Steps towards AI augmented parametric modeling systems for supporting design exploration

Varvara Toulkeridou[1]
[1]*Carnegie Mellon University*
[1]*vatou@cmu.edu*

Dataflow parametric modeling environments have become popular as exploratory tools due to them allowing the variational exploration of a design by controlling the parameters of its parametric model schema. However, the nature of these systems requires designers to prematurely commit to a structure and hierarchy of geometric relationships, which makes them inflexible when it comes to design exploration that requires topological changes to the parametric modeling graph. This paper is a first step towards augmenting parametric modeling systems via the use of machine learning for assisting the user towards topological exploration. In particular, this paper describes an approach where Long Short-Term Memory recurrent neural networks, trained on a data set of parametric modeling graphs, are used as generative systems for suggesting alternative dataflow graph paths to the parametric model under development.

Keywords: *design exploration, visual programming, machine learning*

INTRODUCTION

Exploring multiple design alternatives is an integral part of the early design process. Dataflow parametric modeling environments have gained a general appreciation as exploratory tools due to them allowing variations of a design to be explored by adjusting the parameters of its parametric model schema. The parametric modeling environments that extend our CAD tools are employing the single state model of interaction (Terry and Mynatt 2005), therefore typically these design variations are being accessed sequentially.

To streamline the sampling of the design space defined by a parametric model schema and support the comparison of these design alternatives in parallel, both industry and academia have been exploring parameter based interfaces (Mohiuddin et al. 2017, Tomasetti 2017, Autodesk 2018). These interfaces extend such parametric modeling systems and provide automatic mechanisms for generating and visually juxtaposing design variations. Via these mechanisms searching into the design space defined by the parametric model schema and evaluating the variational alternatives becomes more accessible.

Even though such parametric modeling systems facilitate variational exploration, they are not flexible when it comes to working on a design alternative that involves changing the parametric model structure (Chaszar and Joyce 2016). To build a parametric model, designers need to prematurely commit to a structure and hierarchy of geometric relationships and data structures, which is hard to alter during the design process. The cognitive load for making changes to the topological structure of the parametric model schema is big and in many cases, one would need to build the model from scratch (Holzer et al 2008, Turrin and Stouffs 2011). Therefore designers can easily become locked-in (Harding and Shepherd 2017) working on a specific parametric model schema and not attempt to explore different conceptual design alternatives.

The research reported here takes the view that if it is to envision our parametric modeling systems as design exploration tools, they should support designers beyond just variational exploration. Towards this direction, this research investigates how dataflow parametric modeling systems can be augmented to assist designers in considering alternative paths during the process of developing a parametric model. The strategies of *backup*, *recall* and *replay* which traditionally drive forward the designer's exploration action (Woodbury and Burrow 2006) are being revisited and reimagined via the use of machine learning models; the objective is to employ the computational system with the ability to suggest ways to the designer for expanding the design space under consideration.

In particular, Long Short-Term Memory (LSTM) neural networks are being proposed as a model capable of learning order dependence and spatial structure of parametric modeling directed acyclic graphs (DAGs). Once the network is trained, it can serve as a generative system for synthesizing new sequences of nodes based on an arbitrary input node sequence. The scope of this paper is to evaluate the potential of this approach. It is the first step towards the bigger vision of computationally supporting designers on investigating alternative parametric model schemas that could potentially perform better in terms of their set criteria and requirements.

First, exploration in the context of dataflow parametric modeling systems is defined as the transformation of the DAG's design space description; relevant precedent work on design space expansion is presented. Next, the process of training a machine learning model on a dataset of collected graphs, and using the trained model as part of a graph synthesis system for transforming the design space description of the graph is described. The potential of the machine learning model to learn structural information from previously created graphs and used as a synthesis tool is discussed. The paper concludes by highlighting future steps and improvements.

RELATED WORK

Design space is the network of the possible states a design can take. A parametric modeling graph defines a design space out of the combinations of the graph's input parameters. The geometry and numerical primitive units, the topological relationships and constraints, the manipulative functions, as well as the parameter ranges, formulate the description of the design space of the parametric model schema. Due to the structured nature of a parametric model schema, as the graph evolves, the description of its design space becomes fixed, a closed system inflexible to adapt to changing requirements.

When all variables and their relationships are defined a priori, the interaction with the design space is simply a search to determine feasible, satisfying or optimal parameter values (Gero 1994). Design, however, cannot be simply assisted by a search process; at any point, changing requirements and goals may indicate a different design space to be searched. Therefore, computationally assisted exploration needs to involve the process of determining the space within which to search, either by creating new design state spaces or by modifying existing ones (Gero 1994). Gero (1994) proposes that this can be accomplished by creating new symbols out of prior ones by way of addition, substitution or evolutionary combination.

Engineering design demonstrates research work around design space expansion via addition or substitution of new variables and features. Domain-specific heuristics are typically used to control the process of modifying the models or representations of a design (Aelion et al 1992). Cagan et al (1991), for example, use a library of predefined design space expansion techniques as the means for modifying the design topology and discovering innovative solutions. The suggested methodology assumes a knowledge representation of the design problem based on the basic, primitive propositions and assumptions. The expansion techniques are defined as mathematical operations to manipulate the knowledge representation of the problem. Optimization is used to decide which expansion technique may lead to improved designs. In a similar direction, Gero and Kumar (1993) demonstrate how feasible or improved solutions can emerge by introducing new design variables to optimization problems that due to conflicting constraints have no feasible solutions.

In the cases mentioned above, the design space description of the problem is already fixed. The methods for mutating or augmenting this design space are also a priori defined and opinionated towards specific ways of solving a particular class of problems. To make such approaches more generic, Gero and Kumar (1993) propose the use of analogical reasoning - finding precedents of similar past design cases and choosing variables from those cases to expand the design space under consideration.

Research work from the data visualization domain demonstrates relevant efforts particularly targeted to visual programming environments. Systems for creating visual programming pipelines for data exploration assist users in the process of constructing new pipelines by reusing pipeline data from a database of previously created visualization pipelines. Pipeline fragments can be used as templates to query the database, locate, and merge relevant pipelines that match certain criteria (Scheidegger et al 2007). Also, given a partial pipeline under development, sets of likely pipeline additions can be predicted by computing correspondences between existing pipeline subgraphs from the database (Koop et al 2008). These approaches show potential towards assisting users considering wider design spaces or modifying existing ones, however, they are based on matching exact precedents from the database.

Falling into the category of design space expansion via evolutionary combination, Harding and Shepherd (2017) developed an approach they call meta-parametric design. They propose automating the synthesis of directed acyclic graphs for parametric models - based on Cartesian Genetic Programming - as a way to widen the exploration of different design concepts. A graph is represented by an integer-encoded string that includes all its numeric, functional, and topological information. This string representation constitutes the genotype for an evolutionary algorithm that via mutation and crossover drives the automated generation of graphs. The limitation of this approach arises from the fact that the user needs to decide a priori what components will be used. This choice determines the geometry vocabulary available to the algorithm and therefore directs the type of results achieved. Also, the generator produces graphs of higher complexity than manually built models; due to the lack of abstraction, they are inefficient and not easily readable by humans (Joyce et al 2017).

SYNTHESIZING DATA DRIVEN COMPLETIONS

In a parametric modeling graph, nodes represent computational functions, let's call them modules, and edges represent how data flows through the modules. More formally, a parametric modeling graph is a directed acyclic graph $G = (V, E)$ where V consists of a set of modules and E is a set of connections between modules in V. A module is an object that contains a set of input and output ports through which data flows in and out of the module. A connection between two modules v_a and v_b, connects an output port of v_a to an input port of v_b.

Given this context, the problem of deriving partial completions for the graphs can be defined as follows. Given a partial graph (subgraph) H we wish to find a set of completions that reflect the node structures that exist in a collection of completed graphs. The completions comprise a sequence of nodes that can be connected to each other in a meaningful way both syntactically and semantically. VisComplete (Koop et al 2008) proposed a solution to this problem by using exact templates from the training data to make predictions, by counting exact matches between the recent history and the data set. This paper's approach, on the other hand, involves a machine learning model that uses its internal representation to perform a high-dimensional interpolation between training examples. The objective is that the model synthesizes and reconstitutes the training data in a complex way with the possibility for node sequences to emerge that do not necessarily exist in the dataset.

Long Short-Term Memory (LSTM) recurrent neural networks are trained on a data set of previously created graphs and are used as generative systems for suggesting alternative dataflow graph paths to the parametric model under development. LSTMs are widely used for natural language processing having been proven capable of learning order dependence and spatial structure of text. This work employs methods used for training LSTM networks for text generation (Graves 2013, Sutskever et al 2011) and approaches that generalize these methods from sequences of words to graphs (Perozzi et al 2014). It adopts these methods to the case of directed acyclic graphs (DAGs), the representation used by our dataflow parametric modeling environments.

The prototype system consists of three components a) the data parser component, which converts the graphs from the training dataset to sequential data, b) the trainer component, which is responsible for the training process of the neural network, and c) the synthesizer component, which queries the trained model and performs topological changes, node, and link additions, to the directed acyclic graph. Figure 1 displays an overview diagram that demonstrates the process involved in each component. They will be described in more detail in the following sections.

Parsing graphs to sequential data

Like any other CAD tool, a parametric modeling visual programming environment is a system that employs computational mechanisms to effect change. While building a parametric modeling graph, the progressive addition of nodes produces new spatial objects or changes to spatial objects by manipulative operations like arithmetic operations and geometrical operations. The sequence of spatial changes is continuous (Krishnamurti and Stuffs 1996); with every node addition, we derive a new spatial object from a given object. Therefore every manipulative operation is dependent on the history of operations and the structure and hierarchy developed so far in the graph. This fact highlights the need for learning generative models over graphs that can capture their relational structure and order dependence.

Recurrent neural networks (RNNs) are a class of such models that can be trained for sequence generation. RNNs contain cycles that feed the network activations from a previous time step as inputs to the network to influence predictions at the current time step. This adds "memory" to the network that allows it to learn the ordered nature and relationships of the sequential input data. We are using a specific type of RNN architecture, called Long Short-Term Memory (LSTM) designed to be better at learning long-range structure than standard RNNs (Hochreiter and Schmidhuber 1997).

An important challenge we need to address is designing an invertible way of converting a parametric modeling graph structure to a sequential representation (linearization); this way we can treat sequences as sentences in natural languages and use the LSTM language models and methodology. There is a significant amount of work from the natural language processing and program synthesis community. For example, Vinyals et al. (2014) flattened a tree into a sequence following a depth-first traversal order and then modeled parse tree generation as a sequence to sequence task. Other efforts use more domain-specific linear representations as fingerprints to represent graphs (Gomez-Bombarelli et al 2018). For this first iteration of our experiment, we parse the graph into all possible linear paths of nodes and we model the problem as a sequence to prediction task. Given a corpus of examples of linear paths

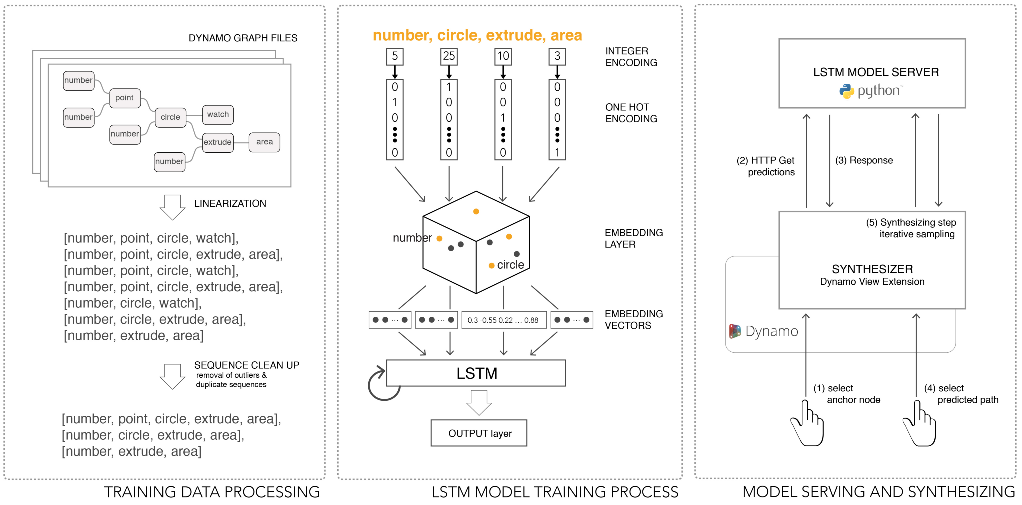
DYNAMO GRAPH FILES
number
point
number
circle
watch
number
extrude
area
number
LINEARIZATION
[number, point, circle, watch],
[number, point, circle, extrude, area],
[number, point, circle, watch],
[number, point, circle, extrude, area],
[number, circle, watch],
[number, circle, extrude, area],
[number, extrude, area]
SEQUENCE CLEAN UP
removal of outliers & duplicate sequences
[number, point, circle, extrude, area],
[number, circle, extrude, area],
[number, extrude, area]
TRAINING DATA PROCESSING
number, circle, extrude, area
5
25
10
3
INTEGER ENCODING
ONE HOT ENCODING
number
circle
EMBEDDING LAYER
0.3 -0.55 0.22 ... 0.88
EMBEDDING VECTORS
LSTM
OUTPUT layer
LSTM MODEL TRAINING PROCESS
LSTM MODEL SERVER
python
(2) HTTP Get predictions
(3) Response
(5) Synthesizing step iterative sampling
SYNTHESIZER
Dynamo View Extension
Dynamo
(1) select anchor node
(4) select predicted path
MODEL SERVING AND SYNTHESIZING

Figure 1
Overview diagram.

extracted from a data set of existing parametric modeling graphs, generate new sequences of nodes that have the structural properties of the corpus. Our selected approach for how to synthesize back the predicted sequences into the graph structure will be explained in the following section.

Model architecture and training

An LSTM neural network is a kind of recurrent neural network with a recurrent, hidden layer of memory blocks (Hochreiter and Schmidhuber 1997). Each memory block contains several self-recurrent linear memory cells. The self-recurrence on each cell enables it to accumulate numerical values over a series of iterations of the network. The accumulated data is passed through a nonlinear function. Each block contains three gated sigmoid units and can regulate the flow of information. These gates can learn which data in a sequence is important to keep or discard; this way the relevant information is being passed down the long chain of sequences to make predictions. The complete equations for the LSTM network are beyond the scope of this paper.

The network consumes the sequence of nodes in a left-to-right sweep, creating one-hot encoded vectors in memory; these vectors are high-dimensional and sparse. In our training set, we have 600 unique nodes; this means that, when using one-hot encoding, each node will be represented by a vector containing 600 integers. In order to do this computationally more efficient and to have the network learn similarities between nodes, we use an embedding layer (Mikolov et al 2013). This allows us to capture relationships in the graph structure that are very difficult to capture otherwise. The use of the embedding layer helps the network learn a representation of nodes in which nodes that tend to appear adjacent to each other are closer in the vector space.

To summarize, the network architecture comprises 3 layers: a) An embedding layer that maps the discrete representation of a node, i.e. the one-hot encoded vector, to a semantic representation, b) This semantic representation is then fed to an LSTM layer which generates a state based on the previous state and current input. This combination of the states provides the context to our model, and c) A layer that maps this generated state to a set of probabilities. We train two separate models, one feeding the input sequences of nodes in a left-to-right sweep and one feeding them backward; this way we can retrieve predictions for downstream and upstream nodes respectively.

Generating suggestions

During graph derivation, the synthesizer component adds new structure to the existing graph, specifically a new node, and the probability of that addition event depends on the history of the graph derivation. We train the model as a predictive model, but we use it as a generative model to generate entirely new plausible sequences of nodes. The predictions are probabilistic; novel sequences can be generated from the trained network by iteratively sampling from the network's output distribution, then feeding in the sample as input at the next step. Although the network itself is deterministic, the stochasticity injected by picking samples induces a distribution over sequences. Since the internal state of the network depends on the previous inputs, the final distribution obtained is conditioned upon the previous inputs as well.

Each synthesizing step generates a downstream sequence of three nodes by querying the model trained on forward sequences. In each iteration, a new downstream node is added and linked to the anchor node. If there are available input ports on the newly added node, then the system queries the model trained on the backward sequences. The iteration completes when the newly added node is designated as the new anchor node (Figure 2). For every request for a downstream prediction, all linear paths that the upstream graph that leads to the anchor node can be parsed into, are taken into account. Similarly, for every upstream prediction request, all downstream graph paths are taken into account.

Let's take a closer look at how the system synthesizes the predictions for the sequences into a node prediction that reflects the graph structure. Given a parametric modeling directed acyclic graph G we want to know which next node could be a possible addition to progress the series of spatial transformations. We can express the probability of an incoming node addition as $P(n|G)$. The question is how can we express this in a computationally feasible manner. To accomplish this, we consider a simplified base case. Let's denote any of the leaf nodes of the graph (nodes that do not have any successors) as our anchor node; we are seeking the next node that is a possible addition to that anchor node. Next, we consider the upstream subgraph that leads to this anchor node and we create the set of node sequences that form all possible directed paths $P = \{p_1, p_2, \ldots p_n\}$ that lead to the anchor node. Given the set of upstream paths, the probability of a single node is the measure of how likely it is that node to show up, given the upstream paths. To compute the

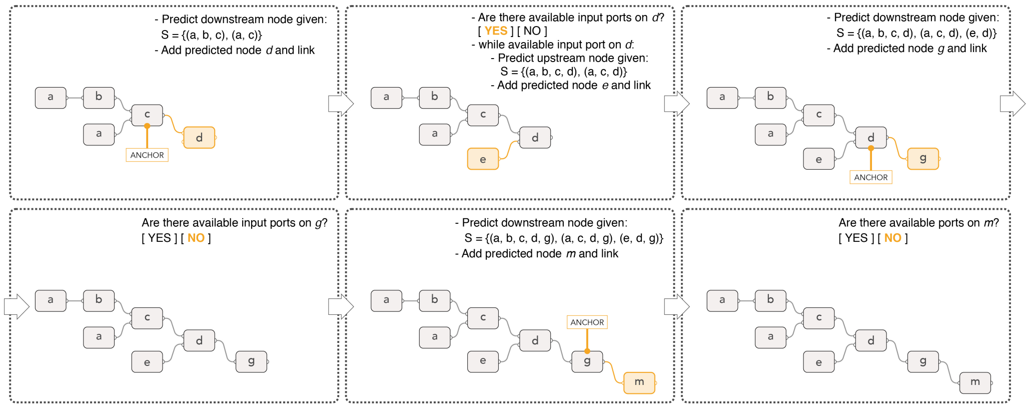
- Predict downstream node given:
S = {(a, b, c), (a, c)}
- Add predicted node d and link
ANCHOR
- Are there available input ports on d?
[YES] [NO]
- while available input port on d:
- Predict upstream node given:
S = {(a, b, c, d), (a, c, d)}
- Add predicted node e and link
- Predict downstream node given:
S = {(a, b, c, d), (a, c, d), (e, d)}
- Add predicted node g and link
ANCHOR
Are there available input ports on g?
[YES] [NO]
- Predict downstream node given:
S = {(a, b, c, d, g), (a, c, d, g), (e, d, g)}
- Add predicted node m and link
ANCHOR
Are there available ports on m?
[YES] [NO]

Figure 2
Graph derivation during a synthesizing step of 3 iterations.

likelihood of a single node we need to take into account the information given by all upstream paths. In other words, the probability of an incoming node can be expressed as the joint probability of all upstream node sequences:

$$P(n|p_1, p_2, \dots p_n) \tag{1}$$

The formula is commonly used to describe graphs in the bayesian world. For the need of this paper, we make a simplifying assumption that the upstream paths are independent of each other. This reduces our expression to:

$$P(n|p_1, p_2, \dots p_n) \propto P(n|p_1)P(n|p_2)\dots P(n|p_n) \tag{2}$$

Implementation

The data parser component is an executable that runs Autodesk's Dynamo® headless (without loading the graphical user interface) to load each Dynamo file, parse the in-memory graph data structure of each model, and extract all graph linear paths. The training data set comprises Dynamo graphs aggregated from the web; most of them include example workflows, training material and samples from the Dynamo Primer and Dynamo Dictionary repositories. The parsing process resulted in a training set of 6500 node sequences. The sequence size varies between 2 and 32 nodes.

The trainer component is developed using the Keras API running on top of TensorFlow®. As mentioned above two models are being trained, one using the extracted linear paths and one using the same paths in the reverse direction. Each training run took 22 minutes on an NVIDIA Tesla K80 GPU processor.

The synthesizer component consists of two parts: the server, that serves the two trained models and the synthesizer client. The server listens for prediction requests from the client, queries the trained models and returns a response to the client. The body of a request contains the set paths for which a downstream or upstream node is requested. The body of the response contains a mapping of each node to the combined probability given the set of paths. The synthesizer client is an add-on to Autodesk's Dynamo® developed as a View Extension, which sends requests for downstream or upstream node predictions to the model server. The synthesizer client provides the ability to the user to designate a node as the anchor node for the synthesizing step. The moment the user selects an anchor node the system automatically returns a list of possible paths and populates the predictions list on the synthesizer client window. When the user selects a path, a synthesizing step takes place and the generated nodes and links are added to the Dynamo editor by the synthesizer client.

INITIAL RESULTS

The described work is a first step towards the bigger objective of augmenting parametric modeling systems towards design exploration. The goal was to evaluate whether the proposed approach and in particular, describing parametric modeling graph synthesis as a sequence prediction problem, is plausible. Initial tests using the synthesizer system with the trained models have been positive, although there is still much work to be done to understand the potential use of this method and its applications. We believe that this work can inspire the application of machine learning to research related to design space expansion and content creation in the context of our parametric modeling environment.

Overall, the model has demonstrated successful results in learning order dependence and graph structure. In most of the cases, the predictions are semantically and syntactically meaningful. Note that the focus at this point is not to evaluate whether we can get complete recommendations for design ideas. This is meant for future work and requires a system capable of goal-oriented predictions and sampling. Also, the quality of the predictions is largely dependent on the training dataset; a curated training graph data set of bigger complexity and theme consistency, for example, is expected to yield better results.

The examples in Figure 3 demonstrate the be-

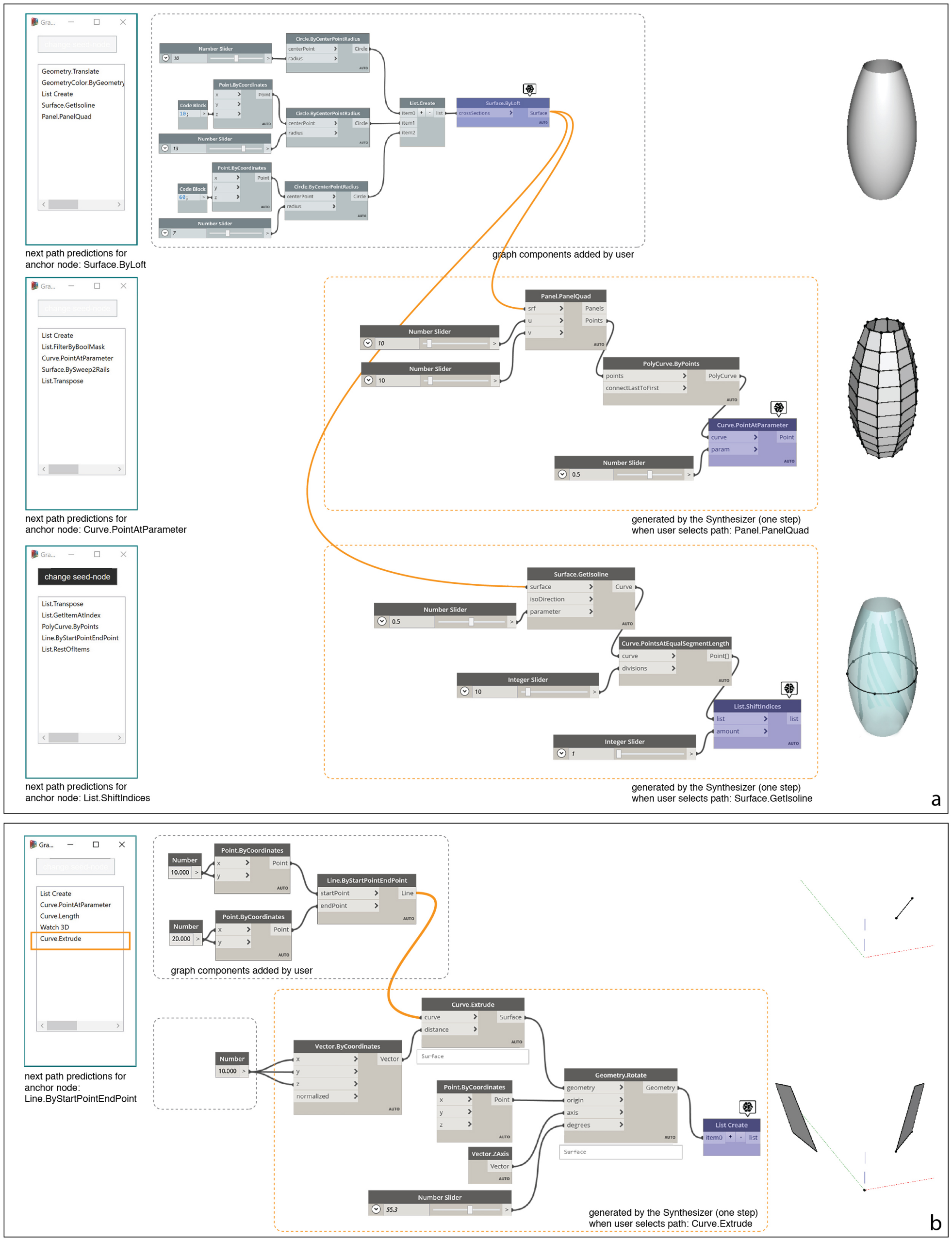
Gra...
Geometry.Translate
GeometryColor.ByGeometry
List Create
Surface.GetIsoline
Panel.PanelQuad
next path predictions for anchor node: Surface.ByLoft
Number Slider
Circle.ByCenterPointRadius
Point.ByCoordinates
Code Block
List.Create
Surface.ByLoft
graph components added by user
List Create
List.FilterByBoolMask
Curve.PointAtParameter
Surface.BySweep2Rails
List.Transpose
next path predictions for anchor node: Curve.PointAtParameter
Panel.PanelQuad
PolyCurve.ByPoints
Curve.PointAtParameter
generated by the Synthesizer (one step) when user selects path: Panel.PanelQuad
change seed-node
List.Transpose
List.GetItemAtIndex
PolyCurve.ByPoints
Line.ByStartPointEndPoint
List.RestOfItems
next path predictions for anchor node: List.ShiftIndices
Surface.GetIsoline
Curve.PointsAtEqualSegmentLength
Integer Slider
List.ShiftIndices
generated by the Synthesizer (one step) when user selects path: Surface.GetIsoline
a
List Create
Curve.PointAtParameter
Curve.Length
Watch 3D
Curve.Extrude
next path predictions for anchor node: Line.ByStartPointEndPoint
Number
Point.ByCoordinates
Line.ByStartPointEndPoint
graph components added by user
Curve.Extrude
Vector.ByCoordinates
Geometry.Rotate
Vector.ZAxis
List Create
generated by the Synthesizer (one step) when user selects path: Curve.Extrude
b

Figure 3
Examples of a synthesizing step. On the left of each graph, the synthesizer's client window displays a list of possible paths given the designated anchor (node in purple). The values of automatically added input nodes (numbers and sliders) were set by the author after the synthesizing step.

havior of the system. Each example shows a synthesizing step. For each synthesizing step, the synthesizer client performs 3 iterations. In each iteration, the forward model is being queried for a downstream node prediction and the backward model for an input node prediction if the predicted downstream node has input ports available. Example *a* shows two alternative paths for the same upstream partial graph and designated anchor node. Example *b* showcases the ability of the system to synthesize data in complex ways that do not necessarily show up in the training dataset. The system links *Curve.Extrude* to the existing *Line.ByStartPointEndPoint*; this is a novel sequence that the system has not been trained on and is a meaningful sequence both semantically and syntactically.

Figure 4 demonstrates the system being successful in learning order dependence and context. The transformative operations in the two graphs are the same, but while the upper graph manipulates a single generated rectangle, the lower one manipulates a list of rectangles. The returned predictions, given *Curve.ExtrudeAsSolid* as the anchor node, show that the system has learned to identify such a contextual difference. In the case of the list of rectangles, the system suggests mostly downstream nodes that correspond to list operations.

FUTURE WORK

There is much potential for development and improvements. These include the following:

- Linearizing the graph to node sequences requires the additional work of merging the predictions for the different paths back together to form the graph structure. Ideally, we would prefer a linear representation that reflects the graph structure as a whole. Other linearization approaches from the machine learning literature, like depth-first traversal order (Vinyals et al 2014) or topological sorting of the graph (Li et al 2018) are worth investigating further.
- A training sequence is composed of integer identifiers corresponding to the node names. The model learns node to node dependency but no output to input port dependency. Therefore the synthesizer component adds links between existing and predicted nodes by trial and error. Incorporating information about input and output ports in the training sequence can improve the syntactic and semantic consistency of the predictions and improve the synthesizing process.
- Synthesis occurs at the node level; this makes it more challenging to lead the generation towards an objective. Experimenting with different degrees of decomposition of the graphs and a more modular approach could be beneficial. For example, each data point in the input sequence could maintain more functionality, i.e. a set of nodes that are semantically related. This way, less potential

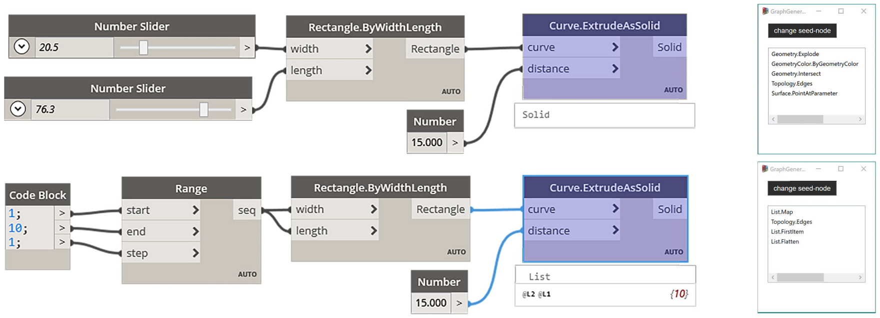

Figure 4
Predicted downstream nodes for the same anchor node given different upstream context.

for variation is afforded, however, less work is required for correct semantic and syntactic matching.

REFERENCES

Aelion, V, Cagan, J and Powers, GJ 1991 'Input Variable Expansion: a formal innovative design generation technique', *Technical Report*, pp. 1-20

Autodesk, Inc 2018 'Refinery Project', *www.autodesk.com/solutions/refinery-beta*

Cagan, J and Agogino, AM 1991, 'Dimensional Variable Expansion: A formal approach to innovative design', *Research in Engineering Design*, 3(2), pp. 75-85

Chaszar, A and Joyce, SC 2016, 'Generating freedom: Questions of flexibility in digital design and architectural computation', *International Journal of Architectural Computing*, 14(2), pp. 167-181

Gero, JS 1994 'Towards a model of exploration in computer-aided design', *Workshop on Formal Design Methods for CAD*, pp. 315-336

Gero, JS and Kumar, B 1993, 'Expanding design spaces through new design variables', *Design Studies*, 14(2), pp. 210-221

Gomez-Bombarelli, R, Wei, JN, Duvenaud, D, Hernandez-Lobato, JM, Sanchez-Lengeling, B, Sheberla, D, Aguilera-Iparraguirre, J, Hirzel, TD, Adams, RP and Aspuru-Guzik, A 2018, 'Automatic chemical design using a data-driven continuous representation of molecules', *ACS Central Science*, 4(2), pp. 268-276

Graves, A 2014 'Generating sequences with Recurrent Neural Networks', *arXiv preprint arXiv:1308.0850v5*, pp. 1-43

Harding, JE and Shepherd, P 2017, 'Meta-Parametric Design', *Design Studies*, 52, pp. 73-95

Hochreiter, S and Schmidhuber, J 1997, 'Long Short-Term Memory', *Neural Computation*, 9(8), pp. 1735-1780

Holzer, D, Hough, R and Burry, M 2016, 'Parametric design and structural optimisation for early design exploration', *International Journal of Architectural Computing*, 5(4), pp. 625-643

Joyce, SC and Nazim, I 2017 'Exploring the evolution of meta parametric models', *37th Annual Conference of the Association for Computer Aided Design in Architecture*, pp. 308-317

Koop, D, Scheidegger, CE, Callahan, SP, Freire, J and Silva, CT 2008, 'VisComplete: Automating suggestions for visualization pipelines', *IEEE Transactions on Visualization and Computer Graphics*, 14(6), pp. 1691-1698

Krishnamurti, R and Stouffs, R 1997, 'Spatial change: continuity, reversibility, and emergent shapes', *Environment and Planning B: Planning and Design*, 24(3), pp. 359-384

Li, Y, Vinyals, O, Dyer, C, Pascanu, R and Battaglia, P 2018 'Learning deep generative models of graphs', *arXiv preprint arXiv:1803.03324v1*, pp. 1-22

Mikolov, T, Corrado, G, Chen, K and Dean, J 2013 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781v3*, pp. 1-12

Mohiuddin, A, Woodbury, R, Ashtari, N, Cichy, M and Mueller, V 2017 'A Design Gallery system: Prototype and evaluation', *37th Annual Conference of the Association for Computer Aided Design in Architecture*, pp. 414-245

Perozzi, B, Al-Rfou, R and Skiena, S 2014 'DeepWalk: Online learning of social representations', *the 20th ACM SIGKDD international conference*, New York, New York, USA, pp. 701-710

Scheidegger, C, Vo, H, Koop, D, Freire, J and Silva, C 2017, 'Querying and creating visualizations by analogy', *IEEE Transactions on Visualization and Computer Graphics*, 13(6), pp. 1560-1567

Sutskever, I, Vinyals, O and Le, QV 2014 'Sequence to sequence learning with neural networks', *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Montreal, Canada, pp. 1-9

Terry, M, Mynatt, ED, Nakakoji, K and Yamamoto, Y 2004 'Variation in element and action: Supporting simultaneous development of alternative solutions', *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, New York, USA, pp. 711-718

Thornton Tomasetti, C. S. 2017 'Design explorer', *http://tt-acm.github.io/DesignExplorer/*

Turrin, M, von Buelow, P and Stouffs, R 2011, 'Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms', *Advanced Engineering Informatics*, 25(4), pp. 656-675

Vinyals, O, Kaiser, L, Koo, T, Petrov, S, Sutskever, I and Hinton, G 2015 'Grammar as a foreign language', *Proceedings of the 28th International Conference on Neural Information Processing Systems*, Montreal, Canada, pp. 2773-2781

Woodbury, R and Burrow, A 2006, 'Whither design space?', *AI EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(02), p. 63–82