Learning to Simulate and Design for Structural Engineering

Kai-Hung Chang [1] **Chin-Yi Cheng** [1]

1. Data Collection

This section describes the data collection process in detail. All unit abbreviations are listed in Table 1. We adopt the same beam spans, materials, cross-sections, and load cases used by a structural design company.

1.1. Skeleton Creation

Building skeleton are created by a fixed sampling algorithm due to the deficiency of real-world data. Each building is erected on a rectangular base which edges are sampled between 60 ft to 400 ft. A grid is created on the base and the intervals are sampled from the set of beam spans, ranging from 28 ft to 40 ft. A connected layout is sampled from the grid using depth-first-search algorithm which expands to neighboring grid cells with 0.5 probability. The same layout is vertically stacked up to 10 stories to form a voxel-like building geometry. Each voxel contains four columns on four vertical sides and four beams which form a rectangle frame on the top to support the floor panel. The story height is fixed at 16 ft.

1.2. Structural Simulation Model in RSA

Given the geometry of the building structure, we can create the corresponding structural simulation model in Autodesk Robot Structural Analysis (RSA), which is an industrial structural simulation software. All the columns on the first story are not pinned, but fixed to the ground. Materials for columns and beams are Steel A500-46 and Steel A992-50 respectively. For each column and beam, the cross-section is randomly assigned from Table 2. 150 pcf Concrete floor panels are modeled as slabs on trapezoid plates with other parameters given in Table 3. The definition of the symbols can be found in this link. In the graph representation, we do not model joists (smaller beams arranged in parallel across two beams to support floor panels). Instead, each surface load is converted to concentrated loads at joist endpoints.

[1] Autodesk Research, San Francisco, California, United States. Correspondence to: Kai-Hung Chang <kai-hung.chang@autodesk.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

Table 1. Unit Abbreviation

Abbreviation	Full Unit
ft	Foot
pcf	Pound per cubic foot
psf	Pound per square foot

Table 2. Cross-Section Library

Column	Beam
HSSQ 16x16x0.375	W 21 x 44
HSSQ 16x16x0.5	W 21 x 48
HSSQ 16x16x0.625	W 21 x 50
HSSQ 16x16x0.75	W 21 x 57
HSSQ 16x16x0.875	W 21 x 62
	W 21 x 68
	W 21 x 73
	W 21 x 83
	W 21 x 93

Table 3. Floor Panel Specification

Parameter Name	Value
h	6.3 in
h1	2.56 in
a	7.4 in
a1	1.73 in
a2	4.96 in
Th	7.46 in
Th 1	8.86 in
Th 2	6.3 in
Joist Direction	Parallel to the shorter edge
Material	Concrete
Material Resistance	3.5 ksi
Material Unit Weight	0.15 kip/ft3
Diaphragm	Rigid
Load Transfer	Simplified one way
Finite Element	None

For each floor panel, three joists are placed across the longer edges.

1.3. Load Cases Setup

IBC 2000 is the building code used for structural simulation. Below lists all the load cases:

1. Self-Weight: This is the self-weight load acting in the gravitational direction for all structure elements. The

coefficient is set to 1.1.

2. Super-Imposed Dead Load: Super-imposed dead load accounts for the static weight of the non-structure elements. Here we add 24 psf surface loads to all floor panels except the roof.

3. Live Load: Live load refers to the load that may change over time, such as people walking around. We consider 100 psf surface loasd on all floor panels except the roof.

4. Roof Live Load: Roof live load is set as 20 psf surface load, different from the live load on other stories.

5. Roof Dead Load: We assign 15 psf surface load for non-structure elements on roof panels.

6. Cladding Load: 20 psf $\times$ 16 ft (story height) $+ 90$ lb/ft $= 410$ lb/ft line load is added to all boundary beams on each story for self-weights of cladding walls.

7. Modal Analysis: Modal analysis determines eigenvalues (eigenpulsations, eigenfrequencies, or eigenperiods), precision, eigenvectors, participation coefficients and participation masses for the problem of structural eigenvibrations. The number of modes is set to 30.

8. Seismic X: Seismic loads are automatically computed by RSA given the building code. We consider seismic loads in two directions: X and Y. Settings of seismic loads are listed in Table 4. Seismic X refers to the seismic loads in direction X.

9. Seismic Y: This is the seismic loads in direction Y.

10. Static Load Combination: Load combination linearly combines multiple load cases. Static load combination is defined as $1.2D + 1.6L + 0.5L_r$, where D is the sum of dead loads (1+2+5+6), L is the live load (3),and L_r is the roof live loads (4).

11. Seismic Load Combination X: Complete quadratic combination (CQC) method is used for seismic load combination. This is defined as $0.9D + 1.0E_x$, where E_x is the Seismic X load (8).

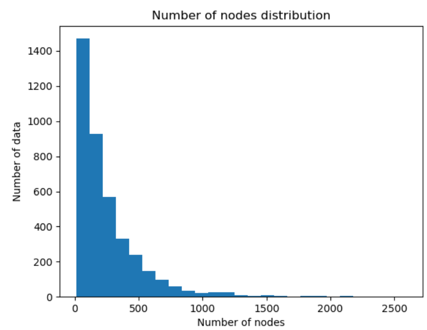
12. Seismic Load Combination Y: This is defined as $0.9D + 1.0E_y$, where E_y is the Seismic Y load (9).

1.4. Saved Results

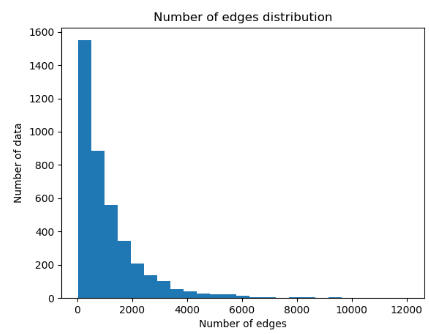
After running the structural simulation, we save all drift ratios in direction X and Y for Seismic Load Combination X and Y load cases respectively. The components of the drift ratios perpendicular to the seismic load directions are relatively small compared to the drift ratio limit. The drift ratio distribution is normalized to $[-1, 1]$. The open-source data set is available under https://github.com/AutodeskAILab/LSDSE-Dataset.

Table 4. Seismic Parameters

Parameter Name	Value
Site Class	D
S_1 (Acceleration parameter for 1-second period)	0.6
S_s (Acceleration parameter for short periods.)	1.8
Ie (Importance factor)	0.0
Load to mass conversion for dead load	1.0
Load to mass conversion for live load	0.1
Load to mass conversion for roof live load	0.25



(a) Number of nodes distribution.



(b) Number of edges distribution.

Figure 1. Statistics of 4000 collected structural graphs.

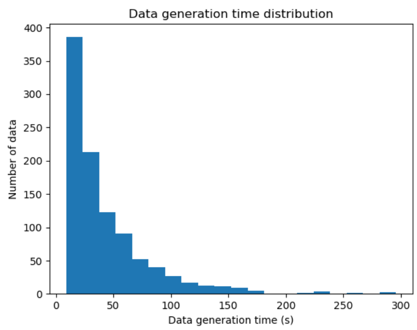
1.5. Statistics

Figure 1 shows statistics of the collected 4000 building structural graphs. Figure 2 plots the two histograms: one for the times to generate one datum and the other for the calculation times spent on solving each structural simulation.

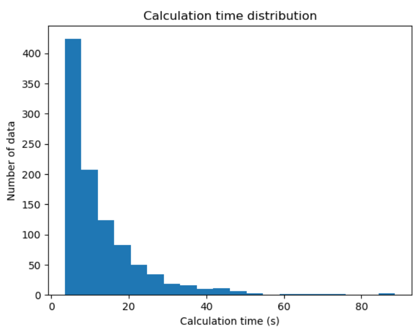
2. Models Details

2.1. NeuralSim

The input node feature of bar i in a structural graph is denoted as $v_i \in \mathbb{R}^{19}$. A single layer perceptron (SLP) encoder



(a) Data collection time distribution.



(b) Calculation time distribution for solving structural simulations.

Figure 2. Statistics of 4000 collected structural graphs.

first maps each node feature to embedding $v_i^0 \in \mathbb{R}^{512}$.

$$v_i^0 = SLP_{encoder}(v_i) \tag{1}$$

To perform message-passing in the propagation step, we first compute aggregated messages then update each node feature. The superscript t denotes the propagation step and $Ne(i)$ denotes the neighbor set of node i.

$$m_i^t = Mean\{SLP_{message}(v_i^t, v_j^t) | j \in Ne(i)\} \tag{2}$$

$$\begin{aligned} mp_i^t &= Mean\{SLP_{p\_message}([v_i^t, \frac{1}{d(i,j)+1} v_j^t]), \\ j &= \arg\max_l \{d(l,i) | l \in A_s\} \forall A_s, s = 1 \dots S\} \end{aligned} \tag{3}$$

$$v_i^{t+1} = SLP_{update}(v_i^t, mp_i^t, m_i^t) \tag{4}$$

Equation 2 computes the aggregated message m_i from neighbor nodes while Equation 3 computes the position-aware message mp_i from the set of 512 anchor nodes $\{A_s | s = 1 \dots S\}$. $d(l, i)$ represents the geodesic distance between node l and i. For more detail about position-aware message, we refer the readers to the (You et al., 2019). Equation 4 updates each node embedding based on the current embedding and the messages. If position-aware message is not used, we drop mp_t^t and change the input dimension of SLP_{update} accordingly. In the end, we apply dropout function with 0.5 probability before the next propagation. In total, we run $T = 5$ propagation steps.

Since NeuralSim is generating per-story output, we compute the story embedding o_k by average pooling all the embeddings in story k.

$$o_k = AvgPool(\{v_i^T\} | i \in \text{Story } k\}) \tag{5}$$

$$o_k \leftarrow SLP_{recursive}\{[o_k, o_{k+1}]\} \text{ for } k = K - 1 \dots 1 \tag{6}$$

Structured Decoder is processed using Equation 6, where each story embedding is updated in the top-down order. In the end, the story embeddings are passed to two multi-layer perceptron (MLP) decoders: one predicts the drift ratios $h_k \in \mathbb{R}^2$ and the other classifies if the ground-truth drift ratios exceed the drift ratio limit $lim = 0.015$ or not.

$$h_k = MLP_{decoder}(o_k) \tag{7}$$

$$c_k = SigmoidMLP_{decoder}(o_k) \tag{8}$$

The multi-task loss is constructed by adding the L1 loss and the binary cross-entropy (BCE) loss.

$$\text{Loss} = \frac{1}{K} \sum_{k=1}^{K} |h_k - \hat{h}_k| - w \text{ BCE}(c_k, \hat{c}_k) \tag{9}$$

, where $\hat{h}_k$ is the ground-truth drift ratio , $\hat{c}_k$ is 1 if $\hat{h}_k > lim$, otherwise is 0, and $w = 1$ is the weight balancing the two losses. NeuralSim is trained with 5 epochs, batch size 1, and learning rate 1e-4 using the Adam optimizer.

2.2. NeuralSizer

The inputs of the NeuralSizer are building skeleton geometries, which are represented as the same structural graphs except that the input node features $v_i \in \mathbb{R}^{10}$ now do not contain cross-sections. The encoder and propagation steps are the same as NeuralSim. Note that NeuralSizer does not compute nor use position-aware message by virtue of a faster training time. After 5 steps of propagation, the graph embedding g is computed by MaxPooling all the node embeddings as below.

$$g = MaxPooling(v_i^T | \forall i) \tag{10}$$

Each node embedding together with the graph embedding is fed into an MLP decoder to generate one-hot vectors $y_i \in \mathbb{R}^9$ using hard Gumbel-Softmax function. The decoder has leaky ReLU function with negative slope 0.01 and dropout function in each layer.

$$y_i = \text{GumbelSoftmax}(MLP_{decoder}(v_i^T, g)) \tag{11}$$

NeuralSizer is trained with batch size 5 and learning rate 1e-4 using Adam optimizer. 50,000 buildings are randomly sampled during training, and a fixed 500 data set is used for evaluation. The drop out probability is 0.5 and linearly decays to zero at the end of training. The loss function is given in the main paper.

3. Dual Gradient Descent

A general constrained optimization problem with an objective function $f(\ )$ and an equality constraint $g(\ )$ can be written as

$$\min f(\ ) \qquad \text{s.t. } g(\ ) = 0 \tag{12}$$

Changing the constrained optimization to the dual problem, we get the Lagrangian:

$$L(\ ;\ ) = f(\ ) \qquad g(\ ) \tag{13}$$

, where is the dual variable. Dual gradient descent alternates between optimizing the Lagrangian with respect to the primal variables to convergence, and then taking a gradient step on the dual variables. The necessity of optimizing the Lagrangian to convergence is optional under convexity. Both (Haarnoja et al., 2018) and our work found updating one gradient step still works in practice. As a result, the primal and dual variables are iteratively updated by the following equations.

$$' = \ + \ (r \ \ f(\ ) \qquad r \ \ g(\ )) \tag{14}$$
$$' = \ + \ g(\ ) \tag{15}$$

where and are learning rates. Inequality constraints can also be formulated similarly.

In this paper, our total loss is $w_0 obj + w_1 l_{dr} + w_2 l_{var} + w_3 l_H$. The initial weights and their learning rates are listed in Table 5.

Table 5. Dual Gradient Descent Parameters

Loss	Initial Weight	Learning Rate
Mass Objective	$w_0 = 1, 10$	n/a
Drift Ratio Constraint	$w_1 = 1\text{e}3$	$\gamma_1 = 1\text{e-}1$
Variety Constraint	$w_2 = 1.0$	$\gamma_2 = 5\text{e-}4$
Entropy Constraint	$w_3 = 1.0$	$\gamma_3 = 1\text{e-}3$

4. Learning Curves for Neural Simulators

Figure 3 and Figure 4 plot the learning curves of different models and ablation studies.

5. User Study

We invite a structural engineer to work on a design in our user study and compare the human design with the design output from NeuralSizer. The structural engineer has 30 minutes to iterate on a 4-story building design following the manual design workflow. We also run NeuralSizer to create one design. All designs are evaluated with respect to the mass objective, the drift ratio constraint($< 0{:}025$), and the variety constraint(< 6). In 30 minutes, the structural engineer is able to create five iterations, and the evaluation results are presented in Table 6. The beam and column usages are sorted in the descending order of strength. Using stronger columns or beams lowers drift ratios, but leads to a larger total mass. The yellow-colored row highlights the most light-weighted designs that satisfy all constraints created by the structural engineer and NeuralSizer. The red cells indicate that the drift ratio exceeds the limit. The designs are visualized in Figure 5.

In the first two iterations, the structural engineer prioritizes

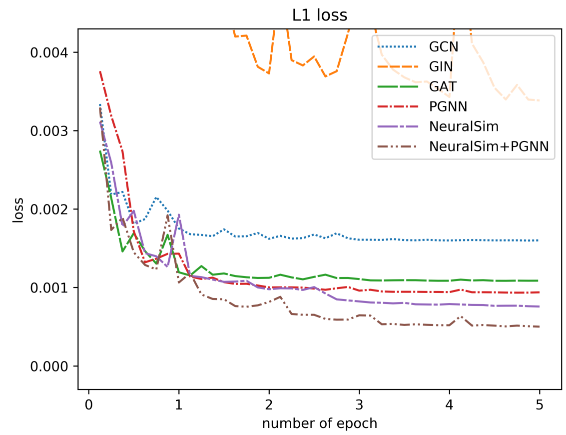


Figure 3. Learning curves of different models.

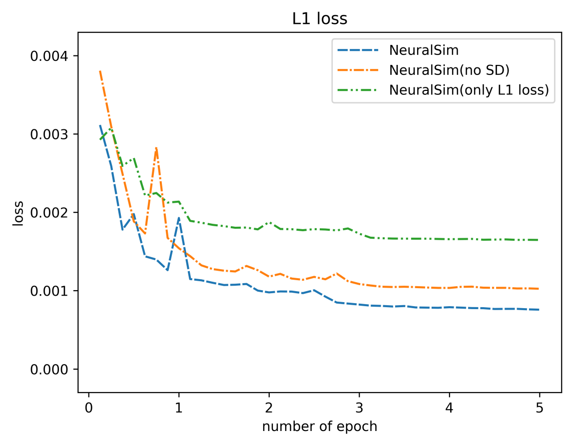


Figure 4. Learning curves of ablation studies.

the constraint satisfaction by choosing two designs using mostly the strongest columns and beams. The simulation results of the first two designs provide not only a baseline, but also a vague idea about the relation between the cross-section decisions and the drift ratio changes. The third design is the first attempt to optimize the mass objective and has a significantly smaller mass. However, when the structural engineer tries to further decrease the mass in the fourth and fifth iteration, both designs violate the drift ratio constraint. Compared to the best human design in the third iteration, NeuralSizer outputs a design that has a lower mass while satisfying the constraints. Note that the 4-story building example is relatively simple. As the building becomes more massive (10 stories with more than 500 bars for example), the performance of human designs can degrade.

Potentially, human might still be able to create more optimal designs given more iterations. As a result, we claim that NeuralSizer can save the time and effort by providing a better initial design to structural engineers.

Table 6. User Study Results

Experiment	Mass Tonnage	Beam Usage	Column Usage	Drift Ratios in Seismic X (Limit = 0:025)				Drift Ratios in Seismic Y (Limit = 0:025)			
				Story 1	Story 2	Story 3	Story 4	Story 1	Story 2	Story 3	Story 4
			Human								
Iteration 1	422.06	188; 0; 0; 0; 0; 0; 0; 0	62; 62; 0; 0; 0	0.0209	0.0174	0.015	0.0094	0.0207	0.0173	0.0149	0.0093
Iteration 2	432.47	188; 0; 0; 0; 0; 0; 0; 0	124; 0; 0; 0; 0	0.0213	0.0178	0.0135	0.0084	0.0211	0.0177	0.0134	0.0083
Iteration 3	279.41	0; 0; 0; 0; 0; 0; 0; 188	62; 62; 0; 0; 0	0.0208	0.0173	0.0149	0.0093	0.0206	0.0172	0.0148	0.0093
Iteration 4	201.86	0; 0; 0; 0; 0; 0; 0; 188	0; 0; 0; 0; 124	0.0305	0.0253	0.0192	0.0119	0.0302	0.0251	0.0191	0.0119
Iteration 5	267.95	47; 0; 47; 0; 47; 0; 0; 47	0; 0; 0; 0; 124	0.0306	0.0253	0.0193	0.0119	0.0303	0.0251	0.0191	0.0119
			NeuralSizer								
Objective Weight 10	**257.47**	0; 0; 0; 0; 0; 0; 0; 188	71; 7; 2; 4; 40	0.0196	0.0172	0.0149	0.0165	0.0192	0.0198	0.0146	0.0161

References

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019.