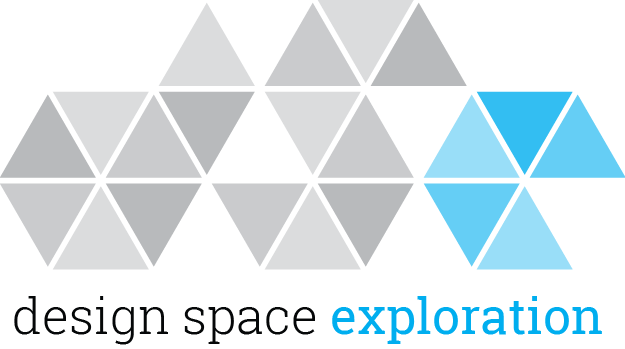
**MIT Digital Structures**

**digitalstructures.mit.edu**

****

**User Manual Version 1.0**

**Updated February 8, 2017**

**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 poseriori,* and *interactive* articulation of performance objective priorities.

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 a component that approximates the Pareto front of a multi-objective problem. 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 **Stormcloud**, which is a platform for interactive evolutionary exploration in Grasshopper.

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.

ADD LINK

MENTION STRUCTUREFIT

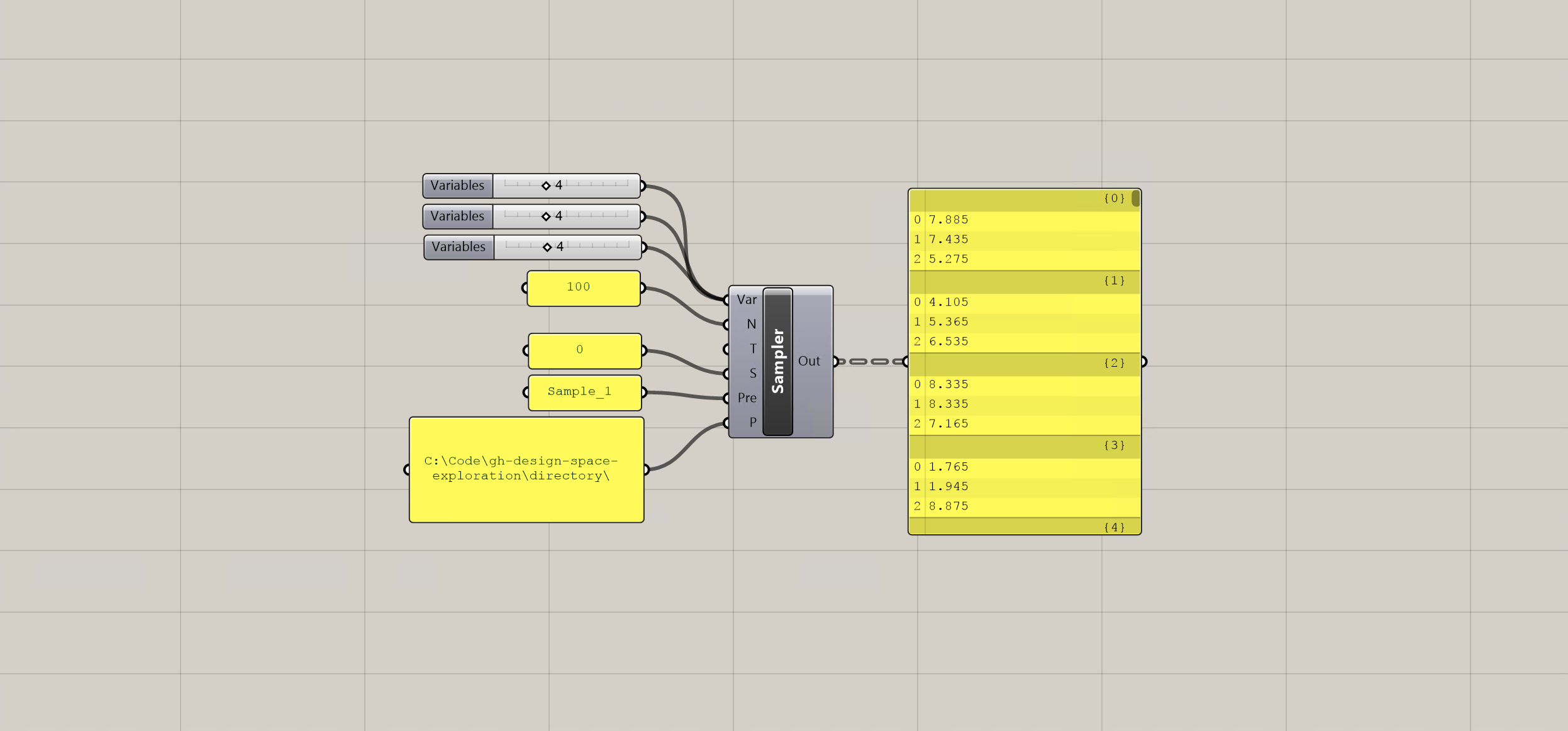
Section 4describes the **Simplify** set of tools, which help designers focus more quickly on areas of the design space that are high-performing. Simplify tools include Tilde, a tool for building surrogate models to rapidly approximate objective functions; Cluster, which reduces the number of designs being considered to distinct families; and others.

*(Section 4 tools have not been released yet, and are coming soon)*

**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(citation).

**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.

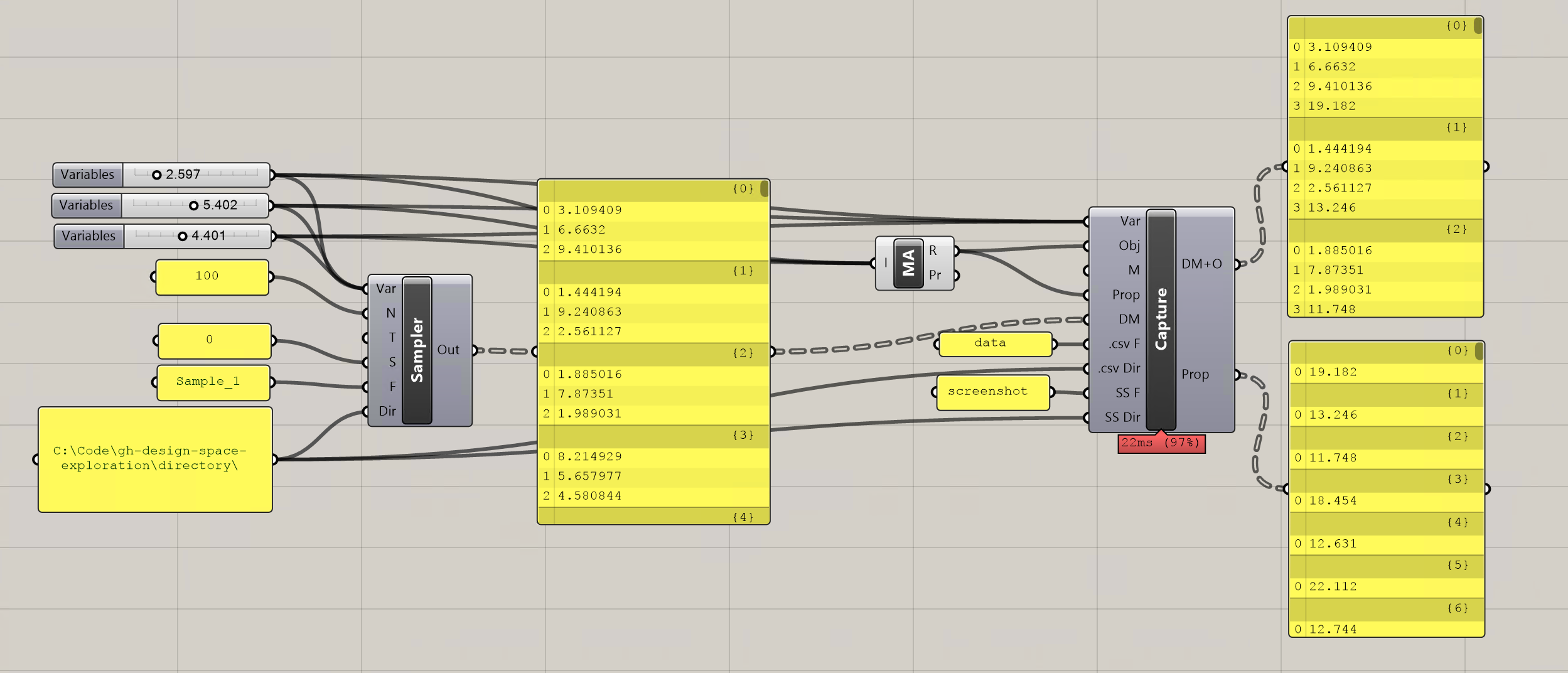
**Directory (Dir) –** The location on your computer where the Design Map will be saved. The directory MUST end with a “\” – otherwise, the file will be written one level higher than the intended directory.

**Outputs:**

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

** 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, or both. 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:**

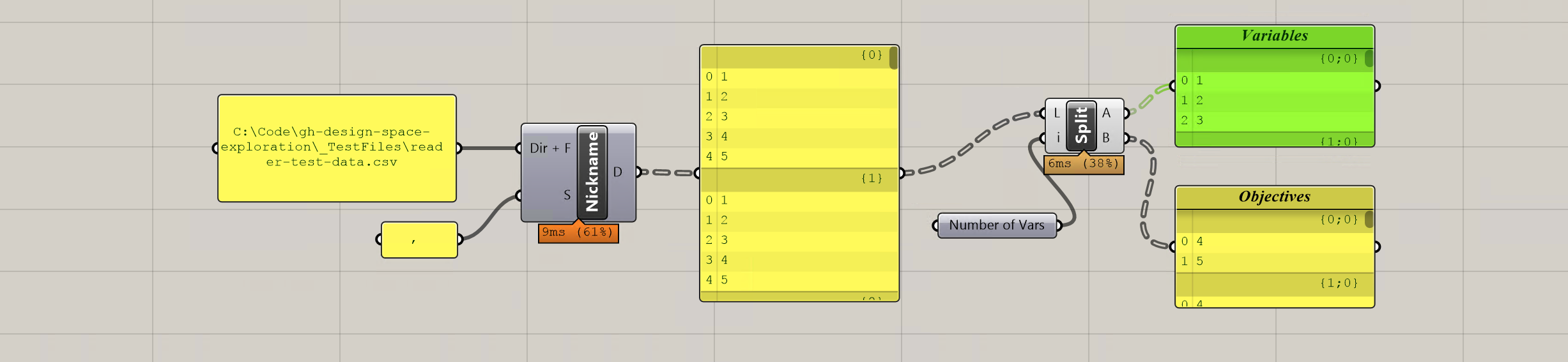
**Design Map + Objectives (DM+O) –** A nested list of design vectors and their objective values that span the design space being explored. Standard format for each design is the design vector followed by as many objective values as are recorded.

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

****

**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 “,”.

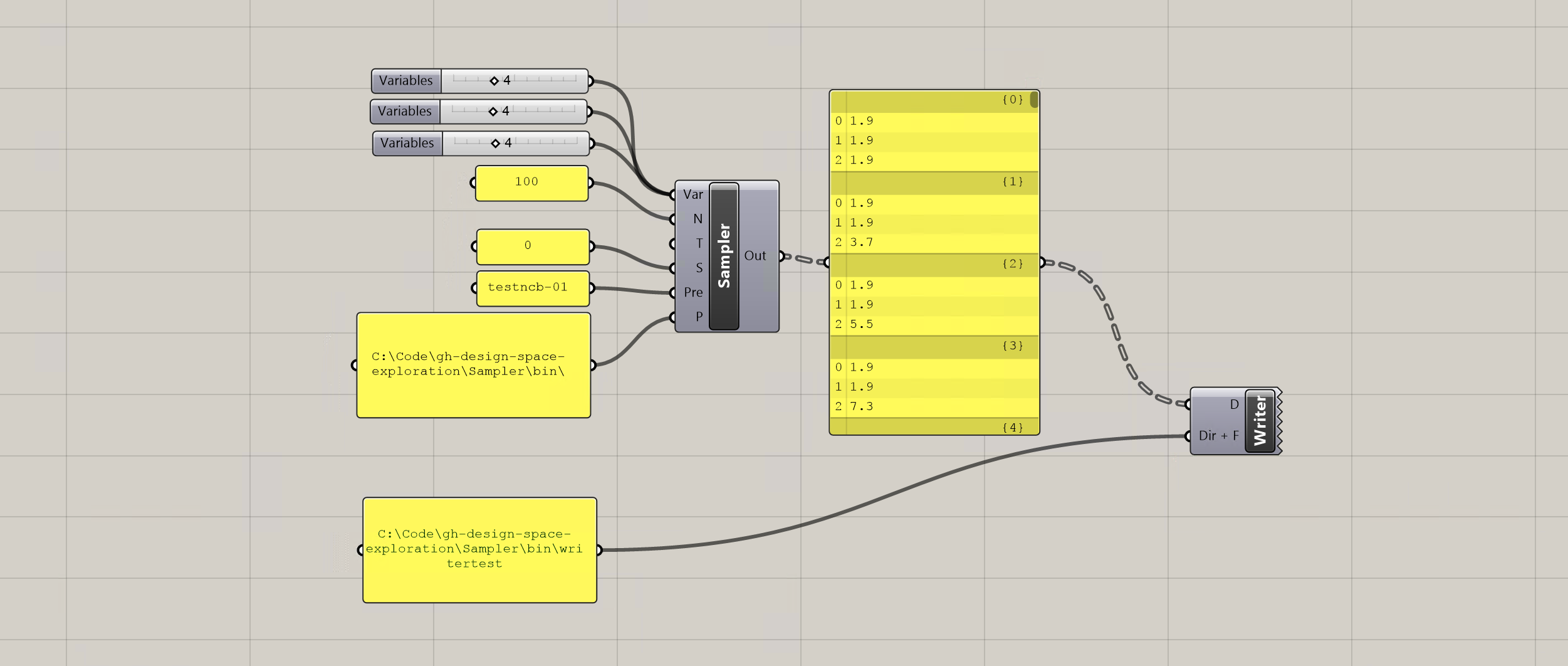
**Outputs:**

**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.



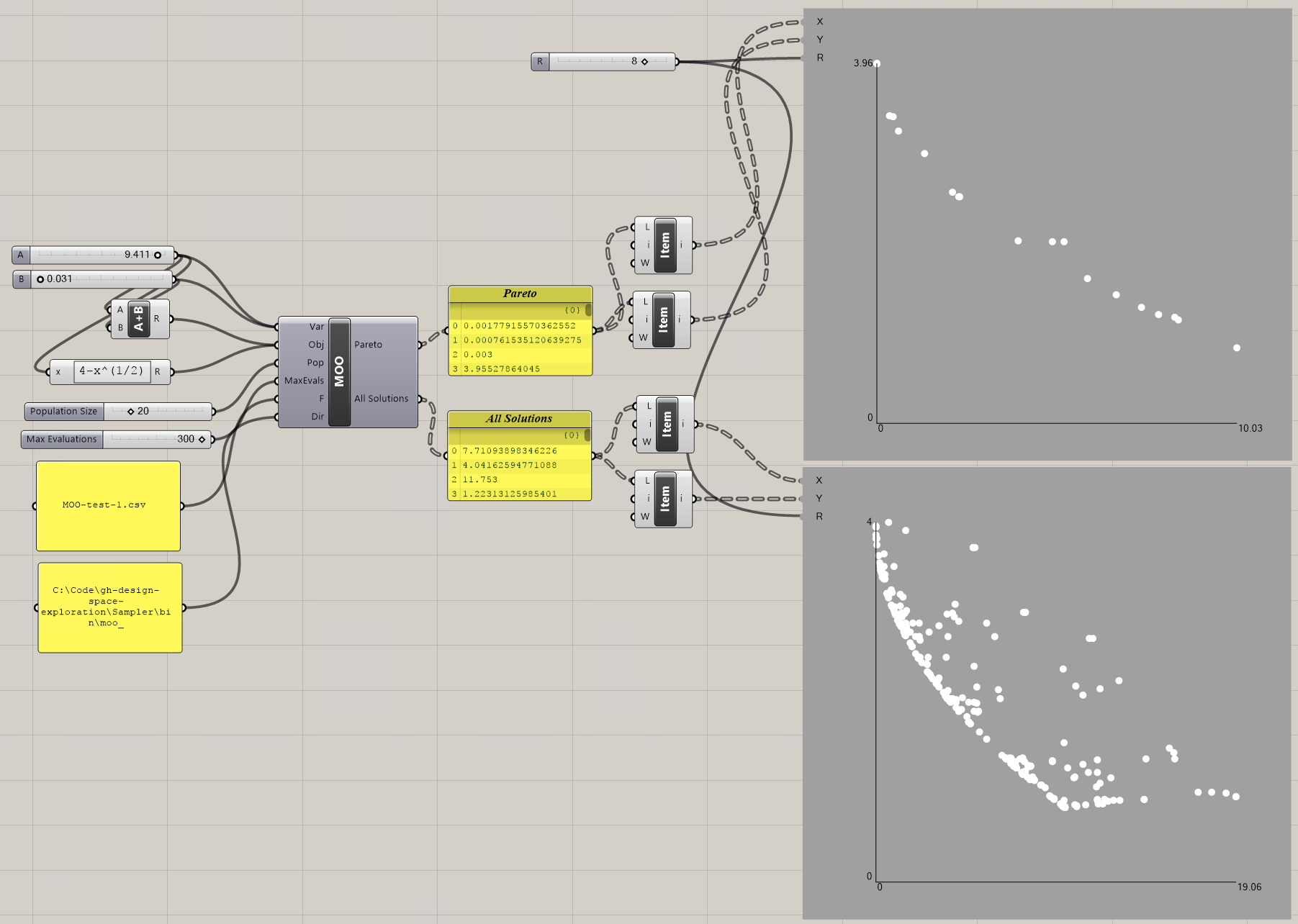
**Inputs:**

**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.

** 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.



**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 number of generations tested while the algorithm is running will be the maximum evaluations divided by the population. **Note:** due to the way in which the NSGA-II is implemented, the population size *must* be even, and it should be a factor of the Maximum Evaluations. MOO will still run if the population is not a factor of MaxEval, but it will not run if the population is even.

**Maximum Evaluations (MaxEvals)** – Sets the amount of function evaluations to be conducted during the optimization. Since there is no “threshold” mode for this version of MOO, the MaxEvals effectively sets the stopping condition for the optimization—if the results do not satisfactorily represent the Pareto front, more evaluations, and consequently more generations, should be attempted. MaxEvals should be a multiple of the Population size. If your objective function evaluations take a long time, that duration multiplied by the number of evaluations 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 MUST end with a “\” – otherwise, the file will be written one level higher than the intended directory.

**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.

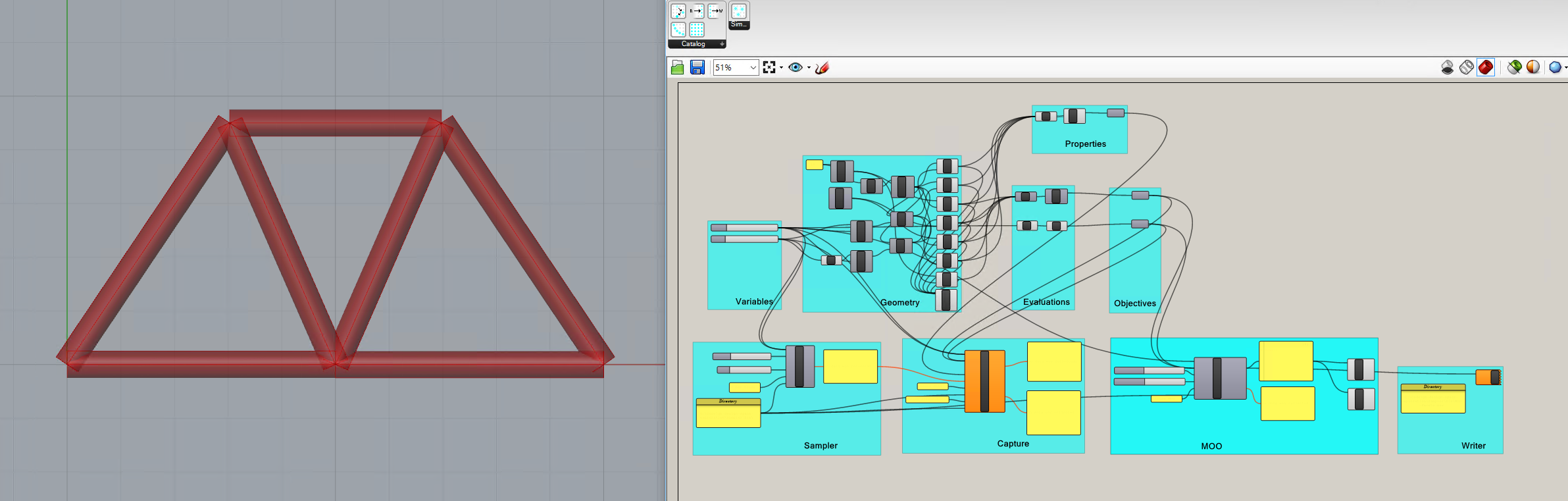
**2.2 Catalog - Getting Started Example**

**7 Bar Truss**

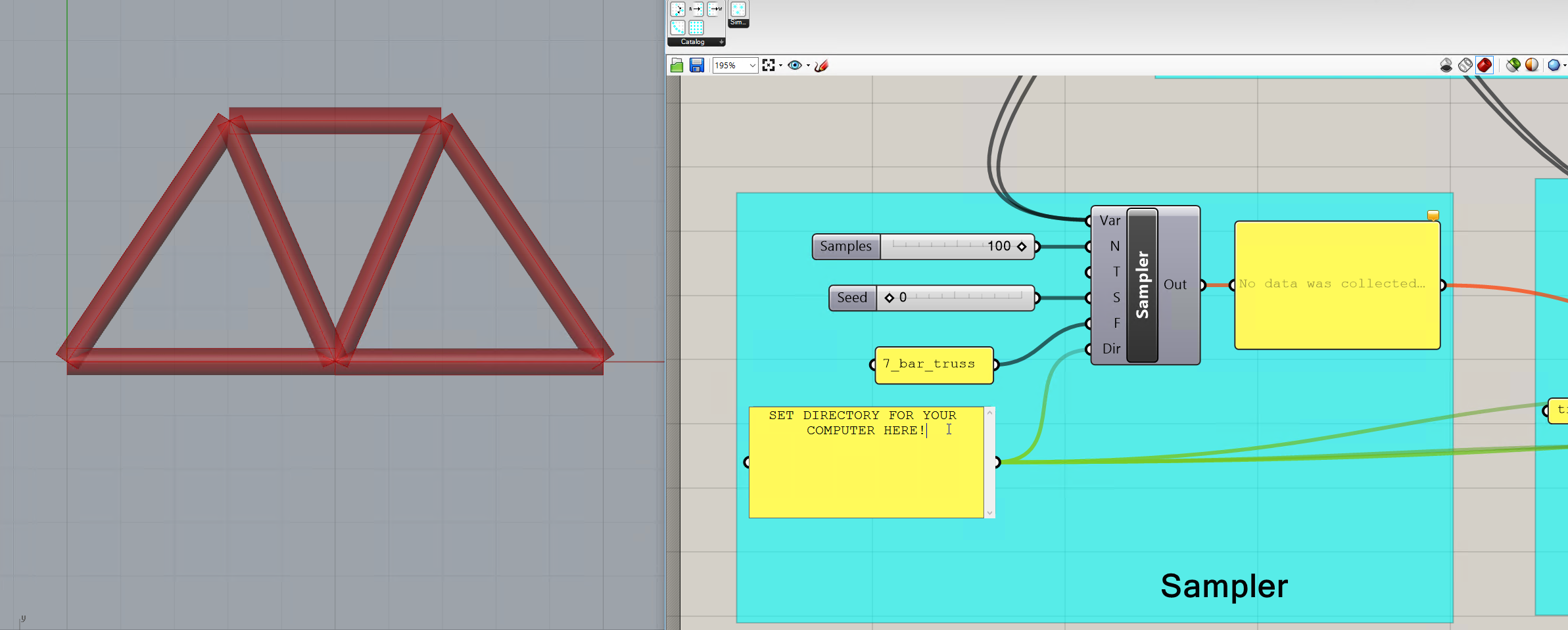
This script demonstrates a simple workflow for how the DSE components can all be used together to explore a design space and run multi-objective optimizations. In addition to the Design Space Exploration components, the script contains a 2-variable, parametric, geometric definition of a 7-bar truss,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

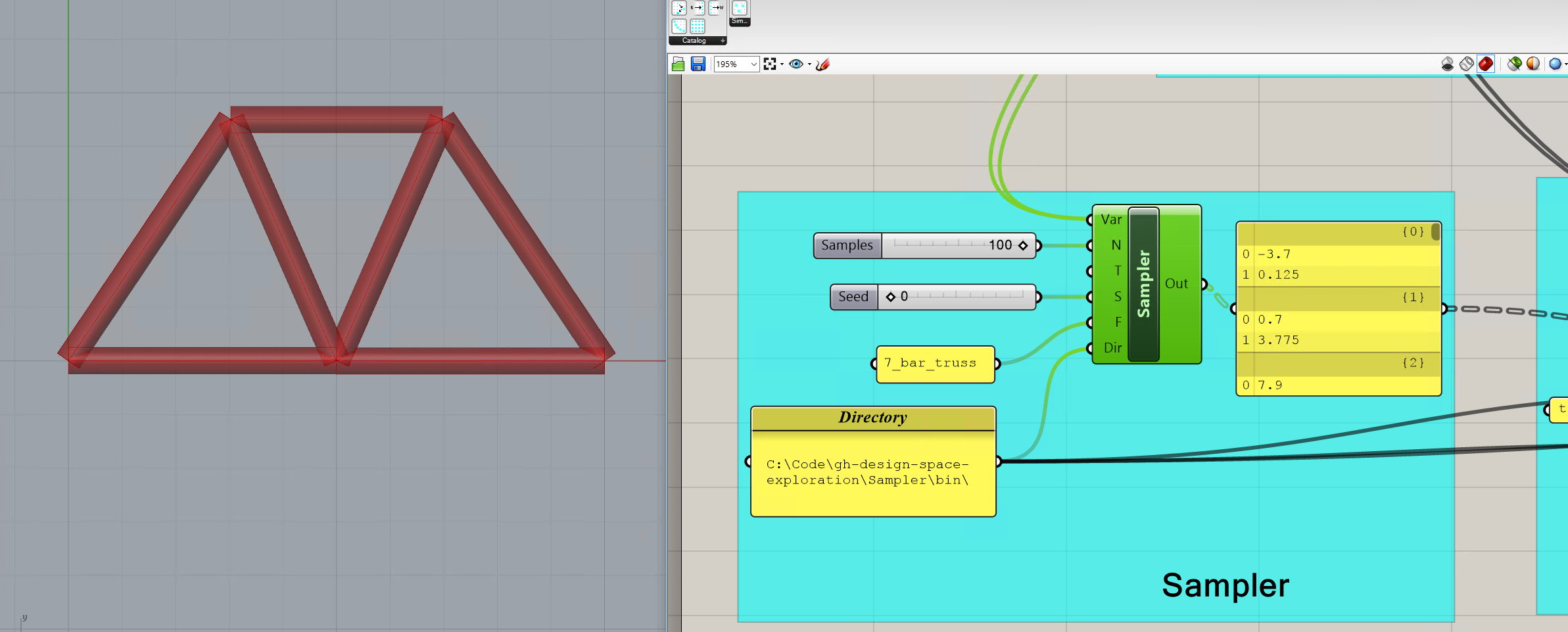
When opening the script, the parametric truss geometry can first be explored live by adjusting the variable sliders directly:



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:

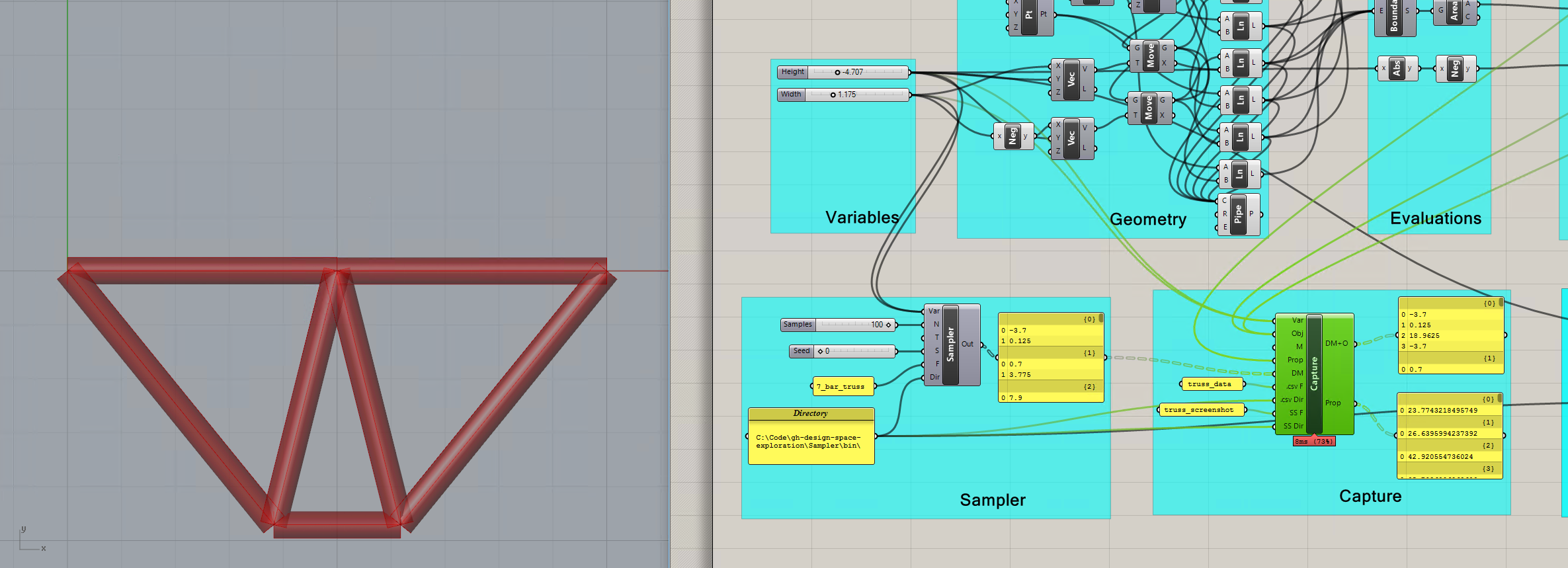


Once the Directory is set, the rest of the components will become functional. To take a representative sample of all potential truss options, doubleclick the **Sampler** component. This creates a Design Map of potential truss variable settings:



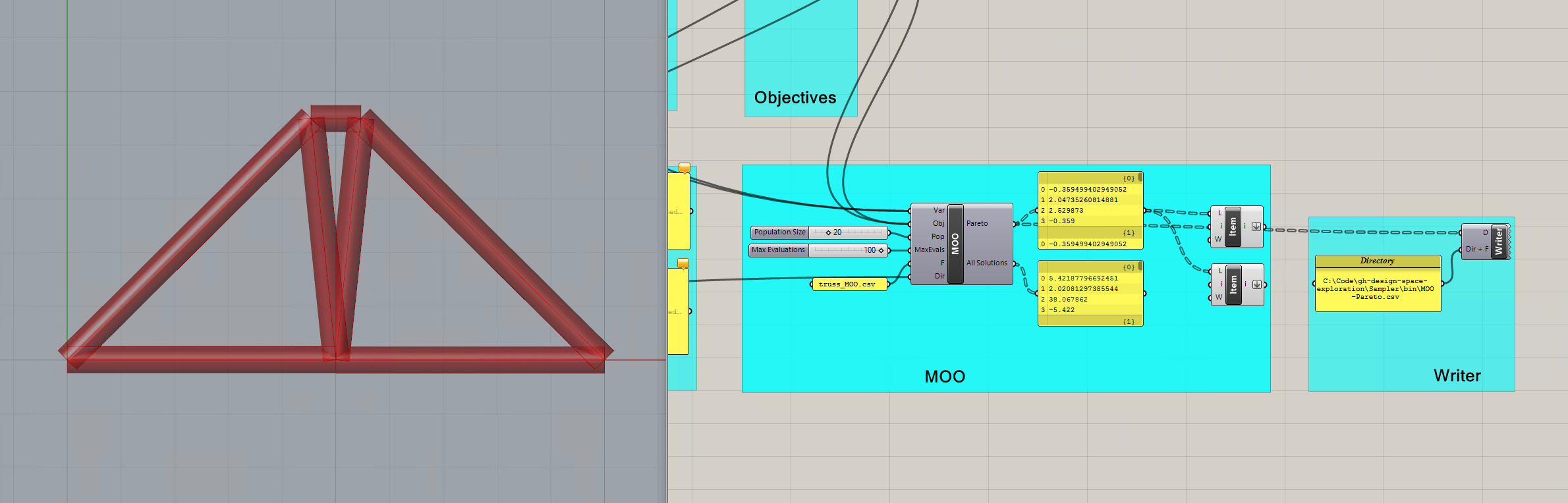
Alternatively, the **Reader** component could be used to read in a Design Map previously saved in a Directory.

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:

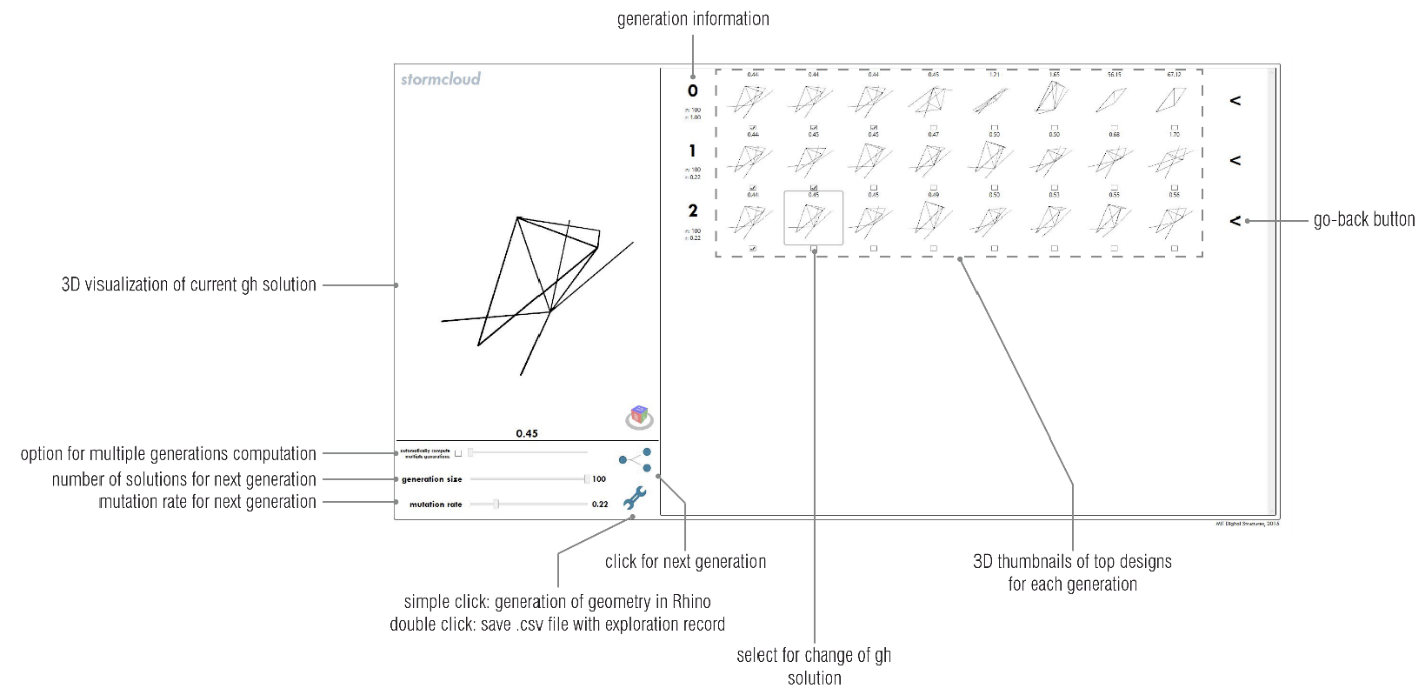


**3.1 Stormcloud Description**

The tool is implemented as a single component placed on the Grasshopper canvas (see Figure 1) Double clicking on the component opens the user interface of stormcloud. The component takes three different input – geometry (in the form of lines), score, and design variables - and has no output parameter. Data should be flattened when lists are used for the geometry lines. 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.

**3.2 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:

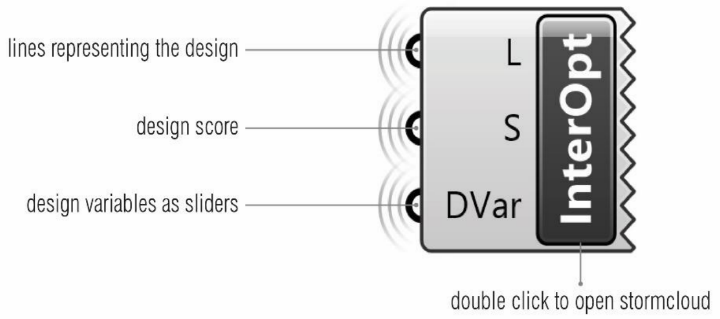


UI features include:

* 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.

**3.3 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.

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.

**4.1 References**

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.

Acknowledgements

Renaud

Jonathas

Anthony

Caitlin

Nathan

Stavros

Figure out open source – figure out rules to follow

License – what are we releasing over it?

Math.net

Accord?