ALPHATRUSS: TOWARDS OPTIMAL TRUSS LAYOUT DESIGN THROUGH DEEP REINFORCEMENT LEARNING

Xingcheng Yao
IIIS, Tsinghua University
yxc18@mails.tsinghua.edu.cn

ABSTRACT

We present AlphaTruss, a 2-stage pipeline designed for truss layout optimization, which incorporates deep reinforcement learning for better expressibility and larger solution space. Empirical results show that AlphaTruss can achieve an average of 377.06 kg improvement over traditional optimization methods. Theoretical and numerical analysis further justifies the effectiveness of AlphaTruss.

Keywords: Truss Generation; Structural Optimization; Deep Reinforcement Learning;

1 Introduction

A truss is a framework composed of several joint bars, whose layout is represented by node locations, connection topology between nodes and cross-sectional areas of each connection [4]. After properly adjusting the components of a truss layout, its weight could be reduced without affecting the load capacity and structural stability. Hence it's vital for a designer to consider the optimal truss layout design, which achieves the minimal weight under certain constraints.

The task of the automatic generation of optimal truss layout is made challenging by the enormous solution space when considering node locations, connection topology and cross-sectional areas simultaneously. To tackle such a challenge, a lot of research efforts have been made through methods based on physical expriments [6], numeric optimization [5, 3, 10, 22, 15, 31] and graphics [11, 1, 23, 16, 27]. However, these methods suffer from two limitations. First of all, their optimization models are usually discrete and heuristic, which limits the expressibility of the model. Secondly, due to the heavy computational budget for optimization, these algorithms cannot be scaled up to larger solution space. Currently, most of these methods were built on the assumption of limited solution space, i.e. generating a truss layout using trangles, polygons or polyhedra, which largely restricts the solution space of the optimization algorithms. For example, a graphic-based algorithm cannot generate a truss structure that is not triangulated, so some additional bars may well be added to the generated truss layout, which is unnecassary to guarantee stability and load capacity.

To improve the expressibility and expand the solution space, we propose a two-stage truss layout generation pipeline which is closely connected to deep reinforcement learning. During the first stage, we model optimal truss layout design as a sequence generation problem in discrete action space. Specifically, the process to design a truss layout is formulated as a sequence of actions, each

*This work is done during Xingcheng's internship at Shanghai Qi Zhi Institute.

action corresponds to a design variable including the location of a node and the cross-sectional area between the connection of two nodes. Automatic generation of a dedicated sequence has been constantly studied in the literature of natural language processing [8, 24]. And policy gradient reinforcement learning is testified as an effective tool to optimize a generated sequence over a discrete reward function [19, 21, 32]. At the second stage, the truss layout generated beforehand is fine-tuned with a high dimensional continuous action space, through deep reinforcement learning algorithms specifically designed for continuous states and action spaces. In both stages, a reward function is calculated as the quality of the generated truss layout, measured by its total weight, displacement, stress etc. We name the pipeline as **AlphaTruss**. To the best of our knowledge, this is the first work that incorporates deep reinforcement learning into truss layout optimization.

In summary, the contribution of this work is 3-fold:

- Infrastructures that support the research of truss layout optimization with deep reinforcement learning is implemented.
- The issue of solution space scalability is largely addressed with deep neural networks and long-time training.
- New state-of-the-art performance on truss layout design is achieved.

2 Related Work

Truss Layout Optimization Ever since [17] derived the optimality criteria for minimal weight truss layout, several research contributions have been made to automating the optimal truss layout design process. There is also a line of work that try to design truss layout through numeric optimization, some of them are based on specific mathematical models such as ground structure [5] and material distribution [3], while others borrow the ideas from more general optimization algorithms and effectively apply them to the problem of truss layout optimization. Those methods includes envolutionary algorithms [10, 31], particle swarm optimization [15, 12] and genetic algorithm [20]. Another dedicated line of works draw their inspirations from graphics, and define graphic shapes as atomic elements to generate truss layout, which includes shape grammar [11], grammar statics [1, 23] and triangulation [16, 27]. Besides, there are some attempts made to utilize physical experiments [6] for structure design, coupled with numerical simulations. However, such methods with physical simulation have been suffering from poor accuracy and low efficiency.

Deep Reinforcement Learning This work involves two applications of deep reinforcement learning. The first is reinforcement learning for sequence generation [19, 21, 32]. Among various methods, REINFORCE [30] is the most frequently adopted one. [21] proposed a variant of REINFORCE named Self-Critical Sequence Training (SCST), which have become the de-facto method to optimize sequence generation models over a discrete reward function. The second is deep reinforcement learning with continuous state and action spaces [25, 9, 18]. This line of work often aims at improving sample efficiency and training stability of deep reinforcement learning framework.

3 Background

A truss layout is characterized by a set of nodes and bars, denoted as $(\mathcal{V}, \mathcal{E})$. Each node $u \in \mathcal{V}$ is a point in Euclidean space, and each bar $e \in \mathcal{E}$ is defined as a tuple $e = (u, v, \alpha)$, where $u, v \in \mathcal{V}$,

and $\alpha \in \mathbb{R}$ is the cross-sectional area of the bar. The objective of truss layout optimization is to minimize the total weight of the generated layout, which can be formally expressed as follows.

$$\min_{(\mathcal{V},\mathcal{E})\in\Omega} \rho \sum_{(u,v,\alpha)\in\mathcal{E}} \alpha \|u-v\|_2 \tag{1}$$

where Ω is the set of legal truss layouts under several design constraints. Among all the design constraints, the most basic one is the design envelope constraint, which means the end points and the cross-sectional area of each bar must lie in a bounded interval. Also, a legal truss layout must meet a line of geometric stability constraints, whose details are defered to Section B.

4 AlphaTruss

4.1 Stage 1

At stage 1, the node location and cross-sectional area bounds are all uniformly discretized. In such a grid world setting, a truss layout can be expressed as a sequence $S = a_1 a_2 \dots a_m$ (We later denote it by productions $\prod_{i=1}^{m} a_i$ for simplicity). Each element $a_i \in S$ is from a finite set. When given the expected number of nodes N, the length of sequence S will be fixed to $|S| = N + N*(N-1)/2$. The first N elements represents the location of each node, and the last $N*(N-1)/2$ elements represent the existance and cross-sectional area (if existed) of each bar. Hence truss layout optimization is reduced to an optimal sequence generation problem.

In AlphaTruss, greedy search is used for sequence generation. When a utility function for each prefix of the sequence $\mu_{\mathrm{p}}(\cdot)$ is defined, each element in the sequence will be iteratively chosen as:

$$a_i = \arg\max_{a} \mu_{\mathrm{p}} \left(\prod_{j=1}^{i-1} a_j \cdot a \right) \tag{2}$$

Suppose the utility is defined for a sequence of length m as $\mu_{\mathrm{s}}(S)$, we define the utility for each prefix of the sequence as:

$$\mu_{\mathrm{p}} \left(\prod_{i=1}^{n} a_i \right) = \max_{\{a_j\}_{j=n+1}^{m}} \mu_{\mathrm{s}} \left(\prod_{i=1}^{n} a_i \cdot \prod_{j=n+1}^{m} a_j \right) \tag{3}$$

Theorem 1. *The greedy search process defined with (2) and (3) produces the optimal sequence*

We defer the proof of Theorem 1 to Section A. To evaluate the utility of a prefix, a reinforcement learning agent is trained to generate the optimal sequence with the prefix as the initial state. The utility of the optimal sequence generated during the training process will then be used to approximate the utility of that prefix. We use SCST [21] to train the agent, by which the gradient of the current policy is estimated as follows:

$$\nabla_\theta L(\theta) = -(\mu_{\mathrm{s}}(S) - \mu_{\mathrm{s}}(\hat{S}))\nabla_\theta \log p_\theta(S) \tag{4}$$

where $\hat{S}$ is the previously generated optimal sequence, S is the sequence generated by the current policy with greedy decoding. We parameterize the policy p_θ with a Transformer model [29], which could be used to generate the sequence in an auto-regressive manner.

One could also interprete the greedy search process from the perspective of exploration and exploitation: each time an element is generated, the results from the last trained agent is exploited to restrict the exploration space. Such a progressive restriction on exploration by exploiting previous results is heuristically designed for sequence generation.

4.2 Stage 2

At stage 2, the truss layout generated in grid world during the previous stage is refined in a high dimensional continuous space, by incrementally moving the node positions and modify the cross-sectional area of each bar. We model the refinement process as a deterministic infinite-horizon Markov decision process (MDP), denoted by the tuple $(\mathcal{S}, \mathcal{A}, t, r)$. Here the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are continuous with the same dimensionality. Each state $\mathbf{s} \in \mathcal{S}$ is a concatenation of all the point locations and bar cross-sectional areas in the truss, and each action $\mathbf{a} \in \mathcal{A}$ is the concatenation of increments for each point location and cross-sectional area. The trainsition function is defined as $t(\mathbf{s}, \mathbf{a}) = \mathbf{s} + \mathbf{a}$, and the reward function $r(\mathbf{s})$ is defined to measure the quality of the resulting truss layout.

We optimize the refinement policy using Soft Actor-Critic (SAC) [9], which is a maximum entropy deep reinforcement learning algorithm. Extensive experiments have shown that SAC can effectively address policy learning in continuous action spaces. Since the state of the truss layout is time independent, multiple rounds of refinement can be conducted by setting the initial state as the optimal layout obtained from the last refinement. Multi-round refinement also progressively restricts the exploration space, in order to speed up the convergence of the refinement process.

5 Experiment

5.1 Settings

Our experiments are conducted on unconditional truss layout optimization tasks and conditional truss layout optimization tasks in a two-dimensional plane.

Unconditional Truss Layout Optimization We first test AlphaTruss on size, shape and topology optimization. In these test cases, only special nodes with supports or loads are initially provided. As shown in Table B.1, we consider two load cases. In Case 1, 4 initial nodes are provided, 2 new nodes are required to generate. In Case 2, 6 initial nodes are provided, 1 new node is required to generate. The algorithm is also responsible to decide the connection topology between the nodes and the cross-sectional areas of each bar. We use previously published results with the same settings, Fenton, et al. [16] which doesn't consider buckling constraints and Petrović, et al. [20] which does, as our baselines.

Conditional Truss Layout Optimization We also test AlphaTruss on size or both size and topology optimization. In these test cases, node positions and connection topology between nodes are all provided. The algorithm is responsible to decide on variables about each bar. For size optimization, the algorithm only need to select the cross-sectional area for each bar. For both size and topology optimization, the algorithm can choose the cross-sectional area for a bar, or directly delete the bar. We choose the ten-bar benchmark that is frequently used by previous works [12, 14, 28, 2] as the input layout, and test the proposed pipeline with both load cases shown in Table B.1. We use the

Methods	Case 1 (kg)	Case 2 (kg)	Case 1* (kg)
Fenton, et al. [16]	2217.54	2097.54	-
Petrović, et al. [20]	-	-	3172.77
AlphaTruss	**2131.88**	**1369.69**	**2855.10**

Table 1: Results for unconditional truss layout optimization. $*$ means the test case takes Euler buckling constraints on stressed bars into account.

Methods	Case 1 (kg)	Case 2 (kg)	Case 1* (kg)	Case 2* (kg)
Lee, et al. [14]	2294.21	**2117.73**	-	-
Kaveh, et al. [12]	**2293.62**	2120.90	-	-
Assimi, et al. [2]	-	-	**2221.92**	2019.66
Tejani, et al. [28]	-	-	2233.50	2228.43
AlphaTruss	2295.83	2122.77	2222.50	**2011.50**

Table 2: Results for conditional truss layout optimization. $*$ means the test case also includes topology optimization.

results reported in previous literatures as our baselines. In all the cases of conditional truss layout optimization, buckling constraints is not considered.

We defer the detailed settings of the environment and training process in Section B.

5.2 Results

Table 1 shows the results for unconditional truss optimization. In Case 1, AlphaTruss outperforms the corresponding baselines with or without buckling constraints by a margin of 317.67 kg and 85.66 kg respectively. In Case 2 without buckling constraint, AlphaTruss even outperforms the baseline method by a margin of 727.85 kg. These results consistently show the effectiveness of AlphaTruss in truss layout optimization problem that has large solution spaces.

Table 2 shows the results for conditional truss optimization. For both cases with or without topology optimization, the results are mixed. This might indicate the reduction on solution space will largely restrict the capability of AlphaTruss. Be that as it may, AlphaTruss still performs on par with the best baseline method, within a margin of +5.04 kg.

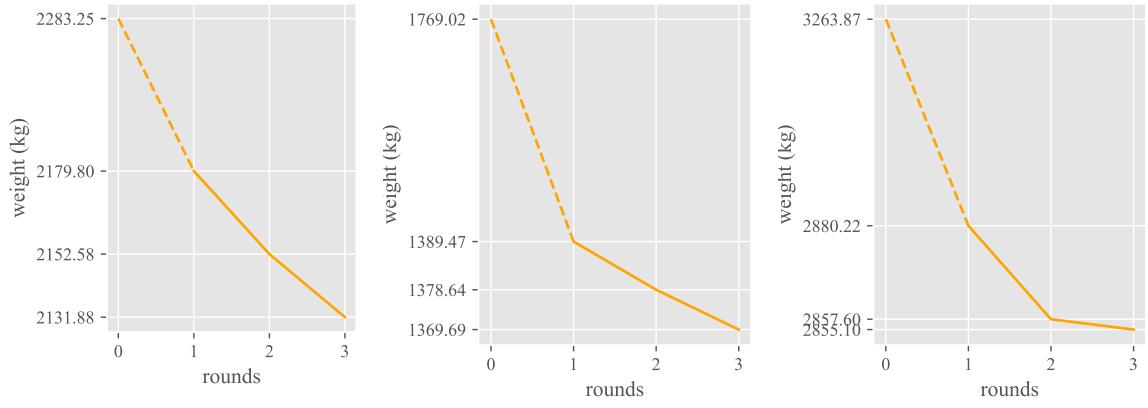
The detailed information about each generated truss layout is defered to Section C.

5.3 Analysis

Effectiveness of greedy search with SCST Given the combination of greedy search and SCST in stage 1 of AlphaTruss, one natural question is how does each component contribute to the generated layout. To understand it better, we conduct ablation studies on components in stage 1. The results for different variants of stage 1 are shown in Table 3. Compared with SCST without greedy restriction on the sequence prefix, our proposed method successfully boosts up the training efficiency, with over 600 kg performance gain under the same amount of training steps. Compared with greedy

Methods	Case 1 (kg)	Case 2 (kg)	Case 1* (kg)
SCST	2930.47	3732.72	4387.55
GS + random	2638.70	2399.39	3674.22
GS + SCST	**2283.25**	**1769.02**	**3263.87**

Table 3: Test results for different variants of stage 1. GS + SCST: greedy search with SCST as proposed; GS + random: greedy search with random sampling; SCST: using SCST without progressive restriction on the sequence prefix. $*$ means the test case includes buckling constraints.



Case 1 Case 2 Case 1 *

Figure 1: Weights of the truss layout after each round of refinement. The dashline represents the improvement from the output in stage 1, and active line represents improvements from the previous round of refinment. $*$ means the test case also includes Euler buckling constraints.

search using random sampling to evaluate prefix utility, SCST can highly improve data efficiency, resulting in more accurate evaluation results with the same amount of sampled trajectories.

Multi-round refinement Stage 2, consisting of multiple rounds of refinements, is the key of AlphaTruss to expanding the solution space. Figure 1 shows the effectiveness of multi-round refinement. The first round of refinement, which expands the solution space from the grid world to a continuous space, brings about up to hundreds of kilograms of improvements. The following up refinements, though with smaller magnitude, can still improve the weight by up to 20 kilograms.

6 Conclusion

We present AlphaTruss, a 2-stage pipeline for truss layout optimization. Deep reinforcement learning is successfully incorporated into this pipeline to improve the expressibility and expand the solution space. AlphaTruss achieves new state-of-the-art results in truss layout optimization that considers size, shape and topology simultaneously. However, one limitation of AlphaTruss is that its two stages are loosely connected, which makes either stage replaceable with respect to another. Also, lots of topics are left unexplored, including the end-to-end training of truss layout optimization model and the transferability of a learned deep model in truss layout design. We defer the solution to these issues to future work.

References

[1] M. Akbarzadeh. Three-dimensional graphical statics using reciprocal polyhedral diagrams. *ETH Zurich*, 2016.

[2] H. Assimi, A. Jamali, and N. Nariman-zadeh. Sizing and topology optimization of truss structures using genetic programming. *Swarm Evol Comput*, 37:90–103, 2017.

[3] M.P. Bendsøe. Optimal shape design as a material distribution problem. *Struct Multidiscipl Optim*, 1:193–202, 1989.

[4] M.P. Bendsøe and O. Sigmund. Topology optimization: Theory, method and applications. *Springer*, 2003.

[5] W. Dorn. Automatic design of optimal structures. *J de Mec*, 1964.

[6] O. Frei and B. Rasch. Finding form: towards an architecture of the minimal. *Stuttgard: Edition Axel Menges*, 1995.

[7] J. Clerk Maxwell F.R.S. L. on the calculation of the equilibrium and stiffness of frames. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 27(182):294–299, 1864.

[8] Albert Gatt and Emiel Krahmer. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170, 2018.

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[10] W Huang and Y-M Xie. A further review of eso type methods for topology optimization. *Struct Multidiscipl Optim*, 41:671–83, 2010.

[11] Sea K. and Cagan J. Languages and semantics of grammatical discrete structures. *Artif Intell Eng Des Anal Manuf*, 13:241–51, 1999.

[12] A. Kaveh and Talatahari S. Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures. *Comput Struct*, 81:267–83, 2009.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] K.S. Lee and Geem Z.W. A new structural optimization method based on the harmony search algorithm. *Comput Struct*, 82:781–98, 2004.

[15] Guan-Chun Luh and Chun-Yi Lin. Optimal design of truss-structures using particle swarm optimization. *Computers & structures*, 89(23-24):2221–2232, 2011.

[16] Fenton M., McNally C., Byrne J., Hemberg E., McDermott J., and O'Neill M. Discrete planar truss optimization by node position variation using grammatical evolution. *IEEE Trans. Evol. Comput*, 20:577–89, 2016.

[17] A.G.M. Michell. The limits of economy in frame-structures. *Lond Edinb Dubl Pil Mag*, 6:589–97, 1904.

[18] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Trust-pcl: An off-policy trust region method for continuous control, 2018.

[19] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization, 2017.

[20] Nenad Petrović, Nenad Kostić, and Nenad Marjanović. Comparison of approaches to 10 bar truss structural optimization with included buckling constraints. *Applied Engineering Letters*, 2(3):98–103, 2017.

[21] Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[22] G.I.N. Rozvany. A critical review of established methods of structural topology optimization. *Struct Multidiscipl Optim*, 37:217–37, 2009.

[23] Mozaffari S., Akbarzadeh M., and Vogel T. Graphic statics in a continuum: Strut-and-tie models for reinforced concrete. *Comput Struct*, 240, 2020.

[24] Sashank Santhanam and Samira Shaikh. A survey of natural language generation techniques with a focus on dialogue systems-past, present and future directions. *arXiv preprint arXiv:1906.00500*, 2019.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[27] Stanković T. and Shea K. Investigation of a voronoi diagram representation for the computational design of additively manufactured discrete lattice structures. *J Mech Design*, 142:1–21, 2020.

[28] G.G. Tejani, V.J. Savsani, V.K. Patel, and P.V. Savsani. Size, shape, and topology optimization of planar and space trusses using mutation-based improved metaheuristics. *J Comput Des Eng*, 5:198–214, 2018.

[29] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[30] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[31] Chun-Yin Wu and Ko-Ying Tseng. Truss structure optimization using adaptive multi-population differential evolution. *Structural and Multidisciplinary Optimization*, 42(4):575–590, 2010.

[32] Xuenan Xu, Heinrich Dinkel, Mengyue Wu, and Kai Yu. A crnn-gru based reinforcement learning approach to audio captioning. 2020.

A Proof of Theorem 1

Theorem 1. *For any finite set $\mathcal{A}$, $\mathcal{S}_m(\mathcal{A})$ is the set of sequences with fixed length m consisting of elements from $\mathcal{A}$, $\mathcal{P}_n^m(\mathcal{A})$ is the set of prefixes with length n of sequences in $\mathcal{S}_m(\mathcal{A})$. For each $P \in \mathcal{P}_n^m(\mathcal{A})$, denote $\mathcal{S}_m^P(\mathcal{A})$ as the set of sequences in $\mathcal{S}_m(\mathcal{A})$ with P as its prefix. For any utility function μ_{s} defined over $\mathcal{S}_m(\mathcal{A})$ and $P \in \mathcal{P}_n^m(\mathcal{A})$, if we define:*

$$\mu_{\mathrm{p}}(P) = \max_{S \in \mathcal{S}_m^P(\mathcal{A})} \mu_{\mathrm{s}}(S)$$

then we have:

$$\arg\max_{S \in \mathcal{S}_m(\mathcal{A})} \mu_{\mathrm{s}}(S) = \prod_{i=1}^{m} \left\{ a_i \mid a_i = \arg\max_{a \in \mathcal{A}} \mu_{\mathrm{p}} \left(\prod_{j=1}^{i-1} a_j \cdot a \right) \right\} \tag{5}$$

Proof. Let S^* be the optimal sequence defined in the left hand side of Equation 5, $\hat{S}$ be the sequence defined in the right hand side of Equation 5 and P_n be the prefix of $\hat{S}$ of length n. Suppose $\mu_{\mathrm{s}}(S^*) > \mu_{\mathrm{s}}(\hat{S})$, then let P_i be the shortest prefix of $\hat{S}$ such that $\mu_{\mathrm{p}}(P_i) < \mu_{\mathrm{s}}(S^*)$, we have $i > 1$ and $\mu_{\mathrm{p}}(P_{i-1}) = \mu_{\mathrm{s}}(S^*)$ by definition. Then from Equation 5:

$$\mu_{\mathrm{p}}(P_i) = \max_{a \in \mathcal{A}} \mu_{\mathrm{p}}(P_{i-1} \cdot a)$$

Note that for any $S' \in \mathcal{S}_m^{P_{i-1}}(\mathcal{A})$, suppose the i-th element for S' is b, we have:

$$\mu_{\mathrm{s}}(S') \leq \max_{S \in \mathcal{S}_m^{P_{i-1} \cdot b}(\mathcal{A})} \mu_{\mathrm{s}}(S) = \mu_{\mathrm{p}}(P_{i-1} \cdot b) \leq \max_{a \in \mathcal{A}} \mu_{\mathrm{p}}(P_{i-1} \cdot a) = \mu_{\mathrm{p}}(P_i)$$

Hence:

$$\mu_{\mathrm{s}}(S^*) = \mu_{\mathrm{p}}(P_{i-1}) = \max_{S \in \mathcal{S}_m^{P_{i-1}}(\mathcal{A})} \mu_{\mathrm{s}}(S) \leq \mu_{\mathrm{p}}(P_i)$$

which is a contradiction. □

B Experiment settings for environment and training

Enviroment Settings

5 stability constraints are taken into account:

1. Maxwell criterion [7]
2. Finite-definiteness of stiffness matrix
3. Displacement allowance for each node
4. Stress allowance for each bar
5. Euler buckling constraint [20] (only for part of test cases)

All physical values involved in those constraints were calculated by OpenSeesPy[1], with Young's modulus of the materials of each bar set to $68950\ \mathrm{MPa}$. We implement the environment for both

[1] `https://openseespydoc.readthedocs.io/en/latest/`

Node id	Locations (mm)	Restricted Degrees	Load Forces (N)	
			Case 1	Case 2
1	$(0, 0)$	2	$(0, 0)$	$(0, 0)$
2	$(0, 9144)$	2	$(0, 0)$	$(0, 0)$
3	$(9144, 0)$	0	$(0, -444800)$	$(0, -667200)$
4	$(18288, 0)$	0	$(0, -444800)$	$(0, -667200)$
5	$(9144, 9144)$	0	$(0, 0)$	$(0, 222400)$
6	$(18288, 9144)$	0	$(0, 0)$	$(0, 222400)$

Table B.1: Initial nodes for both test cases. For Case 1, in unconditional truss layout generation, only the first 4 nodes are used as initial input; in conditional truss layout optimization with topology optimization, Node 5 and Node 6 can be eliminated when all adjacent bars are eliminated.

stages, with the utility for the generated sequence at the first stage and the reward of the refined truss at the second stage both have a form of:

$$r(\mathcal{V}, \mathcal{E}) = \begin{cases} \dfrac{\lambda}{(\rho \sum_{(u,v,\alpha)\in\mathcal{E}} \alpha \|u - v\|_2)^2}, & \text{all constraints are met} \\ c, & \text{otherwise} \end{cases} \tag{6}$$

where c is from a set of constants determined by the violated constraints. When the truss layout violate the Maxwell criterion or finite-definiteness of the stiffness matrix, $c = -1$, otherwise, $c = 0$. And λ is the scaling factor, set as 1.69×10^7. To meet the settings in previous literatures, The density ρ is set to $2767.99\ \mathrm{kg/m^3}$, the maximum displacement for each node is set to $50.8\ \mathrm{mm}$, the maximum compression and tension stress for each bar are both set to $172.3\ \mathrm{MPa}$, the x-coordinate for each node is set to range from $0\ \mathrm{mm}$ to $18288\ \mathrm{mm}$, the y-coordinate for each node is set to range from $0\ \mathrm{mm}$ to $9144\ \mathrm{mm}$, the minimum cross-sectional area for each bar is set to $0.6452\ \mathrm{cm^2}$ and the maximum cross-sectional area is set to $200\ \mathrm{cm^2}$ when Euler buckling constraint is not considered or $400\ \mathrm{cm^2}$ otherwise.

Training Settings At stage 1, for unconditional truss layout optimization, node location is selected from a 7×5 grid in load Case 1, and from a 17×9 grid in load Case 2; for both conditional and unconditional optimzation, value for cross-sectional area of each bar is uniformly discretized to 10 pieces. To evaluate a prefix, we train the agent for 100 steps. In each training step, we sample 32 generated sequences using the current policy network with dropout noise [26] at rate 0.1, and use those sequences as a mini-batch. We use Adam optimizer [13] with learning rate 5e-5, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. At stage 2, we use SAC algorithm implemented in RLkit[2]. The policy in stage 2 is optimized for 3000 epochs. In each epoch, 10000 episodes were collected and for each 1000 episodes, the policy network will be trained for 10 steps with batch size 256. A refinement trajectory is considered to be one episode whenever the refinement is taken over 10 steps and the weight for the truss layout is increased. The refinement is conducted for 3 rounds for unconditional truss optimization and 6 rounds for conditional truss optimization. we report the best results during the process of refinement.

[2] `https://github.com/rail-berkeley/rlkit`

C Detailed information on generated truss layout

New Nodes	
Node id	Location (m)
5	(6.571718521267175, 4.264582944989205)
6	(11.121687366932633, 7.135966499805452)
Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(4,3)	85.65544317166014
(1,3)	123.86447558800383
(1,5)	143.99899731079743
(3,5)	33.250477512677522
(2,5)	32.598720490932464
(4,6)	140.00660687685011
(2,6)	200
(5,6)	102.09372222423551

Table C.1: Optimized truss layout for case 1, with weight 2131.88 kg

New Nodes	
Node id	Location (m)
7	(14.262575678783659, 4.023368473774196)
Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(4,3)	50.10266897678375
(1,3)	162.9724128862221
(4,2)	0.7942145146846764
(3,2)	63.0717089096705
(2,5)	2.773422417831422
(5,6)	13.855296669403711
(4,7)	78.50108960072204
(2,7)	68.22245860497157
(5,7)	24.797533975044893
(6,7)	54.63785142898561

Table C.2: Optimized truss layout for case 2, with weight 1369.69 kg

New Nodes	
Node id	Location (m)
5	(3.781325548857451, 0.0)
6	(13.215890158912533, 4.574480132272839)
Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(4,3)	276.5164307248831
(3,2)	126.5094101947785
(1,5)	236.7708105175734
(3,5)	291.7419581330299
(2,5)	0.6452
(4,6)	128.79807654981616
(3,6)	139.56946235537535
(2,6)	139.41888256146914

Table C.3: Optimized truss layout for case 1 including buckling constraints, with weight 2855.10 kg

Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(1,3)	152.53565339754424
(2,3)	47.89338822960852
(3,4)	98.35076424047151
(1,5)	135.13457657613756
(2,5)	195.6676412458896
(3,5)	0.6452
(4,5)	138.64068322186476
(3,6)	0.6452
(4,6)	3.4014700413227087
(5,6)	0.6452

Table C.4: Optimized truss layout for case 1 with only size optimization, weight 2295.83 kg

Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(1,3)	166.1644706912756
(2,3)	79.39892368060747
(3,4)	89.44342777690888
(1,5)	82.2628322794437
(2,5)	151.9332522342046
(3,5)	0.6452
(4,5)	132.57439003954727
(3,6)	0.6452
(4,6)	12.833880894080794
(5,6)	0.6452

Table C.5: Optimized truss layout for case 2 with only size optimization, weight 2122.77 kg

Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(1,3)	148.16986556063226
(2,3)	38.972172076763042
(3,4)	96.45939446825713
(1,5)	138.3991808131854
(2,5)	190.97812940326274
(4,5)	135.51302842055959

Table C.6: Optimized truss layout for case 1 with size optimization and topology optimization, weight 2222.50 kg

Edges	
Adjacent Nodes	Cross-sectional Area (cm^2)
(1,3)	148.44463820149117
(2,3)	61.3561316210587
(3,4)	89.43289802335117
(1,5)	89.40670726735303
(2,5)	156.13548522413815
(4,5)	122.724817526971
(3,6)	0.6452
(4,6)	13.03386601821847

Table C.7: Optimized truss layout for case 2 with size optimization and topology optimization, weight 2011.50 kg