




# Interactive Urban Synthesis

## Computational Methods for Fast Prototyping of Urban Design Proposals

Reinhard Koenig<sup>1,2,3</sup> , Yufan Miao<sup>4</sup> , Katja Knecht<sup>4,5</sup>,  
Peter Buš<sup>3</sup> , and Chang Mei-Chih<sup>3</sup>

<sup>1</sup> Center for Energy, Austrian Institute of Technology, Graz, Austria  
reinhard.koenig@ait.ac.at

<sup>2</sup> Faculty of Architecture and Urbanism, Bauhaus-University Weimar,  
Weimar, Germany

<sup>3</sup> Department of Architecture ETH Zürich, Zurich, Switzerland  
{bus, chang}@arch.ethz.ch

<sup>4</sup> Future Cities Laboratory, Singapore-ETH Centre, Singapore, Singapore  
{miao, katja.knecht}@arch.ethz.ch

<sup>5</sup> Queen Mary University of London, London, UK

**Abstract.** In this paper, we present a method for generating fast conceptual urban design prototypes. We synthesize spatial configurations for street networks, parcels and building volumes. Therefore, we address the problem of implementing custom data structures for these configurations and how the generation process can be controlled and parameterized. We exemplify our method by the development of new components for Grasshopper/Rhino3D and their application in the scope of selected case studies. By means of these components, we show use case applications of the synthesis algorithms. In the conclusion, we reflect on the advantages of being able to generate fast urban design prototypes, but we also discuss the disadvantages of the concept and the usage of Grasshopper as a user interface.

**Keywords:** Procedural grammars · Artificial intelligence in design · Urban synthesis · Generative design · Grasshopper plugin · Cognitive design computing

## 1 Introduction and Aims

Modern cities exhibit growing complexities and dynamics where traditional urban design methods, which still rely on static and sectorial approaches, reach their limits for fast adaptation. At the same time, with the advent of big data and the improvement of our abilities to process large amounts of data as well as advances in artificial intelligence such as cognitive computing, new opportunities emerge for future-oriented computational design. Inspired by IBM's Watson, cognitive computing provides a productive combination of human cognition and computing power to support automated problem solving [1].

The integration of cognitive computing in computational design assistance systems could revolutionize the urban design and planning process by allowing to model urban complexities and changing urban dynamics and thus helping to address and respond to these challenges in the urban design and planning processes. A framework for this approach called ‘cognitive design computing’ has been proposed by Koenig and Schmitt [2]. The framework rests on four pillars: data analysis, user interaction and learning, geometry synthesis, and evolutionary multi-criterion optimization (EMO).

Following this research track, we present in this paper a computational method for the third pillar, a fast synthesis of urban planning prototypes. The main idea of the method is that the designer can define certain restrictions and the corresponding spatial configurations are generated automatically. To allow a systematic design space exploration, the designer needs to interact with the generative procedure by changing the design framework as well as manipulating the initial geometric elements that are used as the starting point for the urban synthesis process. In this paper we focus on i) the technical description of the methods used for the generation of urban fabric, ii) the presentation of first exemplary case studies, and iii) the long-term idea of our approach concerning the data structure used.

## 2 State of the Art

For the integration of optimization techniques in design or planning processes, we build on the methodology described by Radford and Gero [3], which describes a concept of how problem representation, generative methods, simulation methods, and decision-making can be combined to form a design-oriented optimization method. Similar concepts have been presented in other research fields, for example the concept of Design Synthesis in engineering [4].

Current commercial solutions for generative or procedural modeling, for example CityEngine, exemplify the difficulties of integrating them in the early stages of planning processes: the results of generative or procedural algorithms are based on a set of rules, which are very technical, abstract and not related to a planning problem. As a result, planners don’t understand the control mechanism of the system and its possible benefit. In other words, *“we have a model that can generate designs but has no means of establishing whether those designs are any good”* [3]. With the methods and components we present in this paper we want to improve this situation.

According to Weber, Müller, Wonka, and Gross [5] the modeling of urban structures consists of a sequence of several processes: the creation of a road network, the definition of land use areas, parceling and building placement. In this context, additive processes are relevant for the generation of road networks and the placement of buildings. Systems have been developed, for example, for procedural creation of road networks based on L-systems [6] and tensor fields [7]. In particular, the system CityEngine facilitates the planning and three-dimensional, rule-based modelling of cities and urban structures to the level of building details [5, 8].

Some components of CityEngine use subdivision methods, for example, for the creation of plots. Using subdivision methods, spatial configurations are generated from the division of a predetermined shape outline in smaller parts. Early work using

so-called subdivision trees as subdivision method originates from Mitchell, Steadman, and Liggett [9] as well as Stiny and Mitchell [10]. A more recent application of this procedure can be found in the work of Duarte et al, who used it to generate floor plan layouts [11] and urban structures [12]. In these examples, both constraints and rules for the creation of a solution have to be specified a priori in detail. The combination of subdivision trees with genetic programming [13] results in promising formal options [14]. The Kaisersrot project [15] employs a different approach creating plots and road network using Voronoi diagrams.

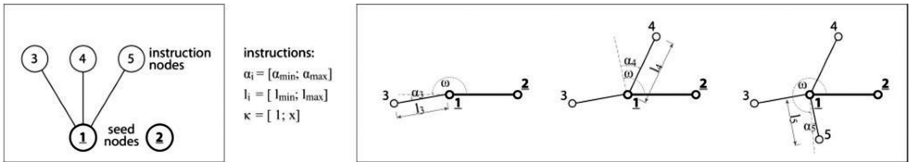
### 3 Methods

In contrast to other more experimental [16, 17] or established [6] urban procedural methods, we introduce a new data structure, which makes it possible to realize computational design that is based on the Backcasting approach [2]. Backcasting allows to define what performance a design solution should achieve and to automatically synthesize a set of best possible solutions.

Our urban synthesis method is composed of three procedural steps, as described by Weber et al. [5]. Firstly, street networks are initialized and extended. Secondly, blocks are defined and extracted from the street networks. Lastly, blocks are sliced into parcels, onto which buildings are placed on.

For the implementation, the urban synthesis is realized in CPlan, an open source library for computational urban design proposed by Koenig and colleges [18, 19], whereas interaction functionalities are provided by the built-in functions of Rhino3D. Based on CPlan, we developed new components for the visual programming interface Grasshopper to connect CPlan to Rhino3D.

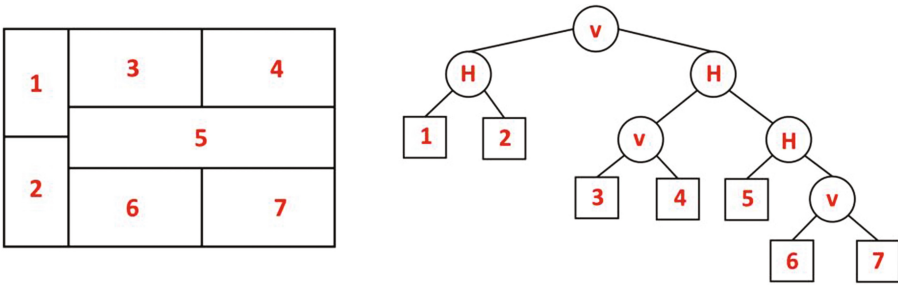
For the generation of street networks, instruction trees are proposed (Fig. 1) to represent instructions of how a street network should grow from initial nodes by defining three key characteristic parameters: the deviation angles of street segments  $\alpha_i$ , the lengths of street segments  $l_i$ , and the possible arms at a crossroad  $\kappa$  [20]. The advantage of this data structure is that it can easily parameterize both the geometric and topological structures of the street networks into tree structures, which provide convenience for the mutation and crossover in evolutionary algorithms.



**Fig. 1.** Mapping process from an instruction tree to a street network. Figure adapted from Koenig et al. [20].

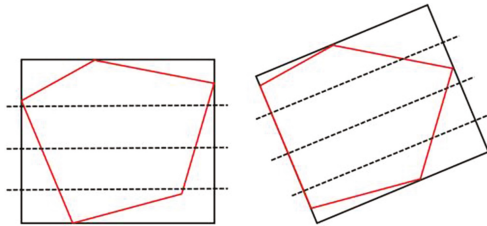
For the extraction of the blocks, street networks are converted into their dual directed graphs. In this graph representation, edge rings are searched and labeled for transformation into their geometric representation as polygons.

For the generation of parcels and buildings, the blocks are subdivided based on parameters specified by the planners. For their representation, a slicing tree structure is adopted given the resulting parcels possess the desired sizes (geometry) and neighborhood relationships (topology) [21, 22]. As is illustrated in Fig. 2, the slicing tree on the left is an abstract representation of the parcels on the right. After the generation of the parcels, buildings are placed inside each parcel according to the characteristics of different building typologies. In our program, we provide four typologies, namely, ‘row’, ‘column’, ‘freestanding’, and ‘block’.



**Fig. 2.** The slicing tree representation of the parcels. The tree on the left is the abstract representation of the sliced parcels on the right. ‘H’ and ‘V’ represent horizontal slicing and vertical slicing respectively. The leaves are indexed identical to their geometric counterpart as parcel on the right. Indices also reflect the branch generating order in the program [22].

The slicing procedure, which generates the parcel layout, is limited to ensure that all the parcels have a direct connection to a street. This is illustrated exemplary in Fig. 2, in which parcel 5 is longer than the others but not sliced further because there is only one edge on the street, whose length is not long enough for further slicing. However, in reality the shapes of street blocks are not always as regular as represented in Fig. 2. To deal with this, we use a rectangular bounding box for the polygon as



**Fig. 3.** Slicing based on the boundary box of irregular shapes. On the left, the slicing uses the edges of a normal bounding box whereas on the right, the slicing is oriented on the minimum bounding box [23].

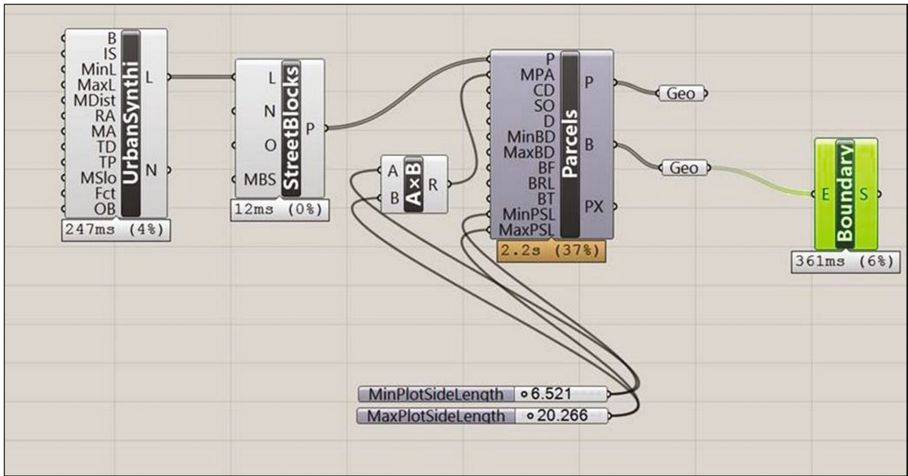


shown on the left in Fig. 3 [23]. This strategy keeps the slicing process working consistently. However, it often generates parcels with irregular shape, which are not suitable for urban design purposes. To improve this, we use a minimum bounding box as base shape for slicing as shown on the right side of Fig. 3.

## 4 New Grasshopper Components

For the development of the components in Grasshopper for Rhino 3D, each step in the generative process is implemented as one component (Fig. 4) including the generation of street networks, extracting street blocks from the street network, slicing the blocks into parcels and generating buildings inside the parcels. Grasshopper is currently a compromise interface for the CPlan framework given its popularity among designers and architects and the existing abundant and versatile add-ons. Its drawback are its limitations in allowing direct user interaction with generated urban configurations.

The street network component (Fig. 4A) has the following input parameters, which control the morphology of the resulting network. The algorithm works iteratively so that one segment is added after the others in an additive process. The input parameters for the component (Fig. 4A) are:

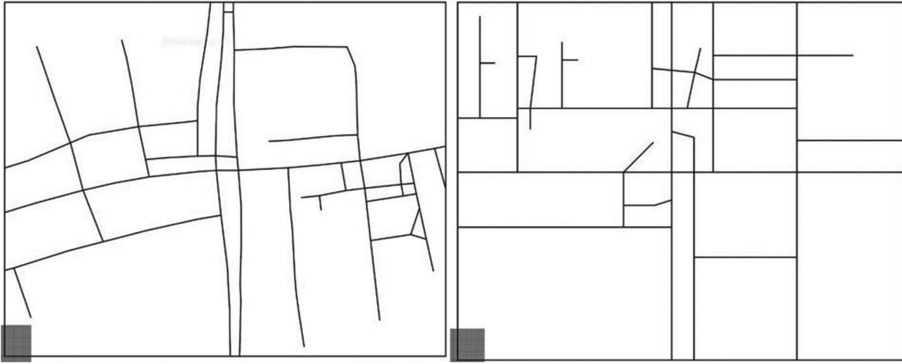


**Fig. 4.** The new grasshopper components. From left to right the components are used for creating A) a street network, B) street blocks, and C) parcels and buildings.

- *B*: A border polygon, which defines the border for the street network to be generated.
- *IS*: The initial street segment, from which the network starts to grow.
- *MinL*: The minimum length of a new street segment  $l_{min}$ .
- *MaxL*: The maximum length of a new street segment  $l_{max}$ .

- *MDist*: The minimum distance between two nodes of the network (between start and end points of the street segments).
- *RA*: Random value that is added to a regular angle to define the direction angle  $\alpha_i$  of a new street segment.
- *MA*: Maximum number of arms (streets) at crossroads  $\kappa$ .
- *TD*: Tree depth defines the size of the resulting street network. This means it defines the branch-levels for adding control-nodes as described in Fig. 1, on the left side.
- *TP*: Topography points represent the points of a topography surface. This input data may be used for custom functions how new street segments shall react to a given topography.
- *MSlo*: Maximum slope for a street segment (using topography information).
- *Fct*: A list of custom functions to define rules of how new street segments are added.

The output  $L$  of the street network component (Fig. 4A) returns a list of line segments that represent the network. The second output  $N$  is the raw data of the street network, which is useful for later optimization purposes. Figure 5 shows two exemplary resulting networks as output of the street network component.



**Fig. 5.** Two street networks generated with the street network component A shown in Fig. 4. The network on the left side is generated using default parameters. For the network on the right side the parameter  $RA$  is set to 0. This setting results in a network in which the segments are primarily rectangular to each other.

The line segments from output  $L$  can now be used as an input for the street blocks component (Fig. 4B), which allows the generation of street blocks based on the street network. The other parameters of the street blocks component are:

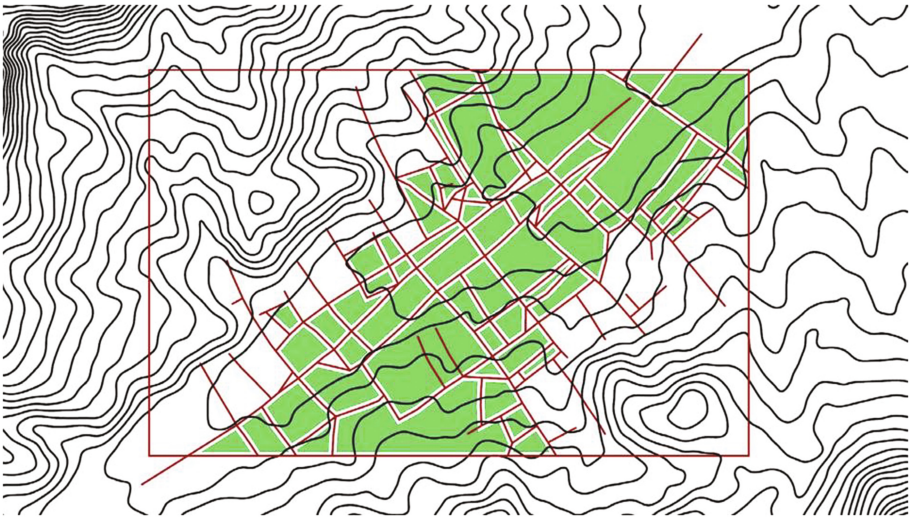
- $N$ : Raw data of the street network generation process.
- $O$ : Offset of the street blocks from the street segments.
- $MBS$ : Minimum block size. If the area of a polygon representing the block is smaller than this value, the block is not created.

The output  $P$  is a list of polygons, which represent the street blocks. To visualize the street blocks, we extrude the street blocks by using a boundary surface component that returns the green colored polygons on the right in Fig. 6.



**Fig. 6.** From a generated street network (component A shown in Fig. 4) with a custom border input  $B$  and three custom initial street segments  $IS$ , the street blocks are generated on the right side using the component B shown in Fig. 4

To show the possible effect of the topography input parameter  $TP$  of the street network component (Fig. 4A), we import data from the EarthExplorer website (<http://earthexplorer.usgs.gov>), and import the GeoTIFF with the help of the Grasshopper component *Elk2*. Based on the topography we can now restrict the network by adding new street segments only if the slope between start and end point is less than the  $MSlope$  parameter values (20 degrees by default). An exemplary street network reacting to a topography is illustrated in Fig. 7.



**Fig. 7.** Street network generated with a topographic input, a custom border  $B$ , and initial street segments  $IS$ . New street segments are only added if the slope of the segment is below the threshold value  $MSlope$  of  $20^\circ$ .

Finally, the third component generates parcels and buildings (Fig. 4C). It takes the street block polygons as input and subdivides them following the procedure illustrated in Figs. 2 and 3. The input parameters of the parcel and building component are:

- *P*: A list of polygons, which represent the street blocks.
- *MPA*: Max parcel area defines the maximum area of a parcel. This means the block is subdivided until all resulting parcels are smaller than this value.
- *CD*: If the distance between two parcel corner points is smaller than this value, they are merged.
- *SO*: Offset from the block border.
- *D*: Density, which defines the ratio between building footprint and plot area.
- *MinBD*: Minimum depth of buildings.
- *MaxBD*: Maximum depth of buildings.
- *BF*: Building front-line.
- *BRL*: Building restriction line, which defines the distance to the backside parcel border edge.
- *BT*: Building type - so far four types have been implemented: rows, columns, blocks, and freestanding houses [24].
- *MinPSL*: Minimum length of a side of a parcel.
- *MaxPSL*: Maximum length of a side of a parcel.

Some of the input parameters may contradict each other or result in an over constraint configuration. At the current stage of the development of the algorithm, not all of these issues are solved. Therefore, the output geometry generated by the component may give unsatisfactory results. As output parameters the component delivers:

- *P*: A list of polygons representing the generated parcels.
- *Bld*: A list of polygons inside the generated parcel polygons that represent the footprint of the generated buildings.
- *PX*: Data structure that may be used for a GeoJSON export.

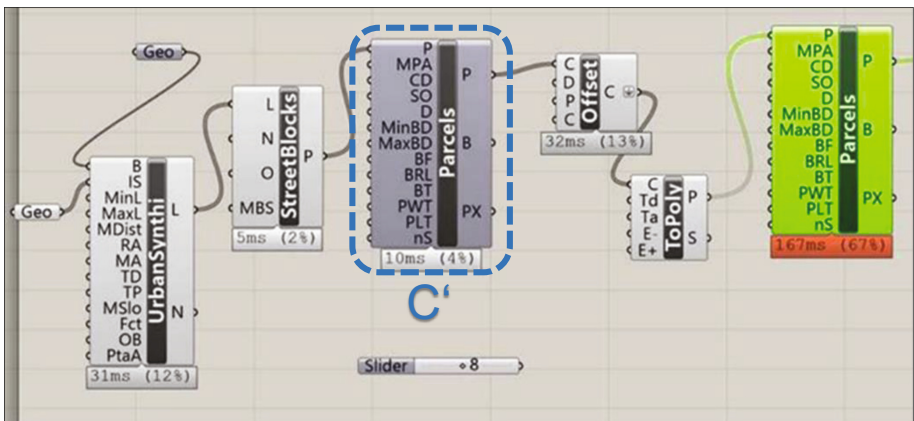
The simplest way to use the parceling component (Fig. 4C) is by connecting it with the polygon output of the street block component (Fig. 4B). Figure 8 shows an example of a result obtained by connecting all tree components (Fig. 4A–C). For some of the generated street blocks, the algorithm does not deliver the expected and intended outcome. Problems, which can be seen in the example, include that parcels and buildings cannot be created due to issues with the block geometry or that resulting parcels possess undesirable geometric properties. These issues will be addressed in the further development of the components.

Furthermore, over-constraint situations resulting in undesirable geometric configurations can be due to initially created street blocks, which are too large. To solve this issue, we use the parceling component to generate a secondary street network and to subdivide the blocks further (Fig. 9). This also allows a better control of the block sizes and a better use of the available space for the generation of parcels and buildings.

Figure 10 shows an exemplary urban fabric that is generated using the nested parceling component as shown in Fig. 9. The secondary streets generated by the parceling component C' in Fig. 9 produce street blocks with similar depth, which results in a relatively regular parceling structure.



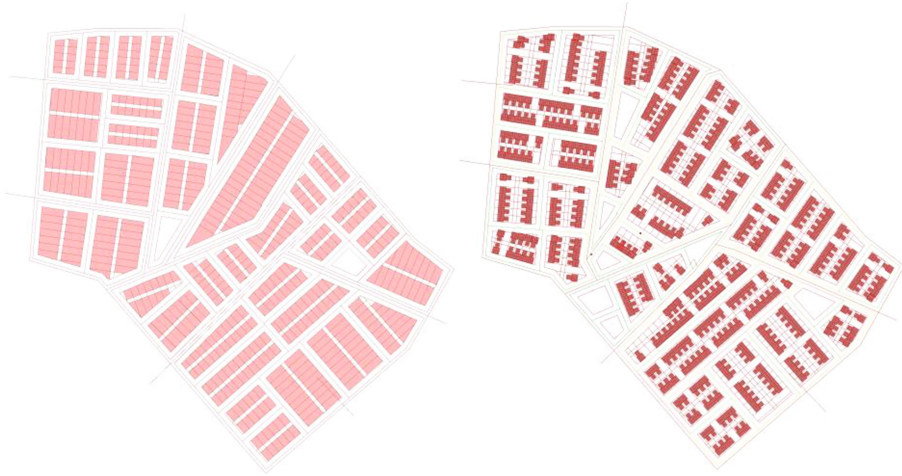
**Fig. 8.** Urban fabric generated by combining the street network, street blocks, and parceling components.



**Fig. 9.** A second parceling component is added to deal with large street blocks. Street blocks are firstly partitioned into smaller ones with the component marked as C' and then partitioned into the final parcels by the green component. (Color figure online)

The generated urban fabric can be controlled by the input parameters of the components shown in Figs. 4 and 9. In addition, the urban fabric can be controlled more directly and interactively by changing its border geometry and the initial street segments that are used as reference-inputs for the components. After moving or rotating one of these elements, the fabric is re-computed immediately. The main drawback of using Grasshopper and Rhino3D as a user interface is that it is not possible to directly manipulate the geometry, which is generated by the components, and have an interactive feedback loop, which would then adapt the remaining elements. The generated geometry can only be manipulated after finishing the algorithmic design process by “baking” the geometry permanently.





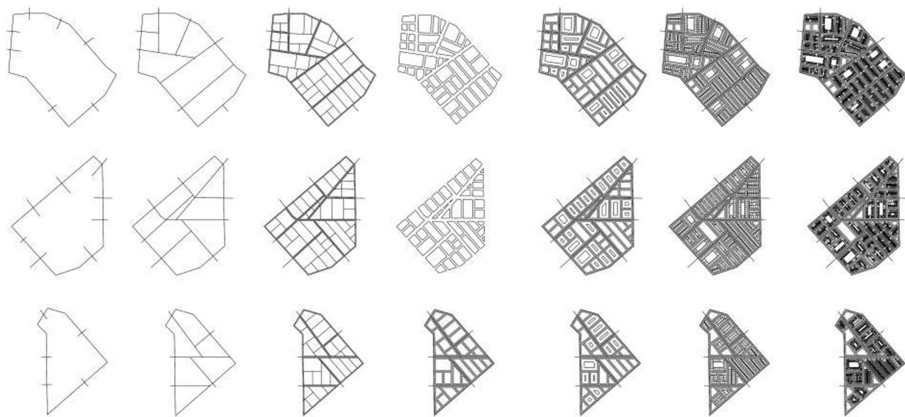
**Fig. 10.** Resulting urban fabric with more controlled street block sizes using a nested parceling component (Fig. 9).

## 5 Case Studies

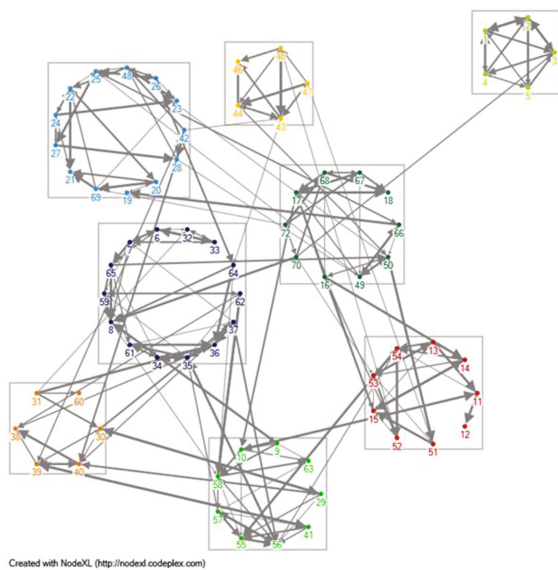
Our methods have been applied and tested in two case studies. The first case study is the EmpowerShack project (<http://u-tt.com/project/empower-shack/>). An important aim of the project was to develop a layout of streets and plots based on the design framework developed by the Urban Think Tank team (<http://u-tt.com/>). The design framework guides the development of a set of customizable functions, which limit the maximum width and length of a plot to ensure that a predefined house type can be placed on the plot.

The case study scenario for the EmpowerShack project includes the investigation of a larger urban area, in particular, the informal settlements in Enkanini in Cape Town. Three smaller selected neighborhoods were selected. Outline polygons representing the borders of the neighborhoods were defined as initial empty shapes for a new urban layout generation (first column in Fig. 11). According to the specific design requirements from the urban designers, each plot within the new urban block should be defined by its precise dimensions. This was achieved by an extension of the basic street network algorithm with new separate algorithms for the plot generation allowing for the definition of precise plot dimensions and taking into account specific predefined models of building prototypes. The plot generation approach combines the developed street network components with standard Grasshopper components. The results, as can be seen in Fig. 11, fulfill the requirements of the urban designers precisely in respect to the accuracy of the plots dimensions. The algorithm provides the necessary flexibility to incorporate other building prototypes and typologies.

In addition, the urban layout generation and building placement algorithm was enriched with a clustering feature (Fig. 12). The building units were assigned to certain households taking into account their preferences for a unit size, a specific location, and



**Fig. 11.** Various urban scenarios for different shapes of initial input polygons representing three different neighborhoods. The tool provides fast results and the designer can modify the initial positions of axis for the street network generation. The layout of streets and blocks rearranges accordingly. For more details on the generative process, see also the video-links listed in the acknowledgement.



**Fig. 12.** Clusters of neighborhood preferences. Each household was asked for the preferred other households it would like to live nearby. Cluster computation and visualization created with NodeXL (<http://nodexl.codeplex.com>).

the proximity to other households. The aim of this approach was to ensure that people living in the neighbourhoods could create or preserve communities. For this, we proposed a social network and integrated cognitive analysis to help arrange how people

live together and in consequence support an intimate relationship in the community. We use the collected information of the preferred proximities as social network structure. In this network, we can identify groups and arrange them in so called clusters or communities. For computing the clusters from given neighborhood preferences of the inhabitants as illustrated in Fig. 12, we used the NodeXL library [25].

Each cluster shows a network consisting of densely connected residents with their preferences. The difficulty was to transfer the household clusters from Fig. 12 to a spatial distribution of houses that form corresponding spatial communities. Distributing the same cluster of residents in proper building locations needed to confirm that each building had a shortest path between them. The measurement of distance in urban space depends on the cognitive distance [26], which represents a better empirical model of pedestrian movement than metric distance. For the computation of angular distances in Grasshopper we used the components developed by Fuchkina [27]. The ranked-shortest path for all buildings allowed to map the household clusters from Fig. 12 into nearby buildings (Fig. 13). The optimized cluster layout was limited by the minimum total metric area size, which allowed to create a more dense living arrangement. This method assisted the urban planner in allocating residents close to each other according to preferences in combination with an efficient plot packing strategy (Fig. 13).

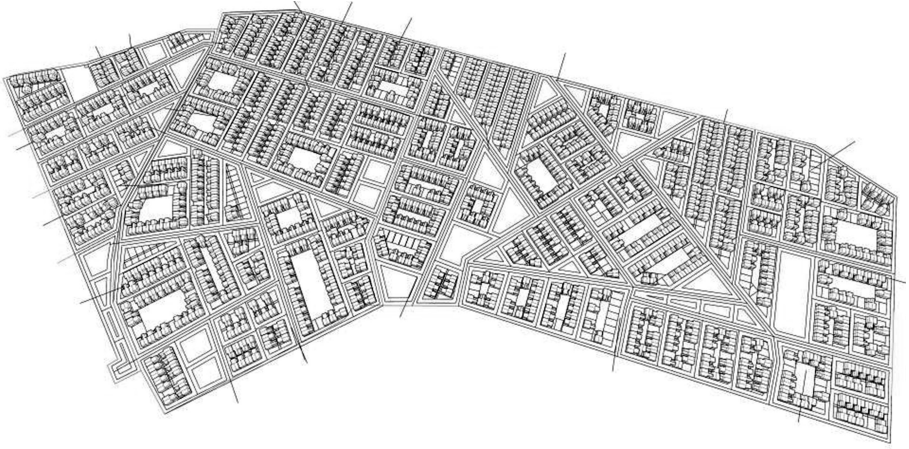


**Fig. 13.** Clustering of placed buildings. The method allows the designer to allocate residents according to their neighborhood preferences.

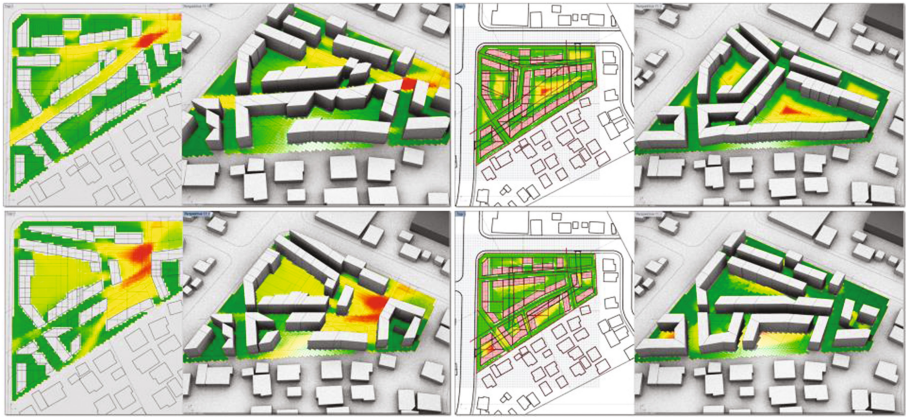
Although the buildings are placed appropriately based on the criteria introduced so far, limitations persist in respect to a meaningful distribution of open public spaces across the site. At the current state, some of the generated blocks are randomly taken out and exempted from the plots generation (Fig. 14). This aspect should be given serious attention in the further investigation in order to generate open public spaces with higher qualities.

For a later optimization of other aspects, we used the generated urban fabric as input for various simulations that measure performances of the generated spatial configurations. Whereas the optimizations methods are not presented in this paper, we consider possible performance measures. We use an Isovist field analysis as shown in Fig. 15 to derive perceived security (measured by the Isovist properties compactness, occlusivity or control), potentials for community spaces (by the Isovist property area), and potentials for pedestrian traffic (measured by the Isovist property view-through). These performance measures shall be used in a later stage of this research for optimizing them at certain locations. For example, we can maximize the maximum area





**Fig. 14.** The larger urban scenario test. The definition incorporates unique building prototypes defined by the designers and they are being placed on the site according to the generated street and plots layout.



**Fig. 15.** Four variations for a synthesized neighborhood in Cape Town. The plots are limited to a defined size. The colors in the layouts show an Isovist field analysis. In this case the area that can be seen from one location is encoded by the colors (red maximum area, green minimum area).

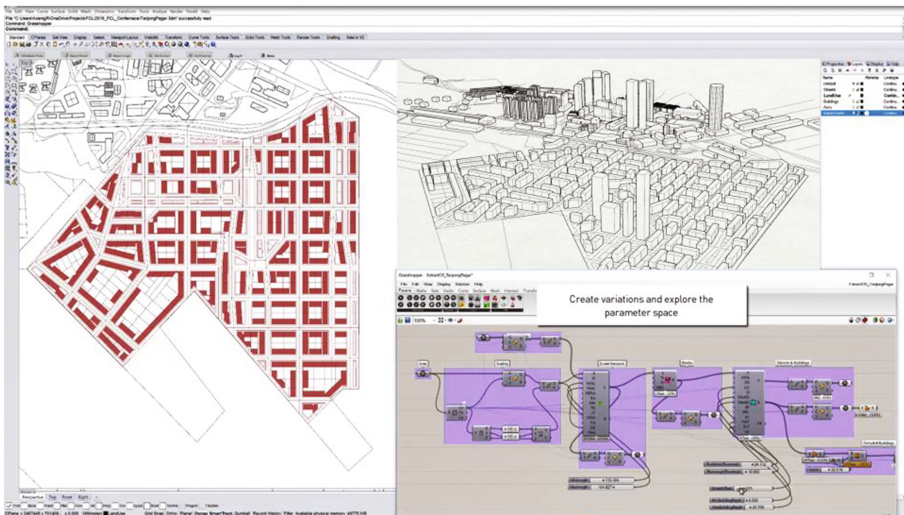
values for creating bigger community spaces. Alternatively, we may maximize the average compactness values in order to increase perceived security. This approach has been exemplified by Schneider and Koenig [28].

The second case study regards the redevelopment of the Singapore container terminal in Tanjong Pagar, which will be relocated by 2027 freeing 325ha of land for a high-profile waterfront development [33]. In collaboration with urban designers from

the Grand Projet and other researchers at the Future Cities Laboratory at the Singapore ETH Center, our computational method is applied to support the design of a sustainable high-density, mixed-use urban waterfront development. In particular, the methods presented in this paper are applied to test and explore different urban design concepts and development strategies for the site, as they allow generating urban fabric prototypes quickly and to easily explore and evaluate design alternatives. There is a link to a video provided at the end of this article, which shows the generative process.

The functionalities of our components are further developed and adapted in this ongoing case study to better respond to the requirements of the collaborating design team. An example and proof of concept, which was generated based on a given initial design concept, can be seen in Fig. 16. Design requirements implemented in the prototype includes the observation of existing street axis and connections to the existing Tanjong Pagar area as well as the Core Business District (CBD) in Singapore. Accordingly, we started the generative process with a set of initial street segments, which represented the access points to the site. The good selection of the position of the initial street also guarantees the generation of relatively regular and realistic plots, as can be seen in Fig. 16. The further generation of blocks, parcels and buildings followed the nested approach describe in Sect. 4 to create a hierarchical street pattern with primary and secondary streets.

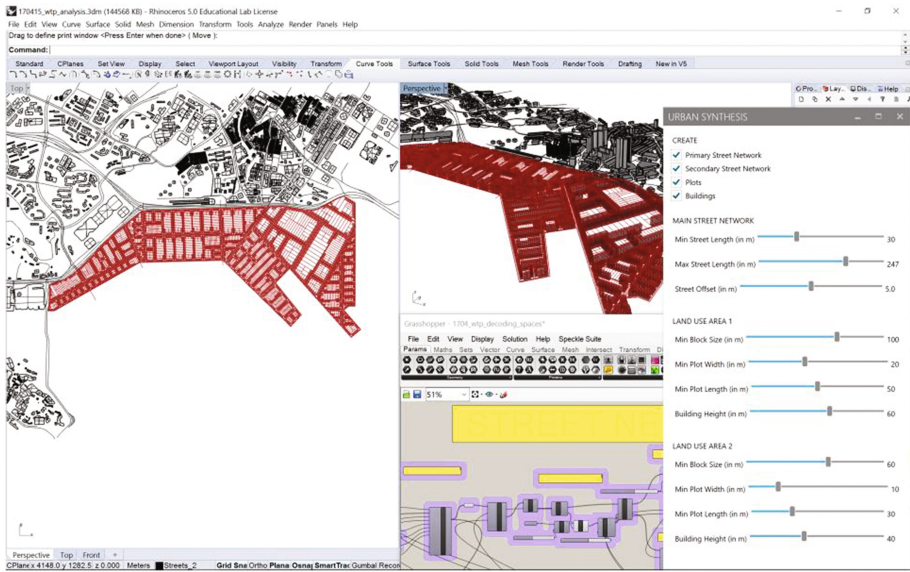
Furthermore, we subdivided the site computationally into two areas of different characteristics in correspondence to the definition of different land use areas in the



**Fig. 16.** Example for a fast urban fabric prototyping for the Tanjong Pagar container terminal in Singapore. The data of the environment including the 3D information of the buildings are imported from open street maps via the Grasshopper plugin Elk2. For generating the urban fabric, we used the components introduced in Sect. 4. For more details on the generative process, see also the video-links listed in the acknowledgement.

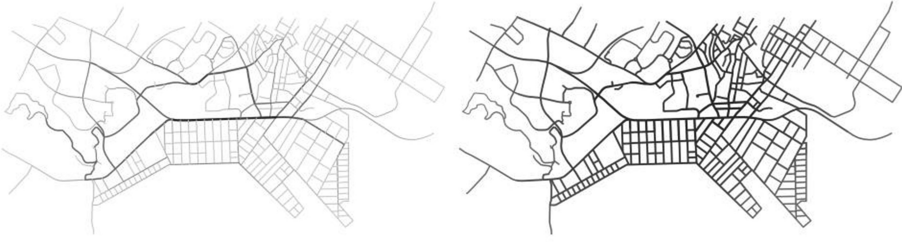
design brief. To exemplify different land uses, different minimum block widths, parcel widths and building heights were defined, with larger blocks and higher buildings in the area towards the CBD and smaller blocks and lower buildings in the area towards the waterfront (Fig. 17).

For the performance evaluation of the generated street patterns, we used the DecodingSpaces Grasshopper components for spatial analysis, which allows to cal-



**Fig. 17.** Exemplary definition of different land use areas for the Tanjong Pagar site with larger blocks and parcels as well as higher buildings towards the CBD area and smaller blocks and parcels as well as lower buildings towards the waterfront.

culate and evaluate centrality measures and thus evaluate the integration and accessibility of the generated urban area [27, 32]. Figure 18 shows the calculated global betweenness centrality of an exemplary generated urban network for the Tanjong Pagar site in relation to the existing neighborhoods in its vicinity on the left and its calculated global closeness centrality on the right. Analysis methods can be easily integrated into the design workflow, as respective components can be integrated into the generative parametric definition. Thus, they can help provide immediate feedback on the performance of a solution based on which the designer can adjust the parameters and adapt the solution accordingly.



**Fig. 18.** Global betweenness (left) and closeness (right) centrality of the generated urban network from Fig. 17 in relation to the existing neighborhood.

## 6 Conclusions and Outlook

In the scope of this paper, we presented a new urban synthesis method as well as its implementation for the procedural modeling of urban designs. Based on two exemplary case studies, we have shown that this method can produce satisfactory results [30] for urban planning tasks with different requirements. Furthermore, due to the interactive nature of our urban synthesis methods and tools, planners can benefit from exploring many possible solutions quickly.

Initial feedback obtained from collaborating designers suggests that our components can support the urban planning process in practice. Whereas our main aim with this research was to develop a relevant part for a cognitive design computing approach, which we introduced in the beginning of this paper. We may conclude that the presented methods provide a good basis for geometry synthesis. Synthesizing geometry is in turn an important pillar for cognitive design computing, since it provides the specific contribution for solving a design problem. Hence, geometric representation is the primary basis for a productive combination of human cognition and computing power. How we may use our geometry synthesis methods in a next step for optimization is described later in this section.

Towards our cognitive design computing approach our geometry synthesis implementation still comes with a few limitations. For example, the generation and desirable distribution of the plots are very much dependent on the generated layout of the street network, which itself is dependent on the defined control parameters and the placement of initial street segments. Even though the nesting of the parceling component (Fig. 9) helped to alleviate the precise plot dimension problem encountered in the Empower Shack case study, it does not solve it fundamentally. A more general issue we are facing is that the rules that generate the spatial configurations cannot be saved. As the generative algorithms use in part random values or behaviors, all parts of a generated solution are changed if a user changes a control parameter or manipulates the initial geometry. Furthermore, the user interaction capabilities of Grasshopper are limited to changes of the referenced geometry and to the manipulation of control parameter values. A more sophisticated user interaction during the generative process, which includes the direct interaction with and manipulation of generated geometries, and for later during an optimization process is not possible. The addition of such



features would require a separate user interface on top of Grasshopper or a standalone implementation as exemplified by Koenig and Schmitt [2, 31].

Moreover, for the time being, we have not integrated components for data analysis and evolutionary multi-criterion optimization (EMO) to achieve our final goal: the realization of a system based on the cognitive design computing framework as described in [2, 29]. For more advanced data analysis, the capabilities to process big data still need to be introduced into the system. One big data application could be the collection and analysis of existing urban patterns and their deployment as a library, which can be utilized for the generation and evaluation of urban layouts. For EMO, an abstract data representation has to be developed, which can parameterize all urban features for unified optimization. Existing approaches and plugins for Grasshopper like Galapagos or Octopus.E (<http://www.food4rhino.com/app/octopuse>) are limited to numerical representations but cannot compute complicated data structures as used in our approach in form of the instruction tree (Fig. 1). However, for example, spatial analysis components, which are available as Grasshopper plugins [27, 32], can be used and integrated to compute the fitness of generated urban designs.

In consequence, further investigation will address the development of a more comprehensive data structure, which will allow to represent generated urban fabrics in a way that they can be used for evolutionary optimization. In a subsequent step, we will then be looking at extending the Grasshopper Octopus.E component to work with our newly developed data structure. Finally, the combination of user-friendly interaction strategies with easy to understand problem representations and intuitively navigable solution spaces as a basis for integrated computational planning and design is an issue that is still unresolved.

**Acknowledgement.** The research presented in this paper was partially conducted at the Future Cities Laboratory at the Singapore-ETH Centre, which was established collaboratively between ETH Zurich and Singapore's National Research Foundation (FI 370074016) under its Campus for Research Excellence and Technological Enterprise program. Some methods presented in the paper were developed as part of the research project ESUM: Analyzing trade-offs between the energy and social performance of urban morphologies funded by the German Research Foundation (DFG) and the Swiss National Science Foundation (SNF, project number 100013L\_149552).

There is a demo video for the generative process illustrated in Fig. 16 at <https://vimeo.com/191807352> and <https://vimeo.com/212540621>. The process for generating the urban fabrics shown in Figs. 11, 12, 13 and 14 is shown in this video: <https://www.youtube.com/watch?v=ELUKfsx89og&feature=youtu.be>

The Rhino3D/Grasshopper components can be downloaded from the website of the Computational Planning Group: <http://cplan-group.net/>

## References

1. Rouse, M.: Cognitive computing. In: WhatIs.com. <http://whatis.techtarget.com/definition/cognitive-computing> (2014). Accessed 15 March 2016
2. Koenig, R., Schmitt, G.: Backcasting and a new way of command in computational design. In: CAADence Archit, Budapest, pp. 15–25 (2016)

3. Radford, A., Gero, J.S.: *Design by Optimization in Architecture, Building and Construction*. Van Nostrand Reinhold, New York, Wokingham (1988)
4. Cagan, J., Campbell, M.I., Finger, S., Tomiyama, T.: A framework for computational design synthesis: model and applications. *J. Comput. Inf. Sci. Eng.* **5**, 171–181 (2005). doi:[10.1115/1.2013289](https://doi.org/10.1115/1.2013289)
5. Weber, B., Müller, P., Wonka, P., Gross, M.: Interactive geometric simulation of 4D cities. In: *Eurographics*, pp. 481–492 (2009)
6. Parish, Y., Müller, P.: Procedural modeling of cities. *SIGGRAPH*, pp. 301–308. ACM, Los Angeles, CA (2001)
7. Chen, G., Esch, G., Wonka, P., et al.: Interactive procedural street modeling. *ACM Trans. Graph.* **27**, 10 (2008). doi:[10.1145/1360612.1360702](https://doi.org/10.1145/1360612.1360702)
8. Müller, P., Wonka, P., Haegler, S., et al.: Procedural modeling of buildings. In: (TOG) ACMT on G (ed) *ACM SIGGRAPH*, pp. 614–623. ACM Press, Boston (2006)
9. Mitchell, J., Steadman, P., Liggett, R.S.: Synthesis and optimization of small rectangular floor plans. *Environ. Plan. B Plan. Des.* **3**, 37–70 (1976)
10. Stiny, G., Mitchell, J.: The palladian grammar. *Environ. Plan. B Plan. Des.* **5**, 5–18 (1978)
11. Duarte, J.P.: A discursive grammar for customizing mass housing: the case of Siza’s houses at Malagueira. *Autom. Constr.* **14**, 265–275 (2005). doi:[10.1016/j.autcon.2004.07.013](https://doi.org/10.1016/j.autcon.2004.07.013)
12. Duarte, J.P., de Rocha, J.M., Soares, G.D.: Unveiling the structure of the Marrakech Medina: A shape grammar and an interpreter for generating urban form. *Artif. Intell. Eng. Des. Anal. Manuf.* **21**, 317–349 (2007)
13. Doulgerakis, A.: *Genetic Programming + Unfolding Embryology in Automated Layout Planning*. University College London (2007)
14. O’Neill, M., McDermott, J., Swafford, J.M., et al.: Evolutionary design using grammatical evolution and shape grammars: designing a shelter. *Int. J. Des. Eng.* **3**, 4–24 (2010)
15. Braach, M.: Programmieren statt Zeichnen: Kaisersrot. *archithese* (2002)
16. Braach, M.: Solutions you cannot draw. *Archit. Des.* **84**, 46–53 (2014). doi:[10.1002/ad.1807](https://doi.org/10.1002/ad.1807)
17. Coates, P., Derix, C.: The deep structure of the picturesque. *Archit. Des.* **84**, 32–37 (2014). doi:[10.1002/ad.1805](https://doi.org/10.1002/ad.1805)
18. Koenig, R.: CPlan: an open source library for computational analysis and synthesis. In: Martens, B., Wurzer, G.T.G., et al. (eds.) *Real Time—Proceedings of the 33rd eCAADe Conference on Vienna University of Technology*, Vienna, pp. 245–250 (2015)
19. Koenig, R.: CPlan Group (2015). <http://cplan-group.net/>. Accessed 28 May 2015
20. Koenig, R., Treyer, L., Schmitt, G.: Graphical smalltalk with my optimization system for urban planning tasks. In: Stouffs, R., Sariyildiz, S. (eds.) *Proceedings of the 31st eCAADe Conference—vol. 2*, Delft, Netherlands, pp. 195–203 (2013)
21. Knecht, K., Koenig, R.: Generating floor plan layouts with k-d trees and evolutionary algorithms. In: *GA2010—13th Generative Art Conference* (2010)
22. Koenig, R., Knecht, K.: Comparing two evolutionary algorithm based methods for layout generation: dense packing versus subdivision. *Artif. Intell. Eng. Des. Anal. Manuf.* **28**, 285–299 (2014). doi:[10.1017/S0890060414000237](https://doi.org/10.1017/S0890060414000237)
23. Miao, Y., Koenig, R., Bus, P., et al.: Empowering urban design prototyping: a case study in Cape Town with interactive computational synthesis methods. In: Janssen, P., Loh, P., Atomic, A., Schnabel, M.A. (eds.) *Protoc. Flows Glitches, Proceedings of the 22nd International Conference on Association Computers in Architecture Design and Research in Asia*, Hong Kong, pp. 407–416 (2017)
24. Knecht, K., Koenig, R.: *Automatische Grundstücksumlegung mithilfe von Unterteilungsalgorithmen und typenbasierte Generierung von Stadtstrukturen*. Bauhaus-Universität Weimar, Weimar (2012)

25. Hansen, D.L., Shneiderman, B., Smith, M.A.: *Analyzing Social Media Networks with NodeXL: Insights from a Connected World*. Morgan Kaufmann, Burlington (2011)
26. Turner, A.: From axial to road-centre lines: a new representation for space syntax and a new model of route choice for transport network analysis. *Environ. Plan. B Plan. Des.* **34**, 539–555 (2007)
27. Fuchkina, E.: *Pedestrian Movement Graph Analysis*. Weimar (2016)
28. Schneider, S., Koenig, R.: Exploring the generative potential of isovist fields: the evolutionary generation of urban layouts based on isovist field properties. In: *30th International Conference on Education Research in Computer Aided Architectural Design in Europe*, Prague, pp. 355–363 (2012)
29. König, R., Schmitt, G., Standfest, M.: Cognitive computing for urban design. *Virtual real plan. Urban Des. Perspect. Pract. Appl.* (2017)
30. Simon, H.A.: *The Sciences of the Artificial*, 3rd edn. MIT Press, Cambridge (1969)
31. Koenig, R.: Urban design synthesis for building layouts urban design synthesis for building layouts based on evolutionary many-criteria optimization. *Int. J. Archit. Comput.* **13**, 257–270 (2015). doi:[10.1260/1478-0771.13.3-4.257](https://doi.org/10.1260/1478-0771.13.3-4.257)
32. Bielik, M., Schneider, S., Koenig, R.: Parametric urban patterns: exploring and integrating graph-based spatial properties in parametric urban modelling. In: *eCAADe 2012* (2012)
33. URA: Master Plan Greater Southern Waterfront (2014). <https://www.ura.gov.sg/uol/master-plan/View-Master-Plan/master-plan-2014/master-plan/Regional-highlights/central-area/central-area/Greater-southern-waterfront.aspx>. Accessed 15 Feb 2017

Computer-Aided Architectural Design. Future  
Trajectories

17th International Conference, CAAD Futures 2017,  
Istanbul, Turkey, July 12-14, 2017, Selected Papers  
Çağdaş, G.; Özkar, M.; Gül, L.F.; Gürer, E. (Eds.)  
2017, XIII, 413 p. 252 illus., Softcover  
ISBN: 978-981-10-5196-8