Spatial Optimisations

Merging depthmapX, spatial graph networks and evolutionary design in Grasshopper

Reinhard Koenig[1], Tasos Varoudis[2]
[1] ETH Zurich and Bauhaus-University Weimar [2] University College London, Bartlett School of Architecture
[1] www.uni-weimar.de/computational-architecture
[2] http://www.bartlett.ucl.ac.uk/people/?upi=TVARO74
[1] reinhard.koenig@arch.ethz.ch [2] t.varoudis@ucl.ac.uk

In the Space Syntax community, the standard tool for computing all kinds of spatial graph network measures is depthmapX (Turner, 2004; Varoudis, 2012). The process of evaluating many design variants of networks is relatively complicated, since they need to be drawn in a separated CAD system, exported and imported in depthmapX via dxf file format. This procedure disables a continuous integration into a design process. Furthermore, the standalone character of depthmapX makes it impossible to use its network centrality calculation for optimization processes. To overcome this limitations, we present in this paper the first steps of experimenting with a Grasshopper component (reference omitted until final version) that can access the functions of depthmapX and integrate them into Grasshopper/Rhino3D. Here the component is implemented in a way that it can be used directly for an evolutionary algorithm (EA) implemented in a Python scripting component in Grasshopper.

Keywords: *Space Syntax, Evolutionary Algorithm, Grasshopper, Python, DepthmapX, Optimization*

CONCEPT

The guiding principle behind the presented network optimization is that if we understand the idea of various centrality measures, we can apply it for a backcasting approach. The main idea of backcasting is that we define a certain goal in the beginning and ask what actions must be taken to get there. In our context this means that we define the requested centrality for certain areas or streets in a city and the corresponding street network or a new part of it is computed. To achieve this, we implement an EA to find automatically the optimal street network for a certain goal. The EA concept follows an approach presented by Koenig, Treyer, and Schmitt (2013).

According to Weber, Müller, Wonka, and Gross (2009), the synthesis of urban structures consists of a sequence of several processes: the creation of a road network, the definition of land use and parcelling,

and building placement. We focus in the following on the first step. Systems have been developed for the procedural creation of road networks based on L-systems (Parish & Müller, 2001). In particular, the system CityEngine by ESRI facilitates the three-dimensional, rule-based modelling of cities and urban structures to the level of building details (Gool et al., 2006; Weber et al., 2009). In all these examples, the rules for the creation of an urban design solution have to be specified a priori in detail. The rules of generative or procedural algorithms are also very technical, abstract and not related to a planning problem. More importantly, they are not combined seamlessly with evaluation models and optimization methods. With these methods we therefore "have a model that can generate designs but has no means of establishing whether those designs are any good" (Radford & Gero, 1988, p. 20).

To achieve more advanced and more meaningful street network synthesis, we therefore need to find a representation that is able to create realistic network and can incorporate a performance measures (objective functions) that can be defined by a designer. This information should make it possible for the synthesis system to generate a correspondingly large amount of possible design solutions.

EVOLUTIONARY OPTIMIZATION

The basic technique we use for synthesizing geometry is evolutionary algorithms (EA) due to their flexibility with regard to problem representation as well as their robustness. This allows us to flexibly experiment with how we technically encode a design problem in the knowledge that the EA still work in an acceptable way even if we have a poor technical implementation. EA can be applied on various scales (Coates & Derix, 2008; Derix, 2009), for example to synthesize layout design (Koenig & Knecht, 2014), building volume arrangement (Koenig, 2015), urban district planning (Knecht & Koenig, 2012), or network development (Koenig et al., 2013; Schaffranek & Vasku, 2013). The EA may be supplemented by a number of local search strategies in order to optimize its calculation speed (Koenig & Schneider, 2012).

During the computer-supported design process, planners obtain immediate feedback in the form of at least one design solution that fulfill the formulated design requirements as well as possible. The presented system for synthesizing designs offers the possibility to experiment with various restrictions and objectives for a street network design. This is an important feature since the definition of a design problem can be considered as a main step towards its solution (Rittel & Webber, 1973).

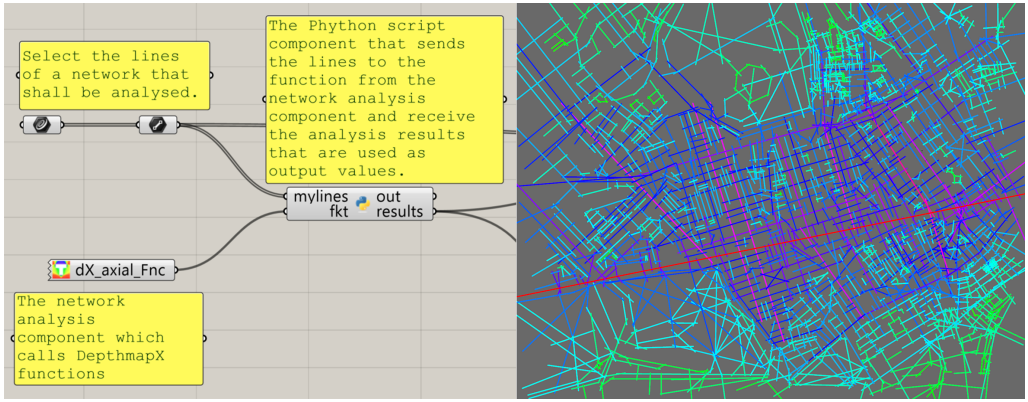
TECHNICAL DESCRIPTION

The experimental Grasshopper component for network analysis (reference omitted until final version) accesses depthmapX via a remote network call. Therefore, you need to run depthmapX (currently depthmapXnet 0.30) in the background. The component is designed for being used directly as a (real-time parametric solver) network analyzer and adapted here as fitness function in the EA for the purpose of the workshop. In principle can be used independently of the optimization if we send input data to it and receive the results from it by using a custom Python script as shown in Figure 1 on the left. On the right side of Figure 1 the results of an exemplary network analysis are shown. The advantage of using the depthmapX Grasshopper component is that if we manipulate the street network in Rhino and the centrality measures are immediately updated. So far it is possible to use an axial map representation and steps as distance measures. This means it is not useful to submit street segments and distances are computed in steps to arrive from one place to the other, whereas to pass an axial line is counted as one step. Other options may follow in next versions of the component.

The code in the Python component (Figure 1) to read the results from the depthmapX Grasshopper component (dX_acial_Fnc in Figure 1) is the following:

```
import rhinoscriptsyntax as rs
lines = []
print(len(mylines))
```

Figure 1
Figure 1. Left: The new network analysis component (dX_acial_Fnc) connecting to depthmapX and the Python script component to control it. Right: Result of the network analysis showing global Integration for an area in the city of London. Red represents high and green low values.




```
for i in range(len(mylines)):
    lines.append(mylines[i])

rad_list = "R, -1"#, 10"
do_choice = 1
results = fkt(lines, rad_list,
 ↪ do_choice)
spRS = results.splitlines()
str2d = []
for i in range(len(spRS)):
    str2d.append(spRS[i].split(','))

place = -1

for i in range(len(str2d[0])):
    #if str2d[0][i] == "Integration [
     ↪ HH] R10":
    #if str2d[0][i] == "Integration [
     ↪ HH]":
    if str2d[0][i] == "Choice [Line
     ↪ Length Wgt]":
        place = i
        print(place)

out = []
for i in range(len(str2d)):
    out.append(str2d[i][place])

strout = ",".join(out)
results = strout
```

The parameter rad_list allows to define the radiuses that shall be analyzed. The results from depthmapX can be acced in the for loop by this line:

```
if str2d[0][i] == "Choice [Line Length
 ↪  Wgt]":
```

whereas you can define the type of result that shall be used by the corresponding string. Two other options are out-commented.

The depthmapX Grasshopper component returns a function, which is used to communicate with in from the Python script:

```
results = fkt(lines, rad_list,
 ↪ do_choice)
```

This function is used in the loop of the EA. This implementation has the advantage that we don't need any additional looping components in Grasshopper which are usually difficult to handle and would make the optimization process slower.

How the EA is implemented is described in the following example. In Figure 2 on the left the initial street network is shown. The yellow marked line serve as reference line for computing the fitness values for our design variants. The objective is to maximize the integration value of the reference line by placing 6 additional lines. The small red points are the pre-defined references for start and end points

of the new lines. Figure 2 in the middle illustrates the Python optimization component. Its inputs are the reference points (refPoints), the existing network (environm), the reference line (refFitLine), the number of lines that shall be added (linesToAdd), the number of iterations of the EA (generations), and the population size of the EA (populatSize). The outputs of the component are the resulting fitness values for the generated variants, and the best solution that was found. On the right side of Figure 2 an exemplary result is shown that was found with the settings of Figure 2, middle.

The function from the depthmapX component, is used for the evaluation operator of the EA. Since the code for the whole EA is too much for this article, we only show the evaluate method:

```
def Evaluate():
    rad_list = "R, -1, 10"
    do_choice = 1
    del childFitnessValues[:]
    for i in range(len(
     ↪ childChromosomes)):
        network = copy.copy(
         ↪ childChromosomes[i])
        if environm is not None: #
         ↪ test if there is a
         ↪ specified environment
            for k in range(len(
             ↪ environm)):
                network.append(
                 ↪ environm[k])

        stringResults = fitnessFkt(
         ↪ network, rad_list,
         ↪ do_choice)
        if isinstance(stringResults,
         ↪ basestring):
            tempFitness = ReadResults(
             ↪ stringResults, "
             ↪ Integration [HH]")
        else:
             tempFitness =
              ↪ stringResults

        #index of the reference line,
         ↪ which shall be used for
          ↪ fitness calculation
        if refFitLine is not None:
            idxRefLine = network.index
             ↪ (refFitLine)
            childFitnessValues.append(
             ↪ tempFitness[idxRefLine
             ↪ ])
        else:
            #childFitnessValues.append
             ↪ (sum(tempFitness)) #
             ↪ we use the sum of the
             ↪ fitness value
            childFitnessValues.append(
             ↪ max(tempFitness))  #
             ↪ we use maximization!
```

The access to the depthmapX component works very similar than in our first example. The centrality measures are used to define the fitness value of a variant in the last line:

```
childFitnessValues.append(max(
 ↪ tempFitness))
```

Of cause it's very easy now to change this function to use others than the max value of the Integration [HH] measure.

CASE STUDY

The Grasshopper component as well as the Python EA were tested in the context of a workshop at the (excluded for review) conference. The workshop was for both beginners and intermediate users. The idea of the workshop was to explore spatial design variants through computational and parametric design thinking. We used Rhino5, Grasshopper and the Python scripting language to introduce concepts behind depthmapX, Space Syntax and parametrism. By the new Grasshopper component we first demonstrated the advantages of real-time analysis within Rhino as well as the new depthmapX[net] that is the brain behind the new component.

In a next step the participants were introduced to the basics of EA. By providing the source code of our Python EA implementation, we showed them the most important parts of the code to quickly adapt it

to their own ideas. For this purpose, a complete understanding of the EA was not necessary. Already by basic changes in the code many possibilities to define own customized fitness functions result. For example, the participants experimented with minimize or maximize certain centrality measures and with a combination of various objectives in a summed fitness function.

CONCLUSIONS

We introduced the first attempt of a Grasshopper network analysis component (reference omitted for final version), which allows to compute spatial measures from the well-established analysis tool, depthmapX. This enables real-time analysis within Rhino and the usage of the analysis for, among others, an EA as presented here. In our tests the implemented components run stable and can be adapted to various planning scenarios. The parametric approach and the integration into Grasshopper allows the usage of the results of the network analysis or the EA optimization process for other even more complex scenarios. Current objectives of our work focus on research of synthesis of urban structures from the street network and the resulting street blocks, parcels and buildings using this set of new tools.

In a next step we plan to extend the depthmapX functions that are accessible via the Grasshopper component. Furthermore, we intend to develop the EA Python optimization component to provide it as a ready-made component which can be feed with custom gens, selectors and fitness functions for the representation of individual design problems .

REFERENCES

Coates, P and Derix, C 2008 'Smart Solutions for Spatial Planning: A Design Support System for Urban Generative Design', *Architecture 'in computro': 26th eCAADe*, Antwerpen, Belgium, pp. 231-238

Derix, C 2009, 'In-Between Architecture Computation In-Between Architecture Computation', *International Journal of Architectural Computing (IJAC)*, 07(04), pp. 565-586

Gool, P, Müller, P, Wonka, P, Haegler, S, Ulmer, A and Van Gool, L 2006 'Procedural Modeling of Buildings', *ACM SIGGRAPH*, Boston, pp. 614-623

Knecht, K and Koenig, R 2012 'Automatische Grundstücksumlegung mithilfe von Unterteilungsalgorithmen und typenbasierte Generierung von Stadtstrukturen', *Arbeitspapiere Informatik in der Architektur*, Weimar, pp. 1-21

Koenig, R 2015, 'Urban Design Synthesis for Building Layouts Urban Design Synthesis for Building Layouts based on Evolutionary Many-Criteria Optimization', *International Journal of Architectural Computing*, 13(3+4), pp. 257-270

Koenig, R and Knecht, K 2014, 'Comparing two evolutionary algorithm based methods for layout generation: Dense packing versus subdivision', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28(03), pp. 285-299

Koenig, R and Schneider, S 2012, 'Hierarchical structuring of layout problems in an interactive evolutionary layout system', *AIEDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(2), pp. 129-142

Koenig, R, Treyer, L and Schmitt, G 2013 'Graphical smalltalk with my optimization system for urban

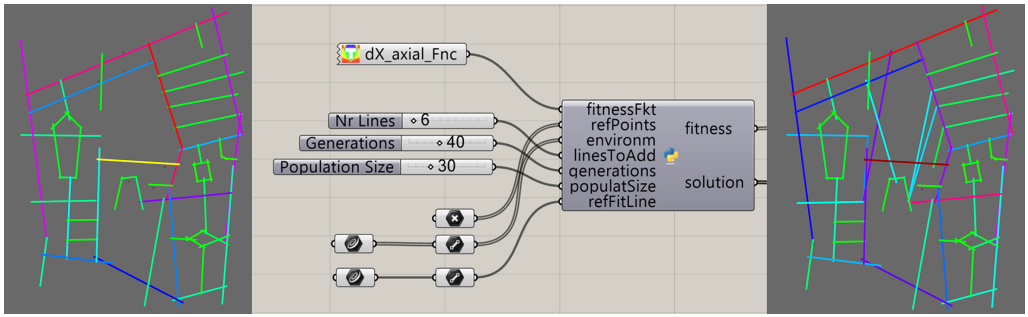


Figure 2
Figure 2. Left: Initial street network with the area to be filled, the red reference points, and the yellow reference line for the fitness function. Red represents high and green low values. Middle: The Python optimization component with input parameters and the depthmapX

planning tasks', *Proceedings of the 31st eCAADe Conference – Volume 2*, Delft, Netherlands, pp. 195-203
Parish, Y and Müller, P 2001 'Procedural Modeling of Cities', *SIGGRAPH*, Los Angeles, CA, pp. 301-308
Radford, A and Gero, JS 1988, *Design by optimization in architecture, building and construction*, Van Nostrand Reinhold, New York and Wokingham
Rittel, H and Webber, M 1973, 'Dilemmas in a General Theory of Planning', *Policy Sciences*, 4, pp. 155-169
Schaffranek, R and Vasku, M 2013 'Space Synatax for Generative Design: On the application of a new tool', *Ninth International Space Syntax Symposium*, Seoul, p. 12
Turner, A 2004, *Depthmap 4: a researcher's handbook*, Bartlett School of Graduate Studies, University College London, London
Weber, B, Müller, P, Wonka, P and Gross, M 2009 'Interactive geometric simulation of 4D cities', *Eurographics*, pp. 481-492
[1] https://varoudis.github.io/depthmapX/