MIT Digital Structures

digitalstructures.mit.edu

User Manual Version 1.4

Updated March 9, 2019

Written by Nathan Brown
ncbrown@mit.edu

1.1 Introduction

Design Space Exploration (DSE) is a suite of Grasshopper tools developed by Digital Structures at MIT. These tools aim to support visual, performance-based design space exploration and interactive multi-objective optimization (MOO) for conceptual design. Rather than one single component or user interface, these tools can be used flexibly with other Grasshopper components or plug-ins to take a variety of approaches to DSE and MOO, including *a priori*, *a posteriori,* and *interactive* articulation of performance objective priorities. In addition to the basic DSE functionality, the suite of tools also contains components for surrogate modeling, constrained optimization, design space simplification, and other tasks often encountered in performance-based parametric modeling.

Section 2 describes the **Catalog** set of tools, which can be used to sample and record a performance-based parametric design space. It also includes components that reconstruct sampled design vectors, iterate and record simulation results, and interact with data files. More information about the **Catalog** tools can be found in:

> Brown, N.C., de Oliveira, J.I.F., Ochsendorf, J., & Mueller, C. (2016). Early-Stage Integration of Architectural and Structural Performance in a Parametric Multi-Objective Design Tool. *Proceedings of the 3rd International Conference on Structures and Architecture*, Guimarães, Portugal.

Section 3 describes the **Simplify** set of tools, which help designers organize and quickly focus on areas of the design space that are high-performing. Simplify tools include Cluster, which reduces the number of designs being considered to distinct families; Effects, which quickly indicates how important variables are for performance; Tilde, which builds surrogate models to rapidly approximate objective functions; and Diversity, which helps filter designs and generate diverse design sets for inspection. Other tools are still in development. More information about Simplify tools can be found in:

> Brown N., & Mueller, C. (2017). Designing with data: moving beyond the design space catalog. *Disciplines and Disruption, ACADIA 2017*, Cambridge, MA.
>
> Brown N., & Mueller, C. (2017). Automated performance-based design space simplification for parametric structural design. *Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2017*, Hamburg.
>
> Tseranidis S., Brown N., & Mueller C. (2016). 'Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures'. *Automation in Construction*, 72, pp.279-293.

Section 4 describes the **Optimize** set of tools, which automatically guide designers towards high-performing designs. These tools include MOO, which automatically approximates the Pareto front for a multi-objective problem; Radical, which enables constrained, gradient-based optimization for both numerical and geometric design variables; Stepper, which allows for interactive, gradient-based control of objectives during exploration; and Contort, which helps map original variables to performance-based synthetic variables. More information about Optimize tools and how to use them can be found in:

> Brown N., & Mueller, C. (2018). Gradient-based guidance for controlling performance in early design exploration. *Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2018,* Cambridge, MA.

Brown, N., & Mueller, C. (2019). Design variable analysis and generation for performance-based parametric modeling in architecture. *International Journal of Architectural Computing*, 17(1), pp. 36-52.

Section 5 describes **Stormcloud**, which is a platform for interactive evolutionary exploration in Grasshopper. Stormcloud is based on the browser tool structureFIT. More information about Stormcloud can be found in:

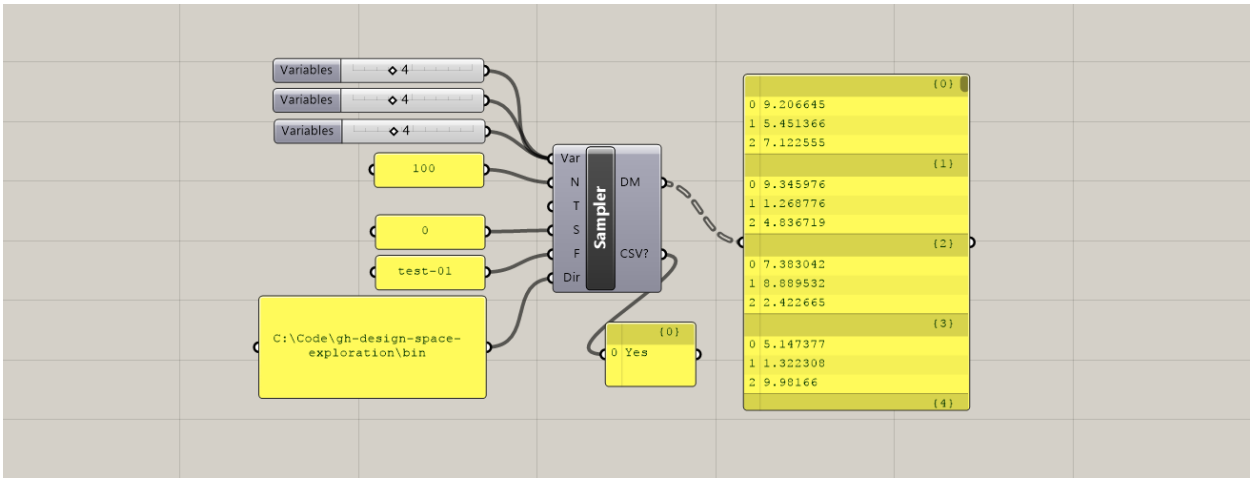
Danhaive, R., & Mueller, C. (2015). Combining parametric modeling and interactive optimization for high-performance and creative structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2015, Amsterdam.

These tools are being developed as part of ongoing research into digital design processes for architects and engineers. The authors welcome any discussion about possible applications, improvements, or new directions.

2.1 Catalog - Component Descriptions

Sampler

This component automatically generates a list of parametric design vectors, called a “design map”, based on user-defined design variable properties. It outputs this design map as a nested list and Grasshopper, which can be used directly or plugged into other DSE tools, and saves it as a .csv file for documentation and interfacing with outside software. Sampler works on double-click, which will both output a Design Map and write a new file to your directory.



Inputs:

Variables (Var) – Takes in any number of sliders that are used as design variables for a given project. Sampler automatically reads the bounds of the sliders to set the limits of the design space being explored.

Number of Samples (N) – The number of samples to be generated. In the ‘grid’ design mode, rather than automatically truncating the list of samples at the edges or adjusting their spacing, the Design Map contains a list of samples that fully covers the design space in each dimension. This results in a number of samples that is higher than the input N, but is divisible by **Vars**. The user can manually trip this Design Map as needed.

Type (Type) – Sets the sampling technique: random, grid, or Latin Hypercube.

Seed (S) – Allows the user to refer back to a previously generated Design Map, even if random as involved. If the seed is set to ‘0’, Random or LHC will generate a new Design Map each time it is clicked. Every other integer will reference back to the same Design Map, even if it is randomly generated.

Filename (F) – The name of the Design Map file that will be written as a .csv. Does not include extension.

Directory (Dir) – The location on your computer where the Design Map will be saved. The directory can end with either a “\” or the folder name.

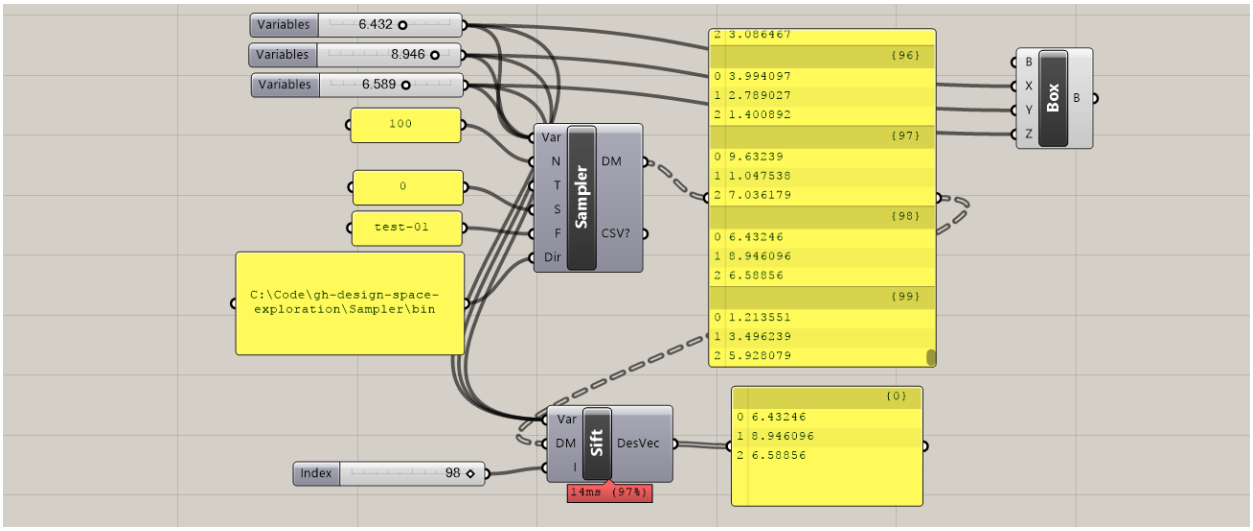
Outputs:

Design Map (DM) – A nested list of design vectors that span the design space being explored, based on the selected sampling technique.

Data (CSV?) – This output indicates whether or not the component wrote a CSV file to the directory. If Sampler is double-clicked without a directory, this output will be null. If it is attempted with an invalid directory, it will not run. If it is run with a valid directory, this output will read "Yes".

Sift

This component allows the user to select a specific design from the map and access the geometry or other properties generated by the script for this design vector. Although the next component (Capture) allows for cycling through all the designs, Sift changes variable settings based on the given index input, which saves the user from setting these variables manually while considering a single design.



Inputs:

Variables (Var) – Takes in the sliders that are used as design variables for a given project. Sift is able to reset these sliders based on the index of the design in the Design Map.

Design Map (DM) – A nested list of design vectors that span the design space being explored, generated by Sampler or read from another file.

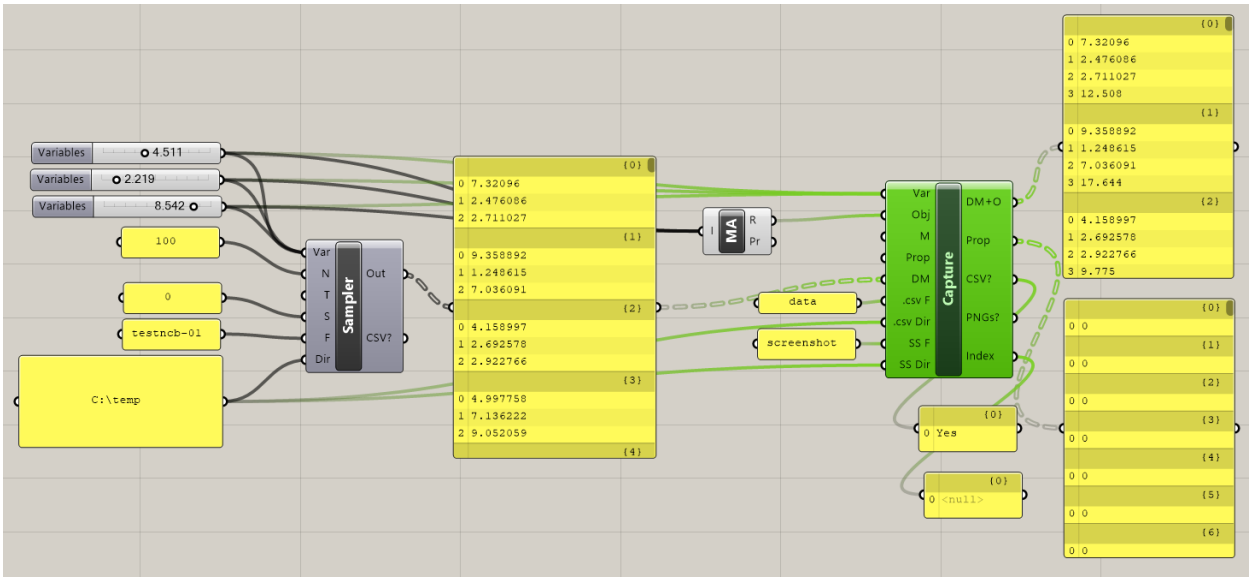
Index (I) – The index of the design under consideration from the Design Map. To activate, navigate to the desired index, double-click on sift, and then click elsewhere on the canvas to view the updated design.

Outputs:

Design Vector (DesVec) – The design vector for the design under consideration.

Capture

This component is a general iterator, which allows the user to automatically generate many different design options and record an image, the performance, and/or other properties of each design. Capture works on double-click, which will cause the component to cycle through each design option and recording the results.



Inputs:

Variables (Var) – Takes in any number of sliders that are used as design variables for a given project. Sampler automatically reads the bounds of the sliders to set the limits of the design space being explored. The sliders should be identical to those used to generate the Design Map.

Objectives (Obj) – Reads in a list of the numerical values from performance evaluations generated by the script. Any performance-based measurement (or objective function), generated from either an outside plug-in or a collection of native Grasshopper components, can be recorded. Depending on Mode, Capture can function with either zero, one, or multiple objective functions in this input.

Mode (M) – Sets the operating mode: screenshots, eval, both, or neither. Screenshot mode records the image of the current Rhino view and saves it as a .png in the directory.

Design Map (DM) – A nested list of design vectors to be evaluated, coming either directly from Sampler or from another previously generated list of design options.

Properties (Prop) – Similar to objectives, reads in a list of numerical secondary design properties to be recorded while Capture is cycling through the different design options. However, these properties are not necessarily objective function evaluations and will be kept separate from the objectives when both are output by Capture.

.CSV Filename (.csv F) – The prefix name of the Design Map + Objectives and Properties files that will be written as a .csv to the directory.

.CSV Directory (.csv Dir) – The location on your computer where the Design Map + Objectives and Properties files will be saved. The directory MUST end with a "\" – otherwise, the file will be written one level higher than the intended directory.

Screenshot Filename (SS F) – The prefix name of the screenshot files that will be saved as a .png in the directory.

Screenshot Directory (SS Dir) – The location on your computer where the screenshots will be saved. The directory MUST end with a "\" – otherwise, the file will be written one level higher than the intended directory.

Outputs:

Design Map + Objectives (DM+O) – A nested list of design vectors and their objective values that span the design space being explored. Standard format for each design is the design vector followed by as many objective values as are recorded.

Properties (Prop) - A nested list of the secondary design properties recorded while Capture is cycling.

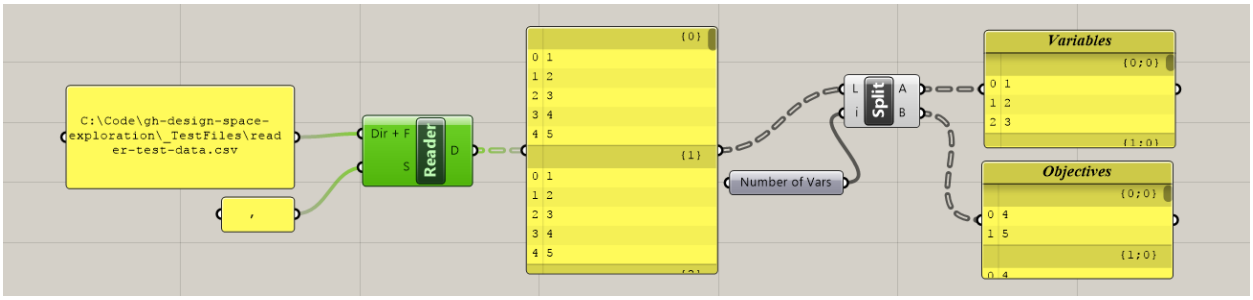
Data (CSV?) – This output indicates whether or not the component wrote a CSV file to the directory. If the component is double-clicked with an invalid directory, it will not run. If it is executed with a valid directory, this output will read "Yes".

Images (PNGs?) – Similar to data, but for provides information about screenshots.

Index – This output exposes the index of each design while capture is running. It is useful for keeping track of Capture's status, or doing any other automated task while iterating through designs. For example, it can be used with a baking component (available through Lunchbox) to bake each design iteration on a new Rhino layer or side by side in Rhino (i.e. connect the index output to "layer name", or connect it to a "move" component that moves each new iteration a certain distance before baking it).

Reader

This component reads in a data file with rows and columns and converts it to a Grasshopper nested list.



Inputs:

Directory + Filename (Dir + F) - Takes in the full address on your computer where the desired file is located, including the extension. The format takes in each row and converts it to a nested list, following the standard format for Design Maps. However, this component could read in any file saved in a similar row/column structure, including .txt files.

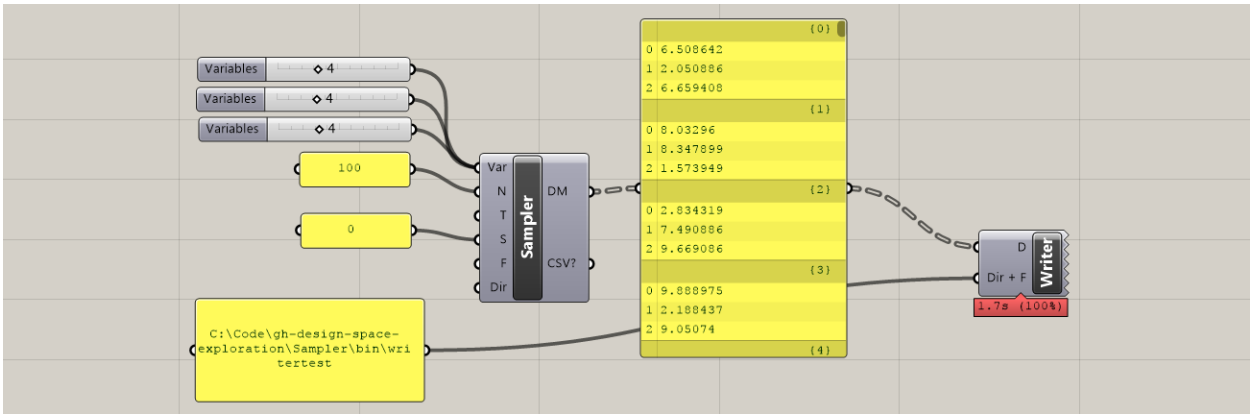
Separator (S) – Sets the character used to separate the data in the .csv. By default it is a space, which is how Design Maps are written, but for other data it could be ",".

Outputs:

Data (D) - A nested list of the data contained in the .csv, following the standard format of design variables followed by objective values in the case of DM+O files.

Writer

This component automatically writes a Grasshopper nested list to a .csv file in the specified directory. It does not need to be doubleclicked, and will write whenever the inputs are refreshed.



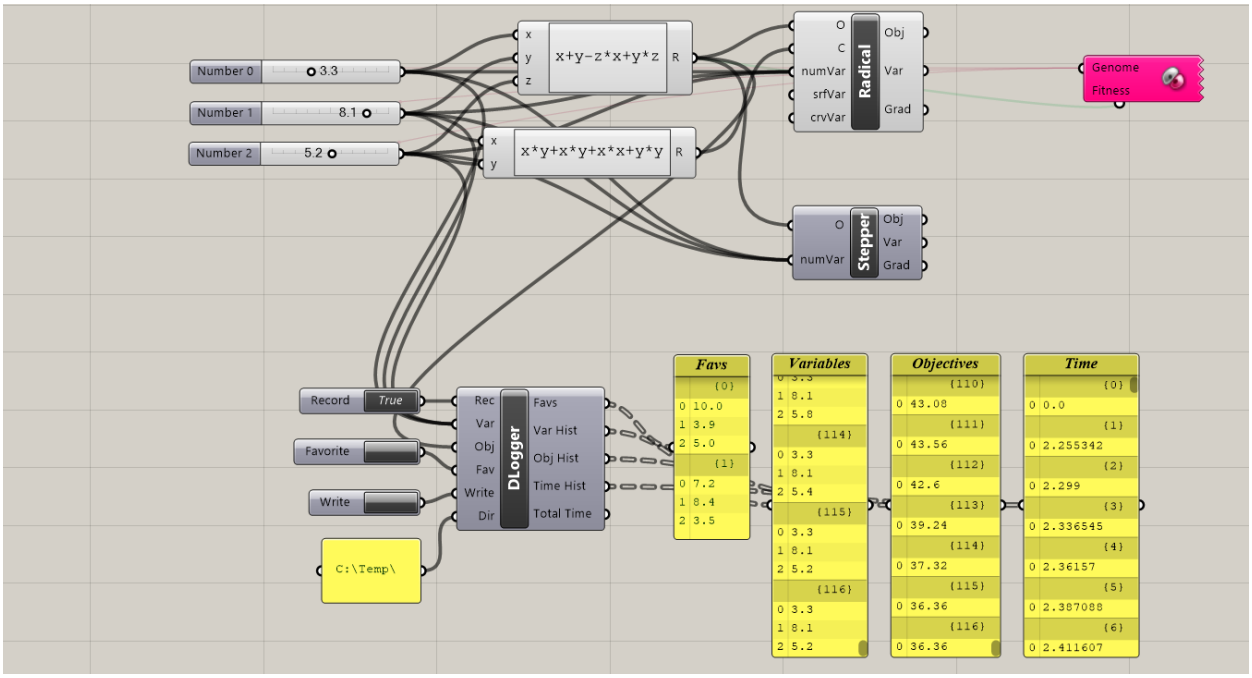
Inputs:

Directory + Filename (Dir + F) - Takes in the full address on your computer where the file should be written (no extension is necessary).

Data (D) – Takes in a nested list of the data that will be converted to a row x column .csv file.

Design Logger

This component records design vectors, objective values, and time during an exploration or optimization run. It also allows users to flag "favorite" designs for revisiting later, and write a file with the history of the exploration. It is useful to record the results of an automated optimization run, or keep track of designs that were considered during a live exploration.



Inputs:

Record (Rec) – Turns recording on and off. Must be set to true while recording. Toggling to false and back to true resets the history.

Variables (Var) – Takes in any number of sliders that are used as design variables

Objectives (Obj) – Takes in any number of objective values to record

Favorite (Fav) – Takes in a button, which can be clicked to record a "favorite" design during exploration, to be revisited later

Write (Write) – When toggled to true, this input writes a file with the design variables, objectives, time, elapsed time, and favorite designation

Directory (Dir) – Location for histories to be written

Outputs:

Favorites (Favs) – List of favorited designs. Selected designs can be reconstructed using Sift, with this history as a design map.

Variable History (Var Hist) – List of recorded variables

Objective History (Obj Hist) – Left of recorded objectives

Time History (Time Hist) – List of elapsed time values, from when the recording was started. To use for timing optimization runs, subtract the second element in this list from the total time, since the second element indicates when the sliders are first changed after setting recording to "true".

Total Time (Total Time) – Cumulative time that the recording has been running.

Note when using with Radical: Radical by default disables unnecessary components on the script, which can considerably speed up an optimization. However, the Design Logger might also be disabled in this situation, and thus only show the first and last designs for an optimization run. Thus, to use Design Logger for recording optimization runs in Radical, users must do one of two things: check "Block Component Disabling" in the Radical options menu, or plug "Total Time" into Radical as a constraint (>0), which will force it to stay active.

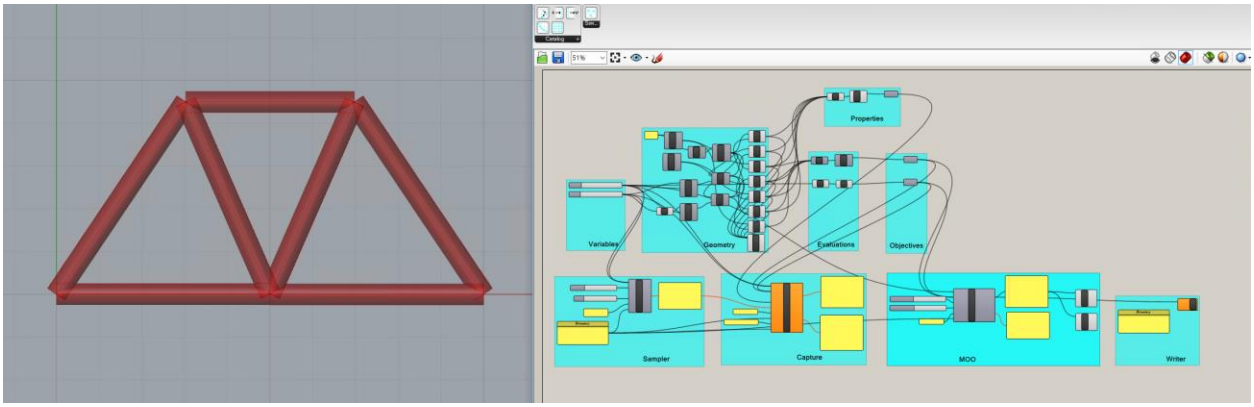
2.2 Catalog - Getting Started Example

7 Bar Truss

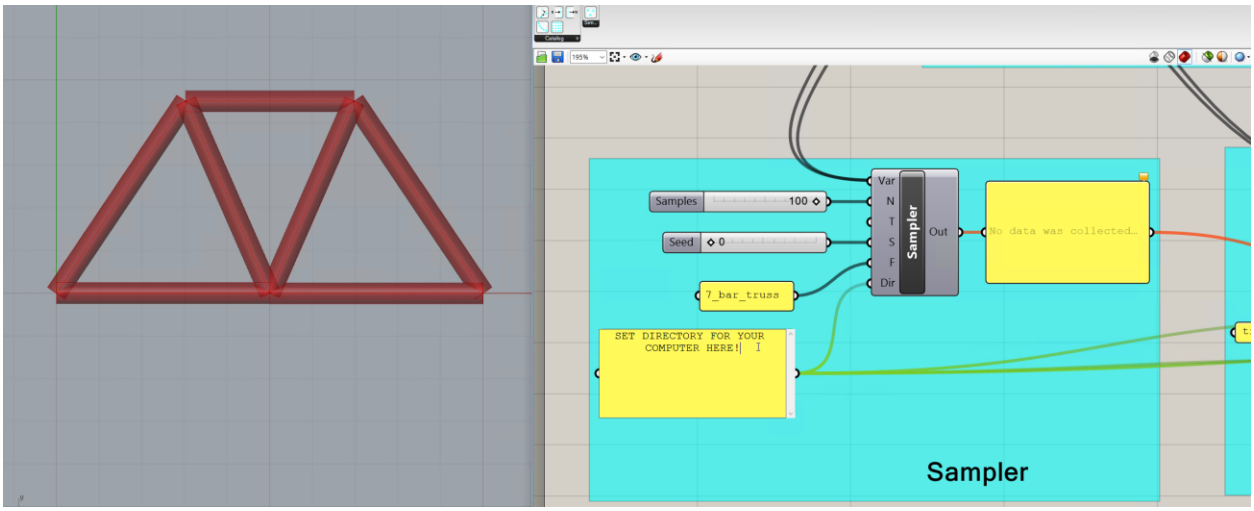
This script demonstrates a simple workflow for how the DSE components can all be used together to explore a design space and run multi-objective optimizations. In addition to the Design Space Exploration components, the script contains a 2-variable, parametric, geometric definition of a 7-bar truss, example objective function evaluations, and example properties. Only the geometry of truss is defaulted to be visible.

The truss is a simple example of a potential design in which the structural form can be explored visually, but also numerically evaluated for performance. In this example, the two objectives being considered are the depth of the truss, which roughly correlates with how much load the truss can support, and the area of the envelope of the truss, which could indicate how much space it takes up in a building. A designer might want the deepest truss possible to support the load, but this objective likely trades off with having a small envelope so as not to be a visually and spatially intrusive element. Since MOO is defaulted to minimize objective functions, the depth of the truss has been made negative. In addition, the script records the total length of all members of the truss, which is treated as a secondary property that the designer would like to track, but does not qualify as an objective. Although the geometry and objective functions used in this example rely on native Grasshopper components and are almost trivial, more complex definitions and plug-ins can be used in their place.

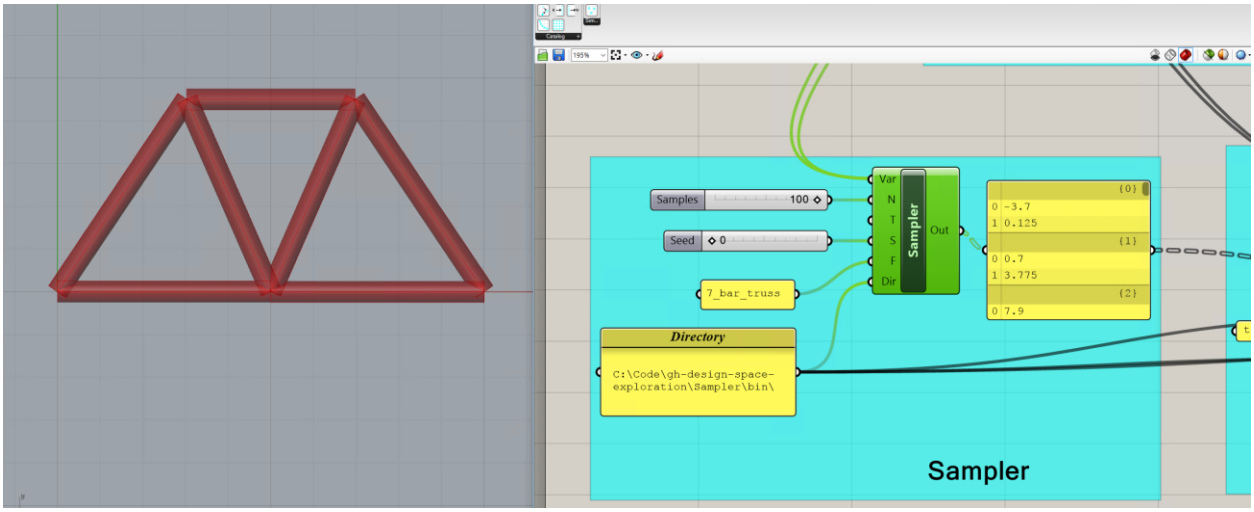
When opening the script, the parametric truss geometry can first be explored live by adjusting the variable sliders directly:



Although free exploration is a powerful parametric method, a designer might want to systematically explore the entire design space, use various optimization tools, and save all results for future decision-making. Each of these tasks can be completed for the provided script using the tools in Design Space Exploration. First, however, the panel containing the “Directory” must be adjusted to the desired location for storing design information. All of the components rely on correct formatting of the directory address in order to work properly:

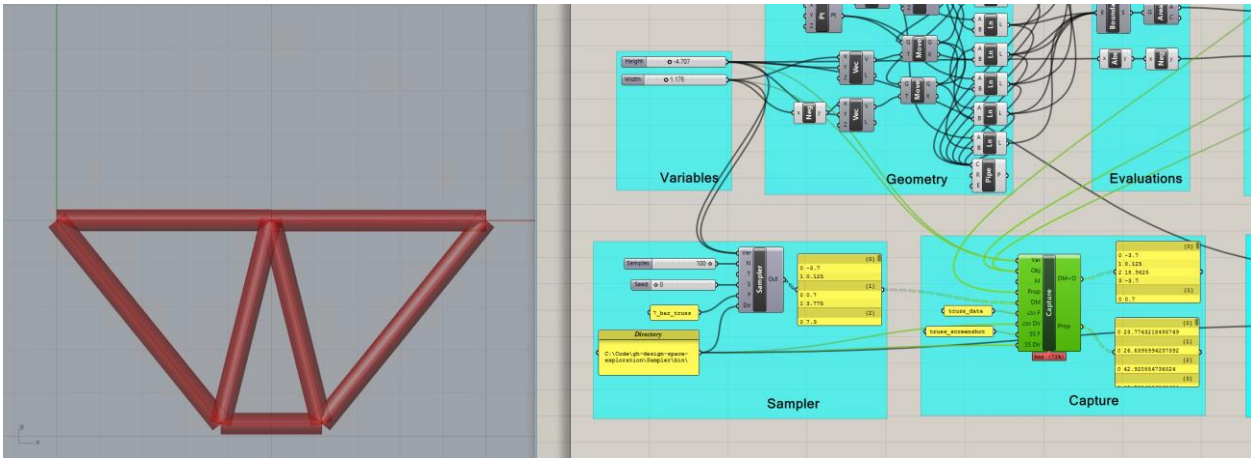


Once the Directory is set, the rest of the components will become functional. To take a representative sample of all potential truss options, doubleclick the **Sampler** component. This creates a Design Map of potential truss variable settings:



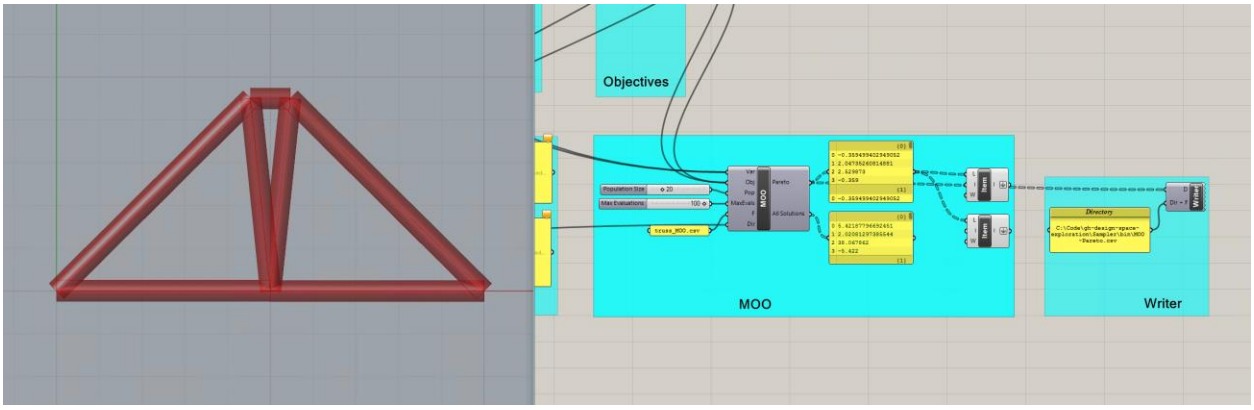
Alternatively, the **Reader** component could be used to read in a Design Map previously saved in a Directory.

After creating a Design Map and feeding it into the **Capture**, it is now possible to cycle through the different design possibilities and record the results. Doubleclicking Capture will start the recording cycle, and when it is finished, the results can be accessed both in the directory and as **Capture** outputs:

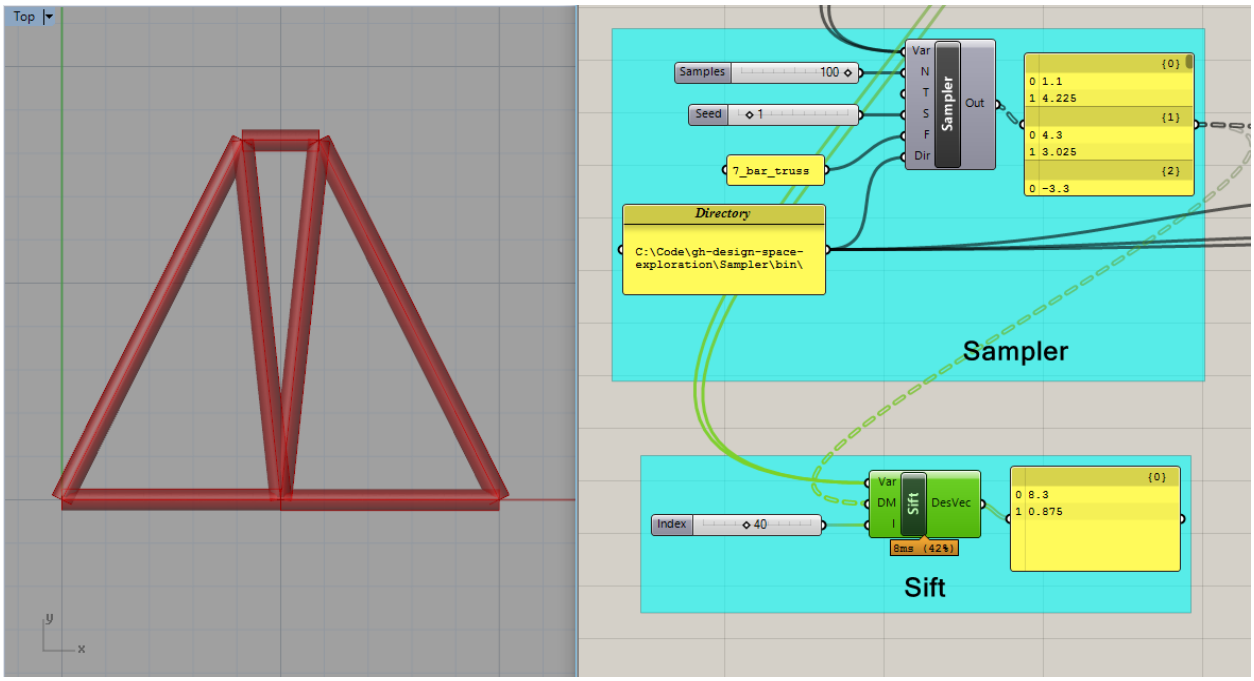


The shot above shows Capture outputting both the new Design Map + Objectives, as well as a corresponding list of design properties.

Instead of taking a representative sample of the entire design space, it may be desirable to automatically find only the best designs, and focus on that area of the design space. Using Galapagos or other plug-ins, it is possible to run single-objective or composite-function optimizations on this 7-bar truss. However, it might also be useful to find the Pareto front between competing objectives, which is done algorithmically by **MOO.** Doubleclicking this component will initiate the algorithm, leading to similar results to those shown below:



If the designer would like to consider only a single design from the sampled design space, this can be achieved using **Sift**. Although resetting the variable bounds is fairly trivial for this problem, it is more labor-intensive for parametric problems with more variables.



2.2 Catalog - General Comments and Known Bugs

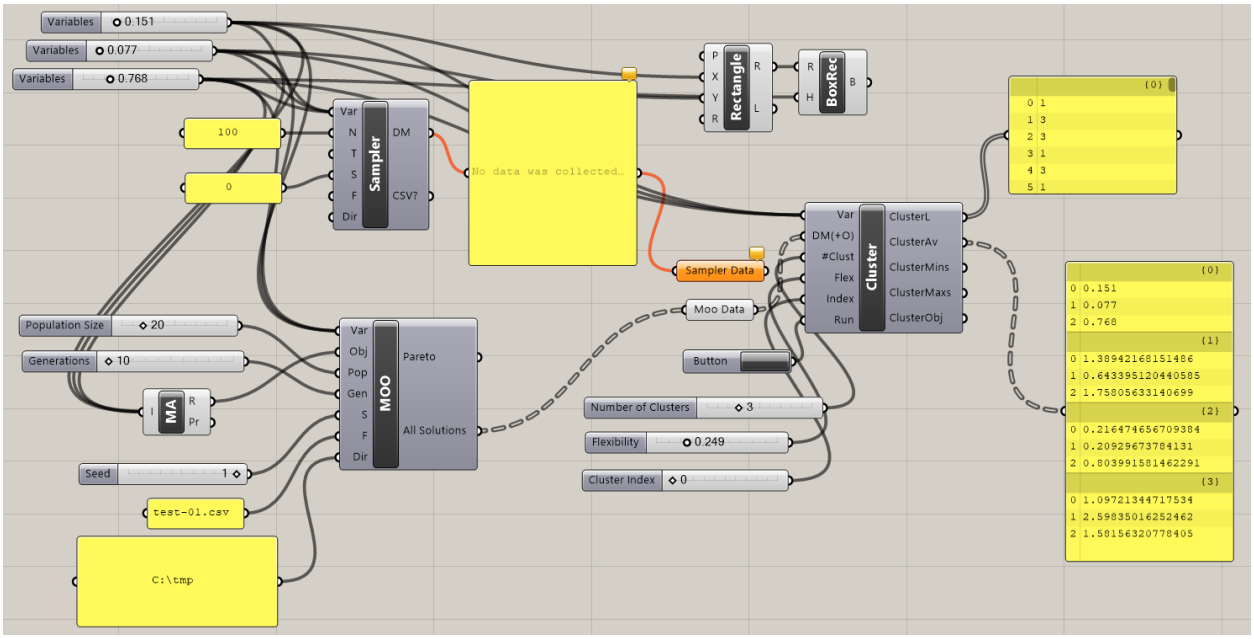
Catalog is a work in progress, and although it has been tested for its main functionality, there may be bugs or quirks about using the software. Here are a few comments that may be helpful for smooth usage:

- Status bars and specific errors have not been added yet, so components will only show that they are working once their action is completed. If a component appears to not be working, or does not respond to a doubleclick, check all of the inputs (especially the directory path, and that inputs are flattened when necessary) to make sure they are properly defined.

3.1 Simplify – Component Descriptions

Cluster

This component executes two separate actions: the first is clustering design samples based on their design variables, and the second is using this information to support cluster-based live exploration. When input "Run" is set to true, the component reads in a map of designs and runs k-means clustering on the dataset, before returning the cluster indices of each tested design as well as the averages, maxes, and mins of each cluster. On subsequent double-clicks, the user can select a cluster and reset the bounds of the problem to correspond to that cluster's properties based on a model flexibility input. As such, a designer can cycle through different clusters organized by performance and explore within the bounds of each cluster, while gaining knowledge about relationship between design decisions and performance. With these two functions, the component can be used for simple data-analysis within Grasshopper or as part of a design space exploration workflow.



Inputs:

Variables (Var) – Takes in any number of sliders that are used as design variables for a given project.

Design Map + Objectives (DM+O) – A nested list of design vectors and their objective values that span the design space being explored. Standard format for each design is the design vector followed by as many objective values as are recorded. Cluster will automatically detect how many variables exist in the problem. The objective values are not used as part of the clustering, and are thus optional. However, if objectives are included, Cluster tabulates the average objective values of the designs within a cluster (as opposed to the performance of an average design in the cluster, which can be determined by navigating to a specific cluster using the "Index" input).

Number of Clusters (#Clust) – Dictates the number of desired clusters.

Flexibility (Flex) – A number between zero and one that determines how flexible the bounds are during cluster-based exploration. If set to one, Cluster will reset the slider bounds to the minimum and maximum values found across an entire cluster. As values get closer to 0, the allowable variable ranges will shrink.

Index (Prop) – After clustering has been completed, this index controls which cluster is being explored. To navigate to a cluster, switch its index and then double-click the component again. This will reset all slider values to the average for the cluster and adjust the bounds based on the flexibility times the difference between max/min and the average. An index of 0 returns to the original bounds and values of the problem. **NOTE:** It is easy to forget to reset the bounds to index 0—make sure this is done every time using cluster if the starting bounds and values are important.

Run – This input executes the clustering algorithm. It should be used with a button.

Outputs:

Cluster List (ClusterL) – Returns the cluster index for each design in the list. Indices start at 1 and go up.

Cluster Averages (ClusterAv) – Returns the average variable settings for each cluster. For this output, the original variable settings are output first at index 0, so that the indices of each cluster average (starting at 1) matches the cluster indices above.

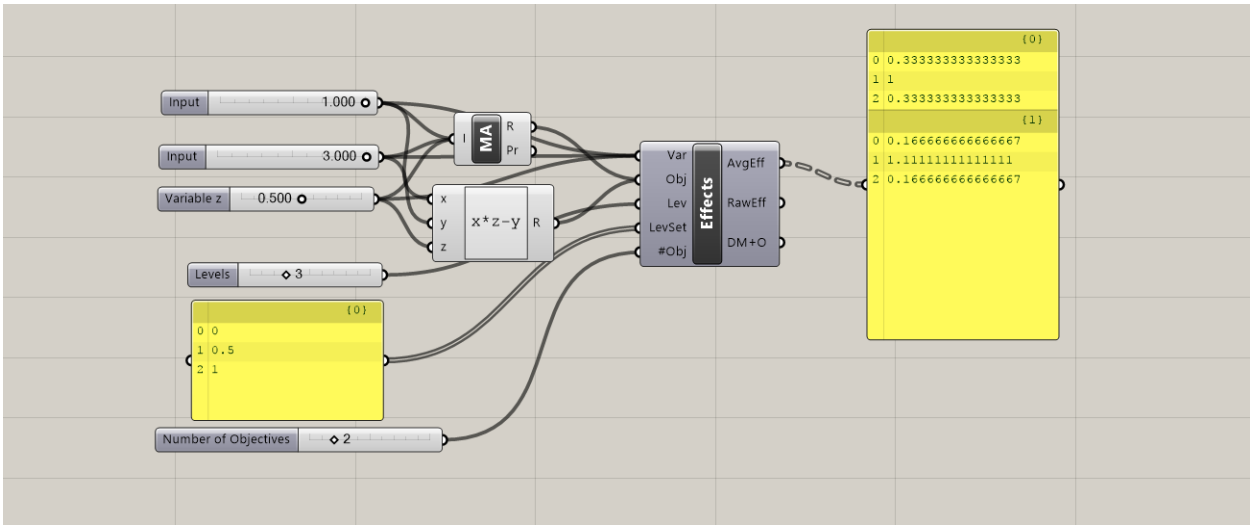
Cluster Maximums (ClusterMaxs) – Returns the maximum variable settings for each cluster. For this and the next output, the first index contains the original settings for each variable, and the cluster properties begin with index 1.

Cluster Minimums (ClusterMins) – Returns the minimum variable settings for each cluster.

Cluster Objectives (ClusterObj) – Returns the average objective values for each cluster.

Effects

Effects gives a quick measurement of variable importance to overall performance by sampling an orthogonal set of design points and attempting to isolate the contribution of each setting. Upon double-click, it reads in variables, objectives, and experiment settings and evaluates a small number of designs before outputting the average and raw effect of each variable. Each "effect" is a number that signifies the effect on the objective value that results from the "cause" of that variable setting. By testing a few different settings and averaging the magnitudes of the effects for a variable, designers can get a sense of what variables have the most influence on the problem. Users can select a 2- or 3- level experiment and corresponding variable settings to be tested. Although these settings are not guaranteed to represent the entire problem overall, Effects can provide valuable information prior to exploration or optimization with only a few objective evaluations.



Inputs:

Variables (Var) – Takes in sliders that are used as design variables for a given project. **NOTE:** Currently limited to 13, because of difficulties in generating orthogonal matrices.

Objectives (Obj) – Reads in a list of the numerical values from performance evaluations generated by the script. Any performance-based measurement (or objective function), generated from either an outside plug-in or a collection of native Grasshopper components, can be used.

Levels (Lev) – The number of levels being considered as part of the experiment. Either 2 or 3.

Level Settings (LevSet) – The (normalized) settings being evaluated during the experiment, in a multi-line list of length either 2 or 3, corresponding to the Levels input. For example, [0.33, 0.66] or [0.25, 0.5, 0.75]. The chosen level settings are extremely important and, depending on the behavior of the objective function, can directly affect the calculation of variable importance. For example, a specific variable setting may correspond to a discontinuity in the design space, and thus would not provide a representative calculation of variable importance for the problem. Although these aspects of the design space are often only discovered

while experimenting with effects, since it is an early design exercise, level settings require thought and intuition on the part of the user.

Number of (#Obj) – Indicates how many objectives should be included in the calculation.

Outputs:

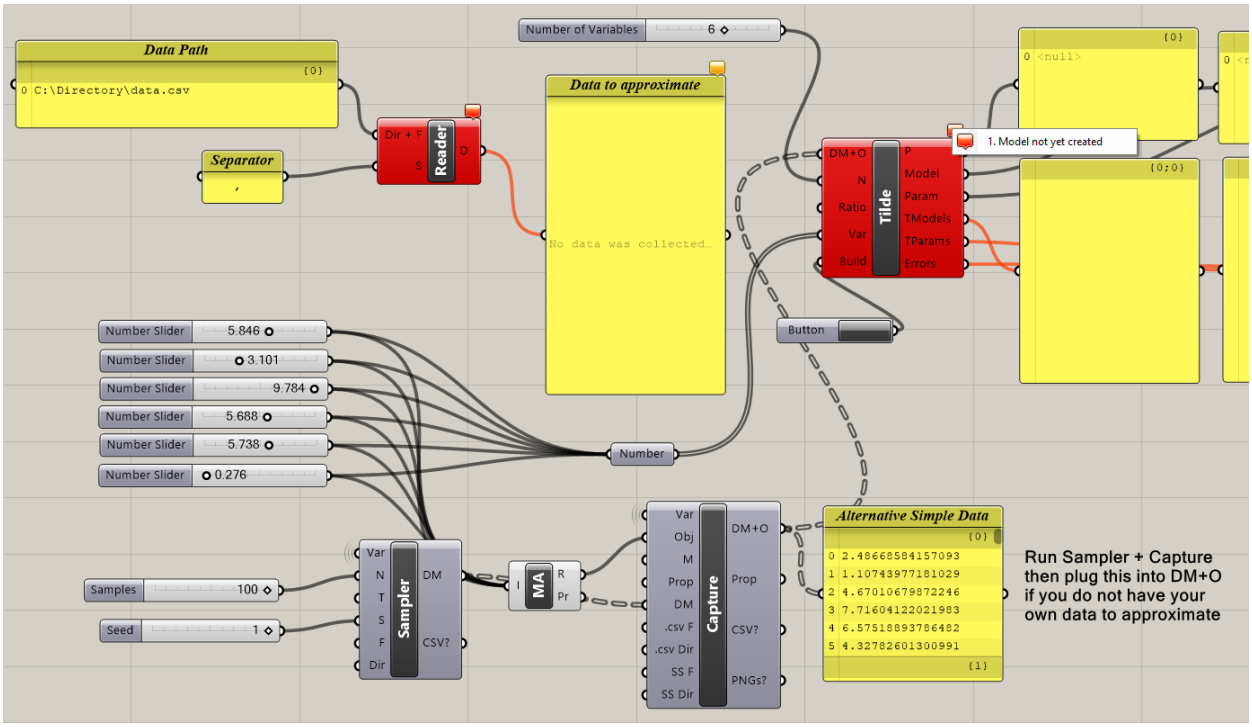
Average Effects (AvgEff) – Outputs the average magnitude of effects for each variable. A relatively large effect is in indication that the variable has a large influence on the performance outcome for the model.

Raw Effects (RawEff) – Outputs the raw effect of each variable setting on each objective. This output shows both the magnitude and direction of each setting, indicating not only which variables are important, but which settings tend to improve performance or make it worse.

Design Map + Objectives (DM+O) – A nested list of design vectors and their objective values that were evaluated during the Effects calculation.

Tilde

This component builds a surrogate model to approximate an objective function using data provided by the user or generated by sampling a design space. Tilde takes in variables and a design map + objectives previously generated for a problem, trains the model when told to "build", and thereafter outputs an estimated value for the objective function based on the current variable settings. Thus, after spending time to produce the original dataset and train the model, users can gain essentially instant feedback about the performance of any design vector within a parametric model. Although there is a tradeoff between speed and accuracy when using approximation techniques, surrogate models have demonstrated enough accuracy for many early-stage design and optimization applications. Tilde works by reading in the DM+O, attempting to fit different Ensemble Neural Network or Random Forest models to the data, validating each model by calculating its errors, and finally choosing the best model for approximation. The component provides information about this training process for transparency, but since it is implemented automatically, users can access powerful models with a simple double-click.



Inputs:

Design Map + Objectives (DM+O) – A nested list of design vectors and their objective values. This must be pre-generated by the user, and can be completed in Grasshopper using Sampler and Capture, the history of a MOO run, or any other dataset.

Number of Variables (N) – The number of variables in the problem.

Ratio (Ratio) – The ratio of training data to validation data for use during the model building process. Tilde needs to split the provided data, and it does so by cutting it into two pieces. By default, the ratio is 0.5,

indicating that half is used for each. **NOTE:** If grid or a related sampling technique is used, it is possible Tilde will train based on one area of the design space and validate based on another, which will not yield good results. For best results, it is important to make sure that the provided data includes points from across the design space in both halves of the DM+O.

Variables (Var) – Takes in the sliders that are used as design variables for a given project. These sliders are only relevant after a model has been trained, at which point the prediction is provided for the current variable settings. The number for variables must match the input for "N".

Build (Build) – When set to "True", this input builds the approximation model. It should be used with a ***button*** to run once. If used with a toggle and that toggle is left "True", a new model will be created every time the canvas updates.

Outputs:

Prediction (P) – Outputs the model prediction based on current variable settings. Predication is only available after the model has been trained, but it will output live predictions when variables without the need for repeated clicks.

Model Type (Model) – The type of surrogate model ultimately chosen by Tilde and used for the problem. Will be either Ensemble Neural Network or Random Forest in the current version.

Model Parameter (Param) – The "nuisance" parameter that resulted in the lowest error model, and thus the parameter that is used in making the current prediction. These parameters are different for the two surrogate model types.

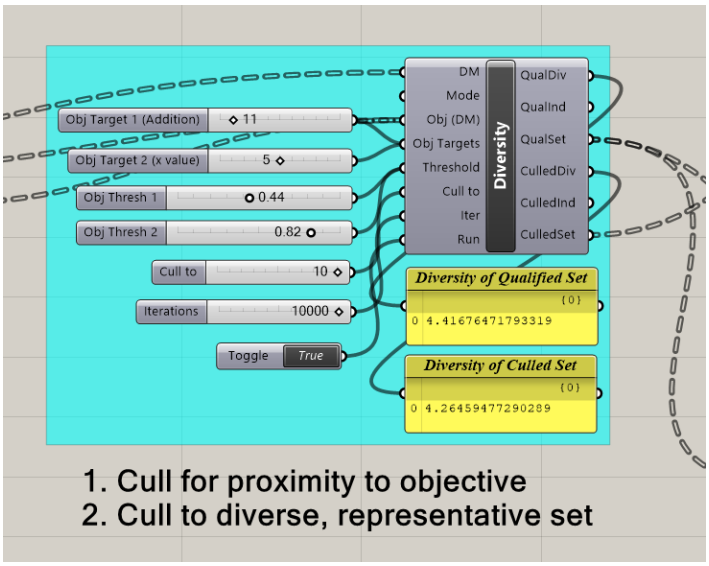
Test Models (TModels) – a ranked list of models attempted, from best to worst (based on Errors).

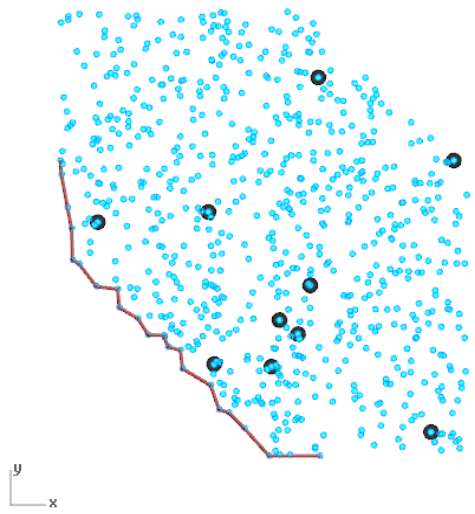
Test Parameters (TParams) – a ranked list of parameters attempted, corresponding to the models above.

Errors (Errors) – an ordered list of RSME for each model attempted, which is how Tilde decides which model to choose.

Diversity

This component helps users filter sampled design space results based on objective values, while also generating a reduced, diverse, representative sample of qualified designs. This process has two steps. First, the user sets target objectives and acceptability limits for removing unwanted solutions. Next, the user provides a number of desired representative samples, at which point the component iteratively searches for a set of solutions with a high measured diversity. Together, these functions help simplify post-simulation analysis, while producing a manageable design set with meaningfully different solutions for visual inspection.





Inputs:

Design Map (DM) – A nested list of design vectors to be evaluated, coming either directly from Sampler or from another previously generated list of design options.

Mode (Mode) – The methodology used to calculate diversity of the sets. Currently, the available methods are sparseness (median), sparseness (average), outlier, or the average of the three. By default, the average is used.

Objective Scores (Obj (DM)) – A nested list of calculated objectives for the design space, often generated using Capture along with various simulations. Rather than other components, in which the Obj input refers to the objective values of the current design, Obj (DM) is a post processing tool that requires the entire objective list.

Objective Targets (Obj Targets) – A list of desired values for each objective.

Objective Threshold (Threshold) – A normalized list of acceptable deviation from the desired target for each objective. For example, an objective threshold of "0.05" means that solutions will be acceptable if they are within 5% of the target. If a single threshold value is given, this number will be used for all objectives—if not, the number of thresholds must match the number of objectives.

Cull to – The desired number of solutions in the final, diverse set.

Iterations (Iter) – The number of attempted iterations when generating a diverse design set. This component repeatedly attempts different combinations of solutions while saving the most diverse set recorded. Thus, a higher number of iteration attempts will generally lead to higher final diversity. While this process is fast enough that many iterations can be conducted almost instantly, the component may slow down for high iteration numbers (>10,000).

Run – This input must be set to "True" in order to iteratively find a diverse solution set. Toggling to False and back to True will reset the process and find a different combination. A Grasshopper toggle is preferred to a button, since the input must be held "True" for the culled outputs to appear.

Outputs:

Diversity of All Qualified (QualDiv) – The measured diversity of the entire qualified set. These metrics are dimensionless as they are a distance within the design space, but can be compared relatively, such as with the diversity of the culled set.

Qualified Indices (QualInd) – The indices of all qualified solutions with respect to the original set.

Qualified Set (QualSet) – The variable settings for each qualified solution, in a nested list.

Diversity of Culled (CulledDiv) – The measured diversity of the culled, representative, diverse set.

Culled Indices (CulledInd) – The indices of culled solutions with respect to the original set.

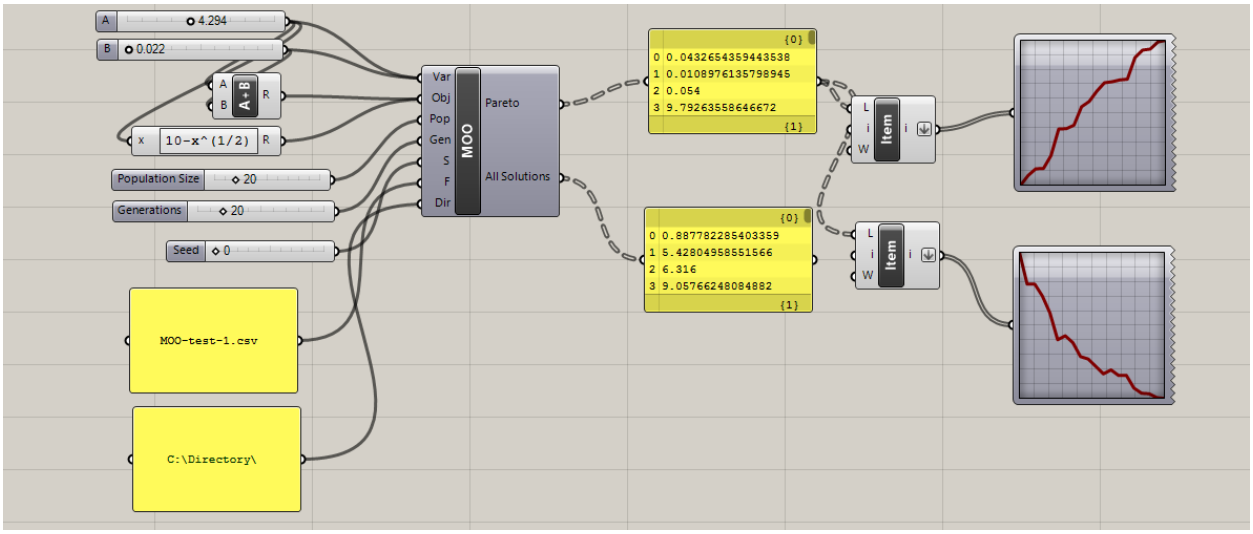
Culled Set (CulledSet) – The variable settings for each culled solution, in a nested list.

4.1 Optimize – Component Descriptions

MOO

This component implements the NSGA-II multi-objective optimization algorithm (Deb et al. 2002), which is a non-dominated sorting genetic algorithm. NSGA-II approximates the Pareto front in a given design problem by using crossover and mutation to iteratively breed successive, higher-performing generations of designs. MOO works on double-click, which will implement the algorithm. When the last generation has been reached, a dialog box will inform the user that the process is finished, at which point MOO will return the results as nested lists and write them to a .csv file in the directory. MOO uses the jMetal Framework (Durillo & Nebro 2011) to implement NSGA-II.

** In order for MOO to work properly, users must also download log4net.dll independently and add it to the Components folder, since it is not distributed with DSE. The .dll file can be found here:*
https://logging.apache.org/log4net/download_log4net.cgi
Under "Binaries", click to download log4net-2.0.8-bin-newkey.zip. Once this zip file has been downloaded and unzipped, "log4net.dll" can be found in (…log4net-2.0.8-bin-newkey.zip\log4net-2.0.8\bin\net\2.0\release)



Inputs:

Variables (Var) – Takes in any number of sliders that are used as design variables for the optimization. MOO automatically reads the bounds of the sliders to set the limits of the design space being explored during the optimization.

Objectives (Obj) – Reads in a list of the numerical values from performance evaluations generated by the script. Any performance-based measurement (or objective function), generated from either an outside plug-in or a collection of native Grasshopper components, can be recorded. Although MOO will still run an optimization with only one objective, it is designed to handle at least two. **Note**: the objectives must be flattened in order for MOO to run properly.

Population (Pop) – Sets the population size for the for the multi-objective optimization algorithm. The total evaluations for the algorithm run will be the population size times the number of generations. **NOTE:** due to the way in which the NSGA-II is implemented, the population size *must* be even, otherwise it will freeze.

Generations (Gen) – Sets the amount of generations to be evaluated during the optimization. Since there is no "threshold" mode for this version of MOO, Gen effectively sets the stopping condition for the optimization—if the results do not satisfactorily represent the Pareto front, more generations should be attempted. If objective function evaluations take a long time to simulate, the duration of one evaluation multiplied by the population size and generations should give a rough indication of the time it takes to run the whole optimization.

Filename (F) – The name of files that will be written to the directory. A successful MOO run will write two files: "LogFile-" + F and "allSolutions" + F. The log file will list details about the optimization for each time the component has been run, but the allSolutions file will reset with the evaluations of only the most recent run. **Note:** outputs may be written with different extensions; as such, ".csv" must be included at the end of the Filename if that is the desired file type.

Directory (Dir) – The location on your computer where the Design Map will be saved. The directory can end with either "\" or the folder name.

Outputs:

Pareto Front (Pareto) – A nested list of design vectors and objective function values of the final generation produced by MOO. These solutions are the approximation of the Pareto front for a given problem. The quality of the Pareto front depends on the each of the inputs, as well as the problem itself, so some experimentation might be helpful.

All Solutions (All Solutions) – A nested list of design vectors and objective function values for all recorded solutions produced while the algorithm is running. The length of this list will match the MaxEvals input.

***Note on Organizing MOO Data*:**

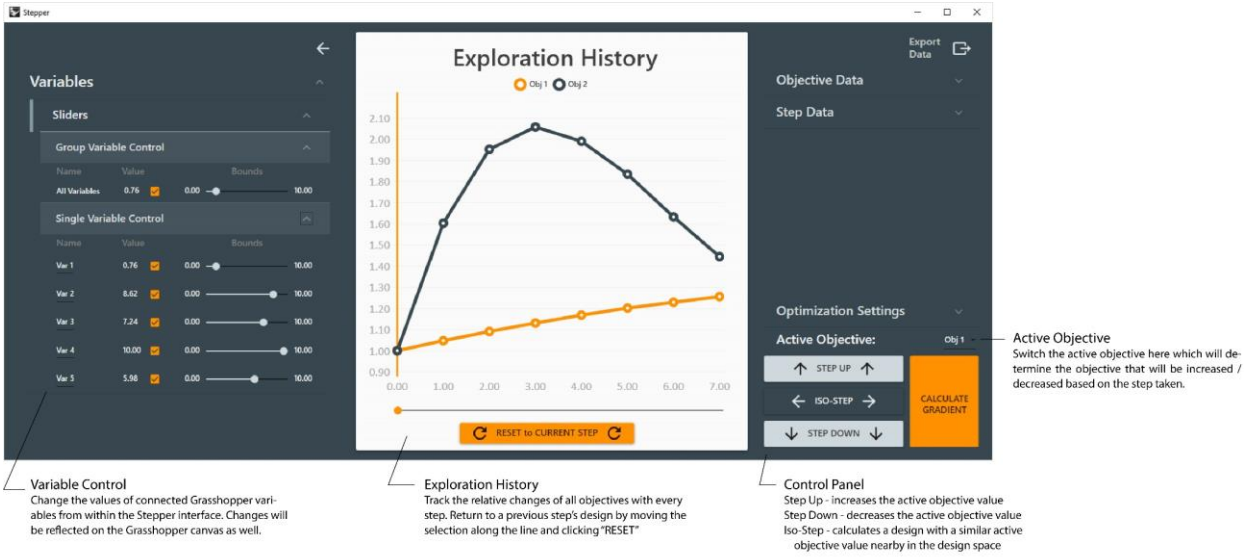
There are a number of useful ways to visualize and interact with MOO data. For example, users can graph the Pareto front, and then use Sift to reconstruct desirable designs along this front. Graphing can be completed using Grasshopper visualization components, Rhino space, or any other preferred software. The outputs of MOO follow the same DM+O format, which includes a nested list of design variables plus simulated objectives added onto the end of each entry. Thus, if only the objective space is being graphed, users should split the data and extract the last items from each design.

Examples of how to manage output data for graphing are given in ***MOO-test-visualize.gh*** (all example files come with the original installation folder).

Stepper

This component is a gradient-based optimization tool that allows users to move in the objective space interactively. This is done through a separate interface from Grasshopper that is accessed by double clicking the component itself. Through the interface, designers can select any objective function and attempt to make it increase or decrease from the current point in the design space. There is also an isoperformance direction which attempts to find a similarly performing design that is nearby in the objective space. The interface includes a history of the changing objective function and options for adjusting the step size and included variables of the problem.

User Interface:

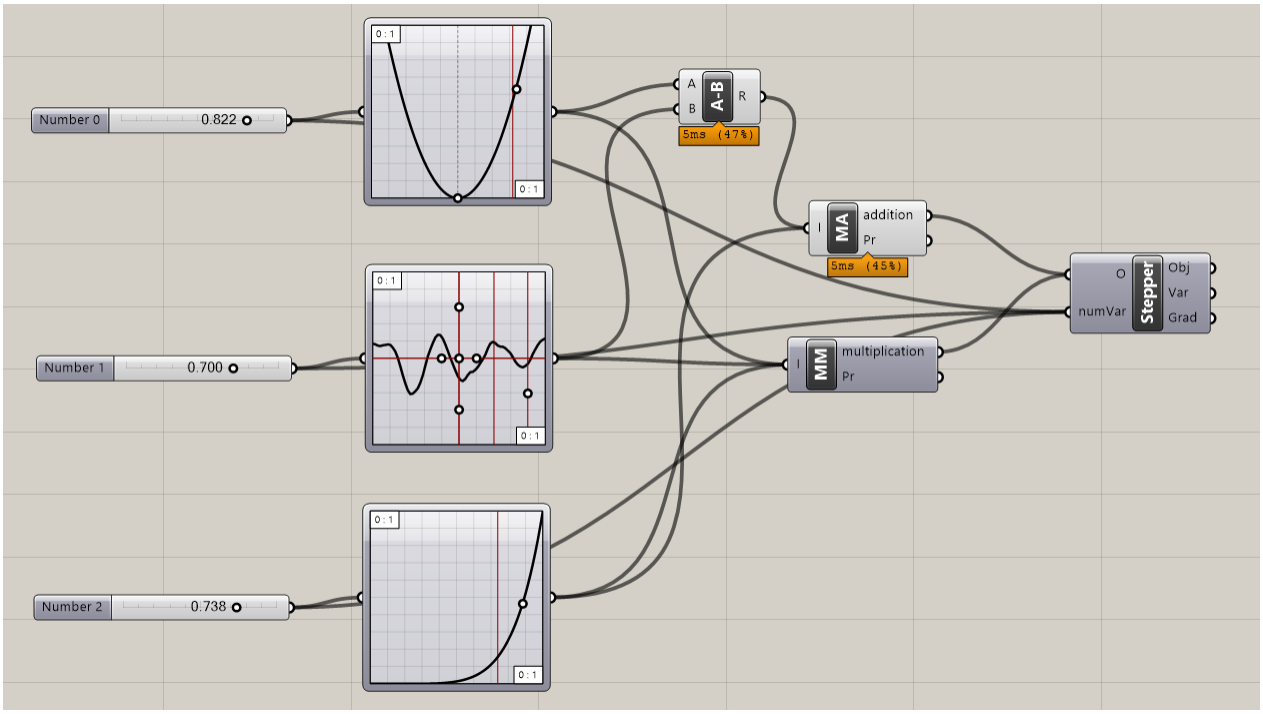


The stepper user interface includes three main panels that allow users to control the variables, view the exploration history and return to previous designs, and manage the settings.

UI features include the ability to:

- Change variable values directly through the interface instead of Grasshopper
- Disable a variable so that its value is not changed during a step
- Reset the system to a previous design found during the exploration
- Change the active objective function
- Calculate the gradient before taking a step
- Use “steps” to navigate the design space based on the active objective value

Stepper Use in Grasshopper



Inputs:

Objectives (O) – Takes in any number of numerical objectives as a list, which can be generated by simulations or other calculations in Grasshopper. Stepper gives the ability to switch between objectives within its interface.

Numerical Optimization Variables (numVar) – Takes in a list of sliders that are used for optimization. The bounds and values of these sliders can be adjusted within the Stepper interface, and they can be made active or inactive.

Outputs:

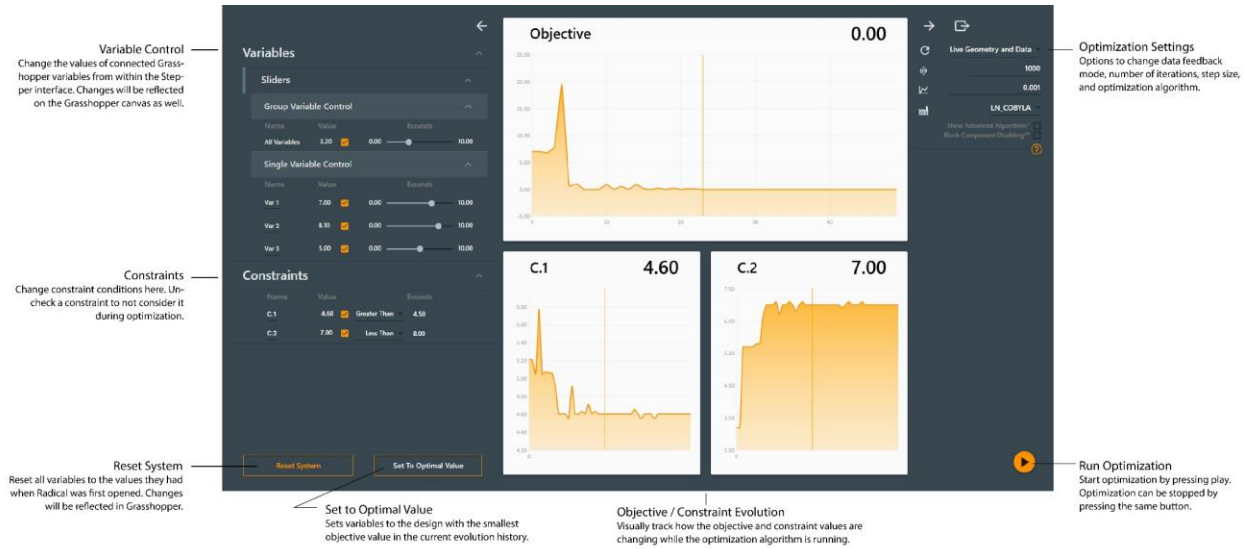
Objective Value History (Obj) – A nested list of each objective step taken during exploration with Stepper.

Variable Value History (Var) – A nested list of each variable step taken during exploration with Stepper.

Radical

Radical is a global, gradient-based constrained optimization solver. The tool contains a separate interface from Grasshopper that can be accessed by double clicking on the component itself. Through the interface users can modify variables and specify the conditions of their constraints before running the chosen optimization algorithm. The interface displays the history of the objective and constraint values and allows for the system to be reset to the original inputs.

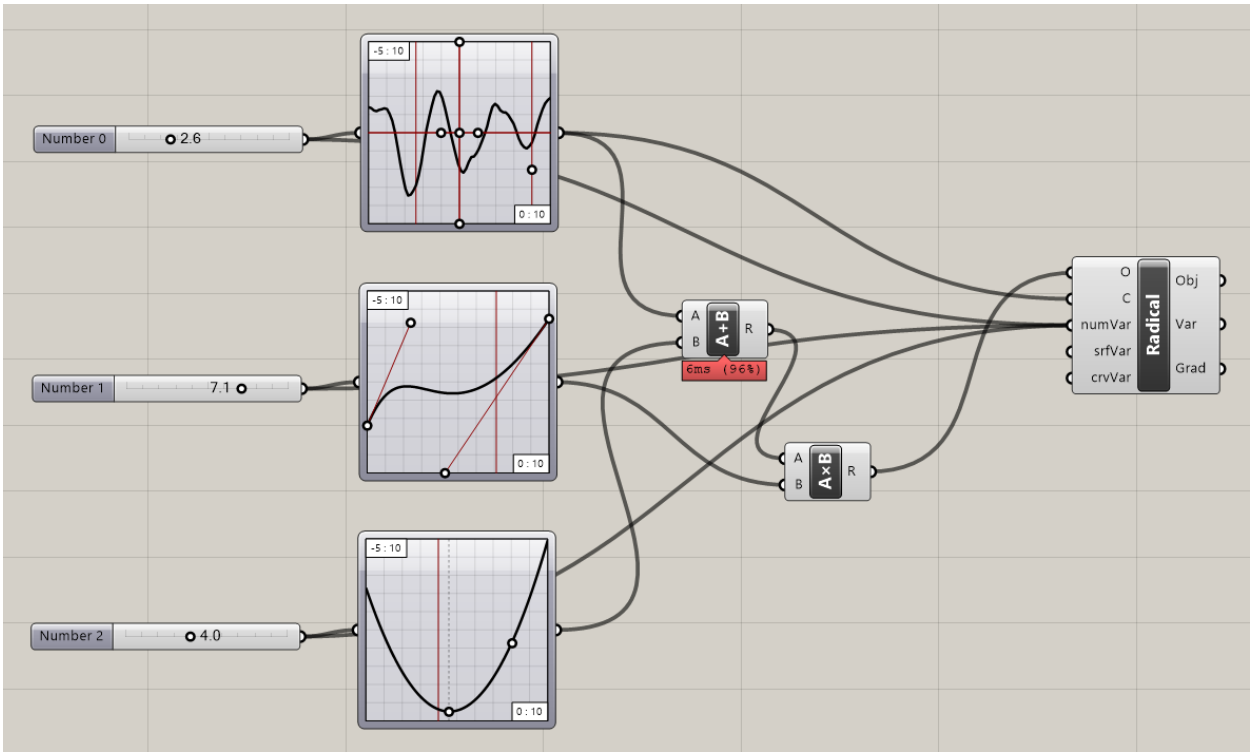
Radical User Interface:



The Radical user interface is composed of three panels that control input variable values, optimization evolution graphs, and optimization settings.

UI features include the ability to:

- Change variable values directly through the interface instead of Grasshopper
- Remove a variable from being considered / altered during optimization by deselecting it
- Specify constraint conditions
- Reset the system to the original values
- Set the system to the lowest value found through the optimization process
- Change the feedback mode
- Change the number of iterations per optimization run
- Change the step size
- Choose the optimization algorithm
- Stop the optimization process while it is running
- Disable unnecessary components during optimization, which can speed up performance (if this functionality is interfering with a script, it can be removed by checking “Block Component Disabling”)



Inputs:

Objectives (O) – Takes in any number of numerical objectives as a list, which can be generated by simulations or other calculations in Grasshopper. Stepper is multi-objective, while Radical only works for a single-objective problem. If multiple objectives are added, the tool will remind you that only Stepper can be used in this situation.

Constraints (C) – Reads in a list of numerical values that can act as constraints while using Radical. After a number is plugged into the constraint input, the user can use the Radical interface to assign whether the constraint values should be greater than, equal to, or less than a provided bound. The presence of constraints will limit which optimization algorithms can be used in Radical. Stepper cannot work directly with constraints—users must manage them manually during exploration.

Numerical Optimization Variables (numVar) – Takes in a list of sliders that are used for optimization. The bounds and values of these sliders can be adjusted within the DSOpt interface, and they can be made active or inactive.

Surface Optimization Variables (srfVar) – Takes in surfaces directly from Grasshopper, at which point DSOpt extracts the U and V parameters of the surfaces and turns them into variables. When working with surfaces, the user can set X,Y, and Z values and bounds for the movement of each control point. Control points can be toggled on (part of the optimization) or off (remaining static) either individually, or by direction (as in, turning all of the “X” variables off). When conducting the optimization, the upstream connection from the surface to other Grasshopper components is broken, so that the final result will be stored in the original Surface container.

Curve Optimization Variables (crvVar) – Similar to the Surface optimization variables, but for curves.

Outputs:

Objective Value History (Obj) – A nested list of each objective step taken during exploration with Stepper.

Variable Value History (Var) – A nested list of each variable step taken during exploration with Stepper.

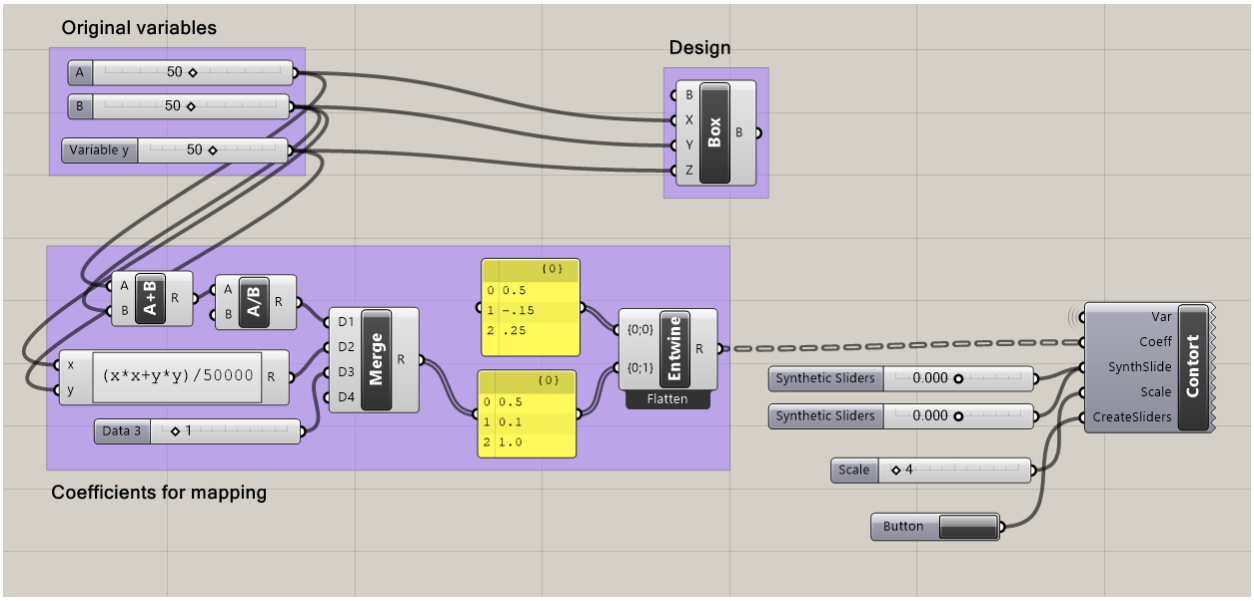
Radical Current Issues

The current version of Radical still include a few bugs and problems. The major ones are listed below:

- Secondary algorithms are not fully incorporated yet. Using an algorithm that requires a secondary algorithm will not run an optimization.

Contort

This component allows users to control a parametric design using synthetic sliders that are mapped back to the original sliders, without needing to unplug/adjust the original variables. These new synthetic sliders "drive" the originals backwards through the script, such that when a synthetic slider is moved, the geometry and everything else downstream of the original variables also updates. These synthetic sliders can be used for live exploration or for more efficient optimization. Various data science and dimensionality reduction techniques can be used to find mapping coefficients for the new sliders.



Inputs:

Variables (Var) – Takes in any number of sliders that are used as original design variables

Coefficients (Coeff) – A set of mapping coefficients for new synthetic variables as a nested list. Coefficients can be static or dynamic. There must be a matching number of coefficients and original variables, although there can be multiple synthetic sliders at a time. For example, if there are three original variables, the coefficients must contain a list of the form:

{0:0} (N = 3) ← Synthetic Variable 1
{0:1} (N = 3) ← Synthetic Variable 2
{0:2} (N = 3) ← Synthetic Variable 3
…

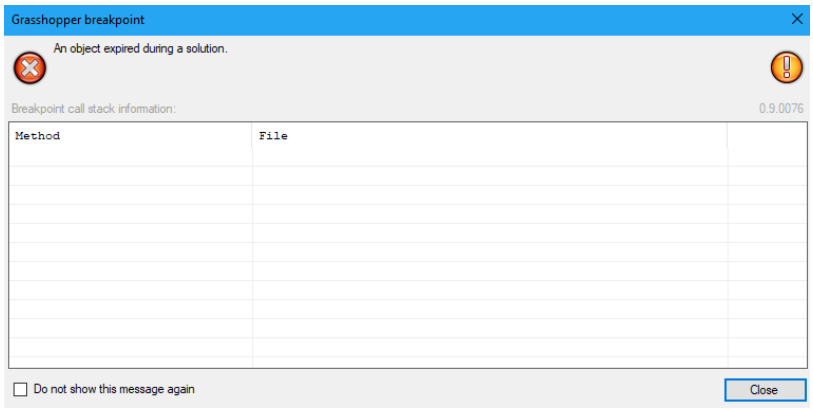
Synthetic Sliders (SynthSlide) – A set of synthetic sliders, ranging from -1 to 1. These can be automatically generated using the "Create Sliders" input for a provided set of coefficients.

Scale (Scale) – The scale of the new synthetic variable, which controls the overall design space movement. For example, a scale setting of 1 and a coefficient of 1.0 will result in an original slider hitting its lower bound when the synthetic slider is set to -1, and its upper bound when the synthetic slider is set to 1.

Create Sliders (CreateSliders) – When a button for this input is activated, it will generate the appropriate number of synthetic sliders based on the provided coefficients. Synthetic sliders can also be input manually, but should contain a range of -1 to 1 in order to work properly.

Contort Known Issues

Grasshopper is not built to have sliders controlled backwards in this manner. Thus, when using Contort and moving a slider for the first time, the following message will likely appear:



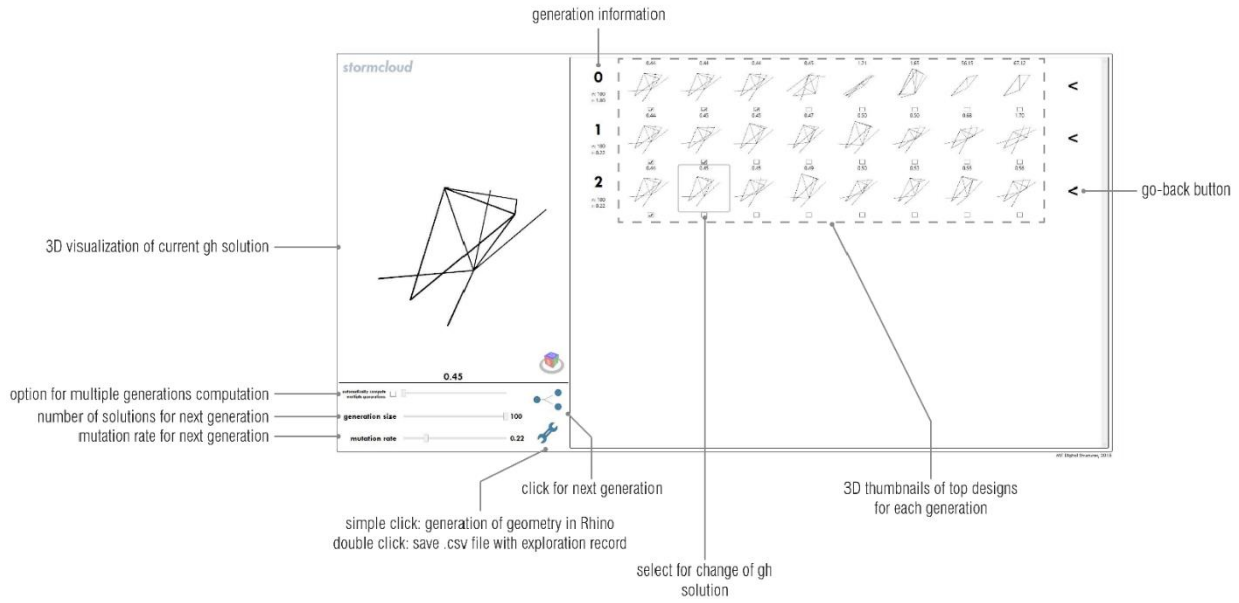
When this box appears, user should click “Do not show this message again”. Given this reality, various issues involving expiring objects may arise when using Contort. However, it can save time when rapidly attempting new directions for design space exploration.

5.1 Stormcloud Description

The tool is implemented as a single component placed on the Grasshopper canvas. Double clicking on the component opens the user interface of stormcloud. The component takes three different inputs – geometry (in the form of curves or breps), score, and design variables - and has no output parameter. Data should be flattened when lists are used for the geometry. The DVar input should be connected to the sliders that will be changed during the exploration. The score is automatically normalized by the component according to the initial solution score.

5.2 Stormcloud User Interface

The user interface is divided in three main parts: the main viewport, which visualizes the current Grasshopper solution, the design grid where best performing designs of each generation are visualized and parents for next generation can be selected, and the exploration control panel with miscellaneous buttons and sliders. The main features of the user interface are summarized in below:



UI features include:

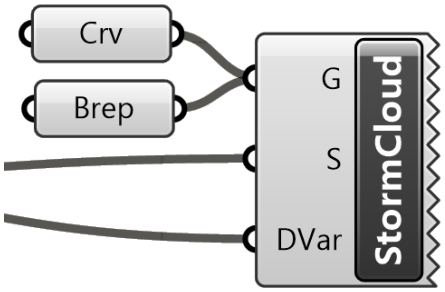
- Navigation in 3D viewports
- Viewports are synchronized for facilitated visualization. The viewports can be navigated using the mouse right-click for rotating, the mouse wheel for zooming, and maj + mouse right-click for panning.
- Selection of design for detailed visualization
- Each candidate solution displayed on the design grid can be selected for detailed visualization on the main viewport by clicking on its corresponding viewport. This also changes the solution state in Grasshopper to correspond with the selected solution.
- Generation of geometry in Rhino viewport
- By clicking on the wrench icon, the user can save the preferred solutions as geometries stored in

Rhino. This feature is equivalent to the 'Bake' feature existing in Grasshopper but improves it by making it more accessible through a simple button click. Each solution is assigned a different sublayer of a common 'exploration' layer.

- By right-clicking on the wrench icon, the designer is offered the possibility to record his exploration by saving the characteristics of each solution explored, i.e. the values of the design variables and the score, as a comma-separated values (.csv) text file on the user's desktop.

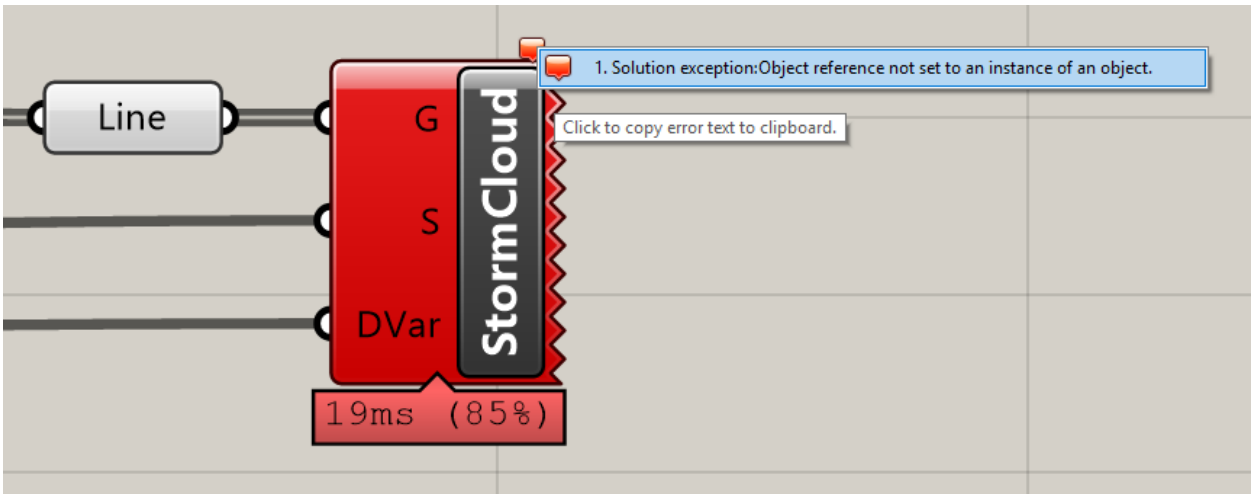
5.3 Stormcloud Step-by-Step Use

Place the IEO component on the Grasshopper canvas. The component can be found in the category "DSE".



Connect the input parameters of the component to the lines representing the design geometry (data must be flattened), the design performance score (as a number), and the design variables as sliders. The exploration will be performed around the initial value of the sliders. Hence, the user may want to reset the sliders to their middle values.

***Inputs must be curves or breps – if lines are input, for example, the component will look like this:**



Double-click the IEO component to open the stormcloud user interface.

Maximize the window. This is currently needed as most of the user interface dimensions are currently hard-coded (=fixed) and starting the exploration when the window is not maximized will result in cropped viewports.

The main viewport might seem empty. However, it may because the camera does not point towards the geometry and needs to be moved around -an automatic move-target-to-objects feature will be added in the near future. Unzooming usually does the trick.

Change the evolutionary parameters to tune the exploration as wished.

Click on the generation button. Depending on the Grasshopper script, the population generation may take more or less time. It is recommended that the user disables the non-vital and slow parts of the script.

When a generation of designs is computed, the 8 top performers will be displayed in 3d viewports on their corresponding row. Hovering over one design will highlight the borders of its viewports with a grey border. Clicking on the viewport will change the Grasshopper solution to the clicked design and will update the main viewport of the user interface. This allows for a detailed visualization of the clicked design.

The user can proceed as before to explore new designs. Parents for next generations can be selected by clicking the checkboxes of the last row of designs.

Recording features can be used to save designs and exploration information.

5.4 Stormcloud Bugs

The current version of stormcloud still includes a few bugs and problems. The major ones are listed below:

- Double-clicking on the component can open multiple windows. The user should make sure that only one window is open per Grasshopper solution as stormcloud windows modify the current solution state which has consequences on other existing stormcloud windows
- Deleting the IEO component does not close its corresponding window.
- The IEO component should be deleted and reloaded for each new exploration as closing the stormcloud window does not reconstruct it and the score normalization will remain based on the solution state corresponding the component construction.
- If the exploration is recorded in a .csv file, and if the corresponding file –with the same full path- already exists and is open, Rhino/GH/Excel will crash.

6.1 Acknowledgements

The following people contributed to the tools in DSE as developers, researchers, or in other ways:

Caitlin Mueller, Nathan Brown, Renaud Danhaive, Jonathas Felipe, Stavros Tseranidis, Anthony McHugh, Alicia Nimrick, Violetta Jusiega

A few concepts from James Ramsden were helpful during development, and can be found here:

http://james-ramsden.com/category/grasshopper/

7.1 Licenses

7.1 References

Brown N., & Mueller, C. (2017). Designing with data: moving beyond the design space catalog. *Disciplines and Disruption, ACADIA 2017*, Cambridge, MA.

Brown, N.C., de Oliveira, J.I.F., Ochsendorf, J., & Mueller, C. (2016). Early-Stage Integration of Architectural and Structural Performance in a Parametric Multi-Objective Design Tool. Proceedings of the 3rd International Conference on Structures and Architecture, Guimarães, Portugal.

Brown N., & Mueller, C. (2017). Automated performance-based design space simplification for parametric structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2017, Hamburg.

Brown, N., & Mueller, C. (2019). Design variable analysis and generation for performance-based parametric modeling in architecture. *International Journal of Architectural Computing*, 17(1), pp. 36-52.

Brown, N. & Mueller, C. (2019). Quantifying diversity in parametric design: a comparison of possible metrics. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 33(1) , pp. 40–53.

Danhaive, R., & Mueller, C. (2015). Combining parametric modeling and interactive optimization for high-performance and creative structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2015.

Deb, K. et al., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), pp.182–197.

Durillo, J.J. & Nebro, A.J., 2011. jMetal: a Java framework for multi-objective optimization. *Advances in Engineering Software*, 42, pp.760–771.

Tseranidis S., Brown N., & Mueller C. (2016). 'Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures'. *Automation in Construction*, 72, pp.279-293.