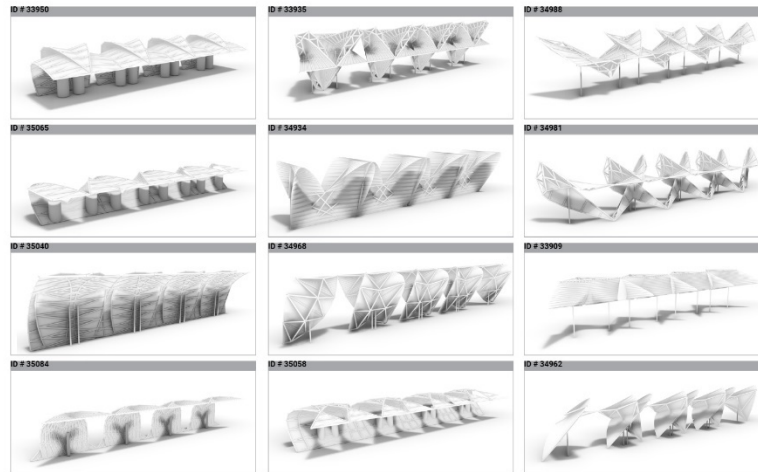


**Fig. 8.** Summary of all options before “run”.

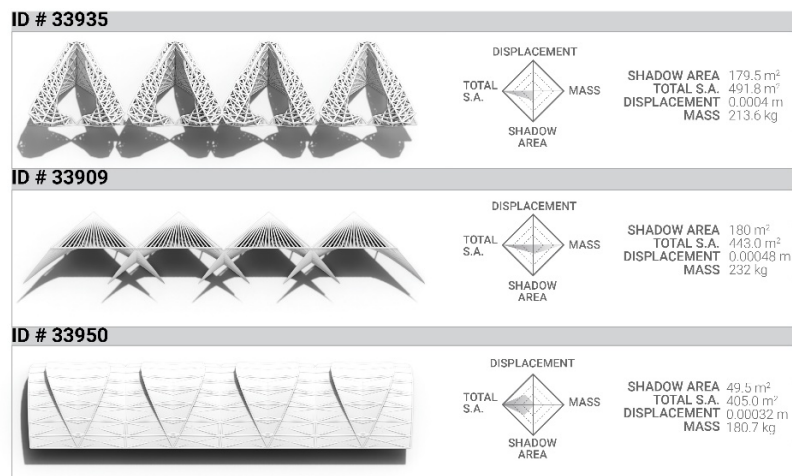


**Fig. 9.** Results tab of the run with some visualization components implemented.

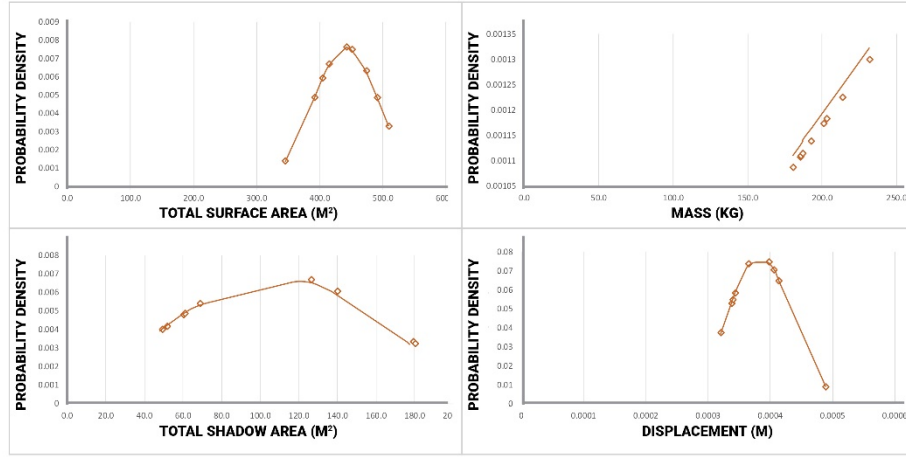
Figure 9 shows the normal distribution graphs for every objective. For minimizing objective (1), the values are more spread out with a larger standard deviation. For maximizing objective (2), the values are more likely to fall within the mean. For objective (3) and (4), the solutions do not have a distribution with a low standard deviation and appears to have converged. In terms of the visual display of each individual, as generations increased, instead of becoming more optimized and geometrically alike, the solutions had varying levels of complexity as displayed in Figure 8, despite that some objectives have converged quantitatively, for example objective (3). These initial experiments indicate that Aplysia has to potential to generate visually complex, and buildable geometries. It demonstrates Aplysia’s ability to create many individuals in a comparably shorter time frame without strain on computational resources, and with user-friendly filtering functions, can be used by a non-expert.



**Fig. 10.** Perspective view of selected solutions in the most evolved run generated from the roof truss Grasshopper model.



**Fig. 11.** Top view of individuals with radar chart.



**Fig. 12.** The normal distribution graphs display how the values of a variable are distributed for four objectives

### 3 Discussion

In the development of generative design tools, Aplysia offers a new opportunity for web-based visualizations and analytics of optimization results within and outside of the Rhino/GH platform. With this framework, a GUI on a different visual programming platform like Dynamo can also be developed in the future and use the same backend infrastructure. The engine aims to make full use of cloud-based parallel computing to decrease run times and allow deployment or updates of algorithms with no impact to users.

#### 3.1 Opportunities for further work

In understanding how NEAT and CPPN works individually and together, Aplysia can be further explored to make full use of CPPN&NEAT's strength in exploiting regularities. Since it would not be limited by an inherent search along the Pareto front of all possible solutions, for example in NSGA-2 or SPEA-2, NEAT can focus more on discovering diverse and novel solutions. Design exploration is likely the most suitable use for this algorithm and can be a future Aplysia feature in addition to optimization, to fully exploit the capabilities of CPPN and advance architectural design.

A formal benchmarking study comparing Aplysia with other commonly used generative design solvers will be conducted for further research, focusing on speed of convergence and reliability of the solutions generated. Backend efficiencies such as profiling optimization will be included to improve the run speed. Currently the Development Team is deploying the tool to the cloud, to remove reliance on a local server. It can then be accessed by the alpha test group, formed by the use case study

interviewees, for further feedback and to identify new pain points for improving the tool.

## 4 Conclusion

Parametric design has allowed the architect and engineer to create forms that expand the designer's imagination and visualization capabilities and, combined with optimization tools, designs can meet functional and performance-based criteria. The emergence of machine learning algorithms into the domain is slowly bridging the gap between optimality and obtaining new types of morphologies.

The work presented here is an initial proof-of-concept for a generative design tool catered towards the AEC industry, that provides the capability to rapidly assess design alternatives that foster diversity not constrained by number of degrees of freedom at the concept stage for architectural, structural, and urban planning design problems. It demonstrates which designs deliver the best value and balances complex requirements to minimize costs downstream of the project. Aplysia has the capability to reduce inefficiencies in existing tools and workflows, diminish the computational time for models with many parameters and goals, minimize the learning curve for generative design tools, and promote remote collaboration by means of visualizing results through a web viewer. Future implementation of the tool will be applied to an increased variety of design problems, such as generating urban blocks from form-based codes, façade modules to increase energy performance, office space planning, and architectural programming with spatial composition requirements in metro stations.

**Acknowledgements.** This research project is supported by the Invest In Arup fund. We thank Nille Juul-Sorensen (Arup) for his tremendous support to the success of the project, and Jared Stock, Arman Ayrapetyan who are currently working on cloud deployment for Aplysia.

## References

- [1] D. Thompson, *Growth and Form*, Cambridge: University Press, 1959.
- [2] D. Nagy, D. Lau, J. Locke, J. Stoddart, L. Villaggi, R. Wang, D. Zhao and D. Benjamin, "Project Discover: An Application of Generative Design for Architectural Space Planning," *SIMAUD '17: Proceedings of the Symposium on Simulation for Architecture and Urban Design*, vol. 7, pp. 1-8, 2017.
- [3] T. Wortmann, "OPOSSUM: Introducing and Evaluating a Model-based Optimization Tool for Grasshopper," *Proceedings of the CAADRIA 17*, no. April, pp. 283 - 292, 2017.
- [4] A. Costa and G. Nannicini, "RBFOpt: an open-source library for black-box



- optimization with costly function evaluations," *Mathematical Programming Computation*, vol. 10, no. 4, pp. 597-629, 2018.
- [5] D. Rutten, "Galapagos: On the Logic and Limitations of Generic Solvers," *Architectural Design*, vol. 83, no. 2, pp. 132-135, 2013.
  - [6] J. Cichocka, W. Browne and E. Rodriguez, "Evolutionary Optimization Processes As Design Tools," in *31th International PLEA Conference ARCHITECTURE IN (R)EVOLUTION*, Bologna, 2015.
  - [7] D. J. Gerber, S.-H. Lin, B. Pan and A. S. Solmaz, "Design Optioneering: Multi-disciplinary Design Optimization through Parameterization, Domain Integration and Automation of a Genetic Algorithm," in *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, Florida, Society for Computer Simulation International, 2012, pp. 1-8.
  - [8] M. Latifi, M. J. Mahdavezhad and D. Diba, "Understanding Genetic Algorithms in Architecture," *The Turkish Online Journal of Design, Art and Communication*, vol. 6, no. AGSE, pp. 1385-1400, 2016.
  - [9] S. Kocabay and S. Alaçam, "Algorithm Driven Design: Comparison of Single-Objective and Multi-Objective Genetic Algorithms in the Context of Housing Design," in *17th Computer-Aided Architectural Design Futures Conference*, 2017.
  - [10] R. Vierlinger, "Multi Objective Design Interface," Vienna, 2013.
  - [11] E. Zitzler, M. Laumanns and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," Zurich, 2001.
  - [12] K. Deb and N. Srinivas, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221-248, 1995.
  - [13] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.
  - [14] D. Richards and M. Amos, "Designing with gradients bio-inspired computation for digital fabrication," in *Association for Computer Aided Design in Architecture*, 2014.
  - [15] H. Salehi and R. Burgueño, "Emerging artificial intelligence methods in structural engineering," *Engineering Structures*, vol. 171, pp. 170-189, 2018.
  - [16] "Wallacei Primer," [Online]. Available: <https://www.wallacei.com/learn>. [Accessed 20 November 2020].
  - [17] J. Harding and C. Brandt-Olsen, "Biomorpher: Interactive evolution for parametric design," *International Journal of Architectural Computing*, vol. 16, no. 2, pp. 144-163, 2018.
  - [18] S. Chatterjee, "Structural Failure Classification for Reinforced Concrete Buildings Using Trained Neural Network based Multi-Objective Genetic Algorithm," *Structural Engineering and Mechanics*, vol. 63, no. 4, pp. 0-000,

2017.

- [19] N. Khean, L. Y. Kim, B. D. J. Martinez, A. Fabbri, N. Gardner and M. Haeusler, "The Introspection of Deep Neural Networks - Towards Illuminating the Black Box," *Proceedings of the 23rd CAADRIA Conference*, vol. 2, pp. pp. 237-246, 2018.
- [20] S. Chaillou, "Space Layouts & GANs: GAN-enabled Floor Plan Generation," *Towards Data Science*, 2020.
- [21] R. Vierlinger, "SVM Learning (oSL)," [Online]. Available: <https://grasshopperdocs.com/components/octopus/sVMLearningoSL.html>. [Accessed 30 November 2020].
- [22] V. Costa, N. Lourenço, J. Correia and P. Machado, "Neuroevolution of Generative Adversarial Networks," in *Deep Neural Evolution*, Singapore, Springer Nature Singapore Pte Ltd, 2020.
- [23] F. Flager, D. J. Gerber and B. Kallman, "Measuring the impact of scale and coupling on solution quality for building design problems," *Design Studies*, pp. 180-199, 2014.
- [24] S. Adriaenssens, P. Block, D. Veenendaal and C. Williams, *Shell Structures for Architecture: Form Finding and Optimization*, London: Routledge, 2014.
- [25] K. O. Stanley, J. Clune, J. Lehman and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24-35, 2019.
- [26] D. Richards and M. Amos, "Evolving Morphologies with CPPN-NEAT and a Dynamic Substrate," in *ALIFE Synthesis and Simulation of Living Systems*, Manhattan, 2014.
- [27] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 131-162, 2007.
- [28] Stanley Kenneth O. and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99-127, 2002.
- [29] F. Andrews, "CPPNX," 2016. [Online]. Available: <https://floybix.github.io/cppnx/>.
- [30] H. Ferstl, "SharpNEAT-based genetic art homepage," 2006. [Online]. Available: <http://oldblog.holgerferstl.de/2006/02/08/GeneticArt.aspx>. [Accessed 30 11 2020].
- [31] D. Richards and M. Amos, "Encoding Multi-Materiality," in *Mixed Matters: A Multi-Material Design Compendium*, Berlin, Jovis, 2016, pp. 40-49.
- [32] R. Vierlinger, "Towards AI Drawing Agents," *Modelling Behaviour*, pp. 357-369, 2015.
- [33] P. Janssen, "Evolutionary Urbanism," in *Computer-Aided Architectural Design Research in Asia*, Hong Kong, 2017.

- [34] P. Koehn, "Combining Genetic Algorithms and Neural Networks: An Encoding Problem," Knoxville, 1994.
- [35] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based many-objective optimization," *Evolutionary Computation*, vol. 19, no. 1, pp. 45-76, 2011.
- [36] S. Lane, P. O'Raghallaigh and D. Sammon, "Requirements gathering: the journey," *Journal of Decision Systems*, pp. 302-312, 2016.
- [37] Y. Rogers, H. Sharp and J. Preece, *Interaction Design: Beyond Human-Computer Interaction*, 3 ed., John Wiley & Sons Ltd, 2011.
- [38] I. F. Alexander and N. Maiden, *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, New York: John Wiley & Sons, 2005.
- [39] A. M. Langer, *Analysis and Design of Next-Generation Software Architectures*, Cham: Springer Nature Switzerland, 2020.
- [40] K. O. Stanley, D. B. D'Ambrosio and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, pp. 185-212, 2009.