Automating Rigid Origami Design

Jeremia Geiger [*1] **Karolis Martinkus** [*1] **Oliver Richter** [*1] **Roger Wattenhofer** [1]

Abstract

While rigid origami has shown potential in a large diversity of engineering applications, current rigid origami crease pattern designs mostly rely on known tessellations. This leaves a potential gap in performance as the space of rigidly foldable crease patterns is far larger than these tessellations would suggest. In this work, we build upon the recently developed Principle of Three Units method to formulate rigid origami design as a discrete optimization problem. Our implementation allows for a simple definition of diverse objectives and thereby further expands the potential of rigid origami to optimized, application-specific crease patterns. By specifying custom reward functions, we can find patterns, which result in novel, foldable designs for everyday objects.

1. Introduction

Origami may be ancient to the arts, but it is young to science. Only in recent years, have researchers begun to rigorously investigate the underlying principles of folding. This has resulted in applications of origami in various fields of robotics, architecture, biomedical engineering, deployable structures, metamaterials, aerospace, and more (Meloni et al., 2021; Wang, 2019; Callens & Zadpoor, 2018; Turner et al., 2016; Kuribayashi et al., 2006; Morgan et al., 2016).

Rigidly foldable patterns, known as rigid origami, are of particular interest in practical applications since the material does not deform while folding. Even more importantly, in principle the folding motion of a rigid pattern can be induced by a single folding activation, also referred to as the patterns' degree of freedom (DOF) (He & Guest, 2018; Zimmermann et al., 2020). This makes rigid origami easy to deploy and particularly interesting for engineering applications.

Traditional origami tackles the question of how a given crease pattern , defined by a set of creases drawn on a flat piece of paper, will fold. However, a more useful question in engineering is how to define a pattern such that it folds into a given target shape. This is commonly known as the inverse origami problem. Note that this is a complex problem since determining the foldability of a general crease pattern alone is NP-hard (Demaine et al., 2016). Recent works started to address the inverse origami problem (Demaine & Tachi, 2017; Dudte et al., 2016; He & Guest, 2018). Unfortunately, these previous works either focus on non-rigid origami and allow the pattern to have many degrees of freedom (Demaine & Tachi, 2017) or approximate shapes using only known tessellations and their variations (Dudte et al., 2016; He & Guest, 2018; Tachi, 2010). Even though it is possible to approximate some shapes very well using rigid origami tessellations, beyond the space of the tessellations there is a much larger space of general foldable patterns, which we seek to explore in this work. This should help in any future engineering applications of rigid origami, which will require flexible design methods (Turner et al., 2016; Meloni et al., 2021).

As common in the literature, we treat the crease pattern as a graph where edges represent the crease lines and vertices their intersections (He & Guest, 2019). We place vertices one after the other on a predefined grid to construct these graphs. Our formulation brings the optimization problem close to combinatorial problems, such as board games, where Reinforcement learning (RL) and other techniques have demonstrated great success (Silver et al., 2016; 2018; Vinyals et al., 2019). We derive the methodology characterizing the game environment, expose it to various types of artificial agents for validation, and showcase it in generative design tasks. Our results highlight the flexibility of our formulation and open up the possibility for future research – both in rigid origami design and optimization methods for this application.

2. Rigid Origami as Discrete Optimization

We denote a crease pattern's state at time step t by $s_t \in \mathcal{S}$, where $\mathcal{S}$ denotes the space of developable crease patterns which can be unfolded into a plane sheet. Every state $s \in \mathcal{S}$ can further be represented by the corresponding crease pattern graph $s = G(V, E)$, where V is the set of vertices

[*]Authors in alphabetical order [1]ETH Zurich. Correspondence to: Jeremia Geiger <jeremia.s.r.geiger@gmail.com>.

Workshop on Machine Learning For Computational Design, 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

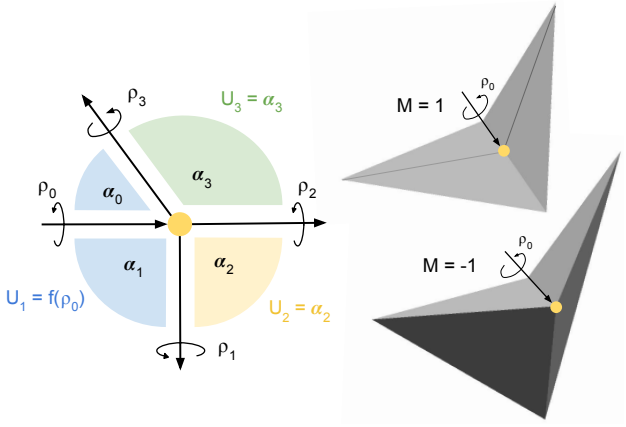


Figure 1: A degree 4 vertex with one incoming and 3 outgoing edges. Indicated are the sector angles α, the folding (dihedral) angles ρ and the unit angles U_1, U_2 and U_3. The illustrations on the right show the effect of the rigid body mode $M \in \{-1, 1\}$ for the same input angle ρ_0.

where crease lines meet and E is the set of directed edges along the crease lines.

Every degree-δ vertex $v_i \in V$ is characterized by a set of sector angles $\boldsymbol{\alpha} = \{\alpha_i\}$ and a rigid body mode M, while every crease line $e_i \in E$ has a folding (dihedral) angle $\{\rho_i\}$ assigned to it. Here $\alpha_i \in (0, 2\pi)$, $\rho_i \in (-\pi, \pi)$, and $M \in \{-1, 1\}$. Figure 1 shows an illustration of these definitions for a simple degree 4 vertex. Sector angles $\boldsymbol{\alpha}$ remain fixed during the folding motion (as panels are rigid), whereas the dihedral angles $\boldsymbol{\rho}$ change as the object folds according to the dynamics implied by the crease pattern. Throughout the paper, we limit our patterns to fold within a single degree of freedom (DOF), represented by the driving angle ρ_0.[1] Note that patterns with a single driving angle are particularly appealing, as we only have to actuate one hinge to fold the whole object. The solution of vertex kinematics for any given configuration depends on the mountain or valley assignment of the creases. This property is expressed by the rigid body mode M.

PTU Kinematic Model The Principle of Three Units (PTU) method (Zimmermann et al., 2020; Zimmermann & Stanković, 2020) provides both general rules for expanding an embedded crease pattern s, and an efficient kinematic model for folding simulation.

Specifically, for a directed acyclic graph $G(V, E)$ the PTU kinematic model allows us to sequentially solve for all folding angles $\boldsymbol{\rho}$, starting from the graph origin and finishing at the leaves.

[1] We do allow for multiple sources in some of our starting configurations, however, all sources have the same driving angle ρ_0.

In its essence the PTU is based on the following observation: for a given vertex, its rigid folding motion is kinematically determinate if all but three of the adjacent edges' driving angles ρ_i are known (incoming edges). The angles spanned between the three outgoing edges then define the three unit angles U_1, U_2, and U_3, which span a spherical triangle on the unit sphere. The PTU kinematic model now provides the formalism to solve for the outgoing edges' driving angles ρ_j by means of spherical trigonometry. Hence the vertex folding motion is fully defined.

Therefore, if we are given the driving angle ρ_0 and a directed acyclic graph as a crease pattern, we can sequentially calculate all unknown folding angles. This only involves forward kinematics and is thereby computationally cheaper than other methods that have to rely on matrix inversions (Tachi, 2010). We refer an interested reader to Zimmermann et al. (2020) for the details of the kinematic model and focus on the graph construction here.

The Origami Game Given the constraints of the PTU method, we aim to formulate the task of finding a suitable crease pattern s as a discrete optimization problem. More specifically, we sequentially construct a directed acyclic graph $G(V, E)$ by placing vertices v and connecting crease lines e on a grid board of fixed size. The board represents the unfolded paper and the discrete nature of the grid allows for efficient exploration of the space of possible crease patterns.

In this formulation, we can also view the task of designing a crease pattern as playing a single-player game in which the player sequentially places vertices on the board. The board is first initialized with a simple pre-selected starting pattern, such as a line or a square, for which the size and the position can be chosen by the agent. Then, in the first phase of each turn, the player chooses a vertex that has only incoming edges so far, i.e. an extendable vertex. In the second phase, the player then chooses three locations on the board, which define the three outgoing edges of the vertex chosen before. A new vertex is created in every chosen location that did not have a vertex so far and the player is asked again to choose the next vertex to extend. In the first phase of each turn, the player can also be given the option to choose a special *source* action instead of selecting an extendable vertex. This action places a new source vertex in an empty space on the board, which receives the same external driving angle ρ_0 as the original source vertex, thus keeping the 1-DOF constraint. We provide this option for an easier approximation of certain shapes in our shape approximation experiments.

The player reaches the end of an episode in the game if either (1) a special *terminate* action is played, (2) some non-foldable game state is encountered, or (3) there are no more actions available in the current game state. See Figure 2 for an illustration of the game.

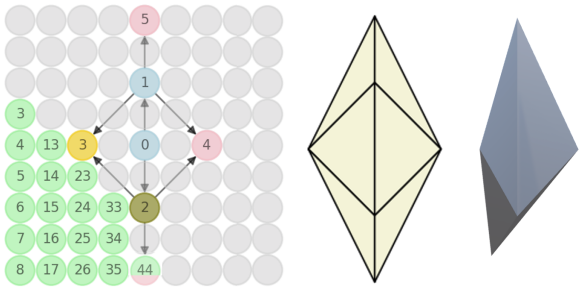


Figure 2: Origami game in a particular state (left: board, middle: creased paper, right: folded shape). The vertices on the board are color-coded as follows: Blue/Olive vertices have been extended, with the color indicating the rigid body mode M, pink vertices are extendable and the yellow vertex is currently selected for an extension. Green slots indicate permitted and the grey slots forbidden locations for the first outgoing edge endpoint – these indicate the available actions. Note that node 44 is colored pink and green, as it is an extendable vertex that can be selected as an endpoint.

Rules and Constraints Apart from an efficient search space discretization, our board game allows us to enforce certain rules and constraints given by the PTU kinematic model and the rigidity of the panels being folded.

Specifically, the PTU provides a kinematic model for the folding motion of extended vertices of arbitrary degree δ, given the following constraints: (1) The vertex has exactly three outgoing edges and $\delta - 3$ incoming edges. (2) The triangle inequality for spherical triangles must be fulfilled throughout the entire folding motion: $U_{min} + U_{med} \geq U_{max}$, where $U_{min}, U_{med},$ and U_{max} represent the smallest, middle and largest unit angle respectively. (3) The crease pattern $G(V, E)$ must be acyclic. Further, as we aim to fold rigid panels, we can infer that: (4) The crease pattern graph $G(V, E)$ must be planar, as crossing edges would result in overlapping panels in the flat folded starting state. And (5) Panels must not intersect with other panels during the entire folding motion.

All constraints 1-5 are either geometrical (1,2,4,5) or topological (3) in nature. In the discrete action space, we can impose constraints 1-4 through action masking, such that an agent is not eligible to take any action that would lead to a violating state. The non-intersection of faces constraint 5, however, cannot be constrained prior to the computation of the kinematics. Hence this constraint can only be evaluated after the full extension of a vertex.

In detail, we impose the constraints 1-5 as follows: (1) By extending each vertex with exactly three child vertices. (2) By computing the triangle inequality for spherical triangles after the second and before the third extension of a vertex for every permitted action and masking accordingly. (3) By masking extended vertices from the available actions. (4) By performing an edge-edge intersection test for available actions and masking accordingly. (5) By performing Moeller's Fast Triangle-Triangle Intersection Test (Möller, 1997) for a set of folding angles that mirror the folding motion. If panels intersect, we roll back to the last foldable state (the state before the failed extension) and terminate the episode.

Domain Knowledge Additionally to the constraints outlined above which guarantee rigid foldability, our formulation also allows incorporating domain knowledge as additional constraints to guide the search. Specifically: (a) The board size can be chosen based on the desired complexity of the resulting shape. Smaller board sizes yield simpler solutions while larger board sizes allow for more complex shapes. (b) Desired symmetries of the resulting shape can be incorporated by duplicating actions across symmetry axes. (c) If a good starting pattern is known, the board can be seeded with it and extended from there. (d) Finally, we can easily mask actions based on a maximal permitted crease length.

Objectives An important advantage of the formulation as an optimization problem is the freedom to choose the objective function. Instead of defining how the pattern should be folded, we can now focus on what we want from the resulting pattern and leave the implementation to the optimization. Specifically, given the general objective $\arg\max f(s, \rho_0)$ we can design the objective function $f : \mathcal{S} \times (-\pi, \pi) \to \mathbb{R}$ to reflect our requirements.

We formulate the objective functions in light of the sequential construction of a solution. That is, we seek to reward every partial solution at time step t with r_t, such that $f(s_T, \rho_0) = \sum_{t=0}^{T} r_t$. Here, T denotes the time step in which the episode terminates. For general objectives f we can always achieve this by setting $r_T = f(s_T, \rho_0)$ and $r_{t \neq T} = 0$. However, both RL and tree search methods benefit from early feedback. Specifically, if we wish to cut a branch in a game tree traversal at the partial solution $s_{t'}$, we can do so if (1) $r_t \leq 0 \; \forall t$, that is, any partial solution can only get worse[2] and (2) $\sum_{t=0}^{t'} r_t < f(s^*, \rho_0)$, where s^* is the best solution so far.

In a traditional inverse origami setting, the reward would measure how closely the folded pattern matches a given shape. We cover this shape approximation setting in Appendix C. However, if the reward function is more abstract and does not constrain the agent to matching a given shape as close as possible the agent can create (imagine) new shapes that accomplish the goal specified by the objective

[2]Note that this also implies that $f(s, \rho_0) \leq 0$ for any state s and driving angle ρ_0. However, this can often be easily achieved by subtracting a constant that upper bounds the original objective function and using the result as an objective function.

function. We sketch four kinds of handcrafted abstract objective functions to showcase the power of this approach. Specifically, starting from a flat sheet in the xy-plane at $z = 0$, we aim to fold everyday objects: (1) Bucket: To get a waterproof bucket, we maximize the smallest z-coordinate of the leaf nodes in the graph. (2) Shelf: Here we maximize the area of parallel planes in the folded shape, discarding solutions with less than three parallel planes. (3) The table objective aims to get 4 points to a target z value (the legs) while keeping all other points in the $z = 0$ plane. (4) Chair: A chair that is symmetrical across the y-axis consists of a backrest (target z and y value) as well as at least three legs on the ground (also with target z and y values). More details and the exact formulation of all objectives are provided in Appendix E. How objectives are formulated in the case of shape approximation can be seen in Appendix C.

Driving Angle Optimization The fitness of the final folded shape greatly depends on the driving angle ρ_0. Unfortunately, the optimal angle is usually not known before trying to fold the shape. In order to not constrain the agent to finding a pattern that is optimal given some particular driving angle, we instead specify a maximum driving angle $\rho_0^{\max}$ and keep track of 10 equally spaced driving angles from $0°$ to $\rho_0^{\max}$. During each reward calculation, we calculate the reward for each of the driving angles and give the highest of those rewards to the agent. If at any point a driving angle results in an intersection we discard it. If there are no more driving angles that do not cause an intersection the episode terminates. Note that this optimization over the driving angle is optional since we can also provide ρ_0 if we know a good value upfront. However, since we intend to highlight the general applicability of our approach in our experiments we include this optimization by default.

Search Methods As with the objectives, our formulation as a discrete optimization problem also allows for a diverse range of search methods. We evaluate various optimization methods including branch and bound searches, reinforcement learning, and an evolutionary algorithm in Appendix C on the traditional inverse origami problem of shape approximation. In Section 3 we focus on results achieved by our evolutionary algorithm as it proved to have overall the best performance in Appendix C.

For the evolutionary algorithm (EVO) we model agents as a simple list of action values, where the list length is proportional to the board size. The first half of the list is used in the node selection phase and the other is used in the extension phase. Actions are chosen greedily with respect to the listed values of the available actions. The best-performing agents reproduce, such that the next generation consists of randomly perturbed copies of these agents as well as some new randomly initialized agents (newcomers).

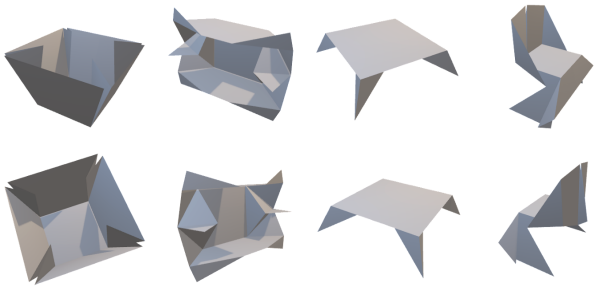


Figure 3: Solutions found on the abstract objectives. From left to right: bucket, shelf, table, chair. The two rows display two viewing angles.

3. Results

Here we focus on a novel shape imagination task described in Section 2 where the agent has to generate a crease pattern which when folded performs some abstract function described by the objective.

For this task, we set the board size to a moderate 13×13 and run our evolutionary algorithm for 500 thousand interactions with the environment. Our environment is implemented in Open AI Gym (Brockman et al., 2016). Experiment details are given in Appendix F.

The resulting folded shapes of the best patterns found for each abstract objective are visualized in Figure 3. See Appendix H for a visualization of the crease patterns and their folding motions. Our setup allowed us to find interesting shapes that reflect well our specifications. Importantly, they can be folded from a flat surface with only a single degree of freedom. This makes them easy to store or transport as they are flat when not folded and easy to fold as due to the single degree of freedom there is a single continuous motion that leads to the folded shape. These results show that there is a lot of potential in using these abstract objectives instead of plain shape approximation of existing objects.

Results for the more traditional shape approximation task can be found in Appendix C. They show that our environment can be combined with various search methods.

4. Conclusion

In this paper, we have described an environment for the iterative search of rigidly foldable crease patterns. We discretize the search space to get an efficient model for exploration and constraint imposition. We showcase its utility in designing novel function-based shapes and traditional shape approximation tasks. The proposed rigid origami design environment could serve both, research into origami pattern design for engineering applications and the application of novel optimization algorithms to origami design.

References

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Callens, S. J. and Zadpoor, A. A. From flat sheets to curved geometries: Origami and kirigami approaches. *Materials Today*, 21(3):241–264, 2018. ISSN 1369-7021.

Chaslot, G., Bakkes, S., Szita, I., and Spronck, P. Monte-carlo tree search: A new framework for game ai. *AIIDE*, 8:216–217, 2008.

Demaine, E. D. and Tachi, T. Origamizer: A practical algorithm for folding any polyhedron. In *33rd International Symposium on Computational Geometry (SoCG 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Demaine, E. D., Fekete, S. P., and Lang, R. J. Circle packing for origami design is hard. In *Origami 5: Fifth International Meeting of Origami Science, Mathematics, and Education*, pp. 609–626. CRC Press, 2016.

Dudte, L., Vouga, E., Tachi, T., and Mahadevan, L. Programming curvature using origami tessellations. *Nature Materials*, 15, 01 2016.

Hanna, B. H., Lund, J. M., Lang, R. J., Magleby, S. P., and Howell, L. L. Waterbomb base: a symmetric single-vertex bistable origami mechanism. *Smart Materials and Structures*, 23(9):094009, 2014.

He, Z. and Guest, S. Approximating a target surface with 1-dof rigid origami. In *Origami 7: The Proceedings from the 7th International Meeting on Origami in Science, Mathematics, and Education, Volume 2, Tarquin Publications (2018), 505-520*, volume 2. Tarquin, 2018.

He, Z. and Guest, S. D. On rigid origami i: piecewise-planar paper with straight-line creases. *Proceedings of the Royal Society A*, 475(2232):20190215, 2019.

Kawasaki, T. On the relation between mountain-creases and valley-creases of a flat origami. In *Proceedings of the First International Meeting of Origami Science and Technology, 1991*, 1991.

Kuribayashi, K., Tsuchiya, K., You, Z., Tomus, D., Umemoto, M., Ito, T., and Sasaki, M. Self-deployable origami stent grafts as a biomedical application of ni-rich tini shape memory alloy foil. *Materials Science and Engineering: A*, 419(1):131–137, 2006. ISSN 0921-5093.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pp. 3053–3062. PMLR, 2018.

Maekawa, J. and Kasahara, K. *Viva! Origami*. Sanrio, Tokyo, Japan, 1983.

McAdams, D. A. and Li, W. A novel method to design and optimize flat-foldable origami structures through a genetic algorithm. *Journal of computing and information science in engineering*, 14(3), 2014.

Meloni, M., Cai, J., Zhang, Q., Sang-Hoon Lee, D., Li, M., Ma, R., Parashkevov, T. E., and Feng, J. Engineering origami: A comprehensive review of recent applications, design methods, and tools. *Advanced Science*, pp. 2000636, 2021.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Miura, K. *Proposition of pseudo-cylindrical concave polyhedral shells*. Institute of space and aeronautical science, University of Tokyo Tokyo, 1969.

Möller, T. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

Morgan, J., Magleby, S. P., and Howell, L. L. An approach to designing origami-adapted aerospace mechanisms. *Journal of Mechanical Design*, 138(5), 2016.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287, 1999.

Resch, R. The design and analysis of kinematic folded plate systems. In *Proceedings of IASS Symposium on Folded Plates and Prismatic Structures, 1970*, 1970.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe,

D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 0028-0836.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018. ISSN 0036-8075.

Simm, G., Pinsler, R., and Hernández-Lobato, J. M. Reinforcement learning for molecular design guided by quantum mechanics. In *International Conference on Machine Learning*, pp. 8959–8969. PMLR, 2020a.

Simm, G. N., Pinsler, R., Csányi, G., and Hernández-Lobato, J. M. Symmetry-aware actor-critic for 3d molecular design. 2020b.

Tachi, T. Freeform variations of origami. *Journal for Geometry and Graphics*, 14, 01 2010.

Turner, N., Goodwine, B., and Sen, M. A review of origami applications in mechanical engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230(14):2345–2362, 2016.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782): 350–354, Nov 2019. ISSN 1476-4687.

Wang, M. *An Origami-Based Soft Actuator and the Application as A Soft Gripper*. PhD thesis, The George Washington University, 2019.

Yoshimura, Y. On the mechanism of buckling of a circular cylindrical shell under axial compression. *NACA Tech. Note No. 1390.*, 1955.

Zimmermann, L. and Stanković, T. Rigid and flat foldability of a degree-four vertex in origami. *Journal of Mechanisms and Robotics*, 12(1):011004, 2020.

Zimmermann, L., Shea, K., and Stanković, T. Conditions for rigid and flat foldability of degree-n vertices in origami. *Journal of Mechanisms and Robotics*, 12(1): 011020, 2020.

A. Related Work

In its most primitive form, a crease pattern consists of a single, center vertex and multiple leaf vertices connected to the center vertex. The fundamental origami theorems of Maekawa & Kasahara (1983) and Kawasaki (1991) determine flat foldability of such a primitive pattern, based on two characteristics of the edges: their direction of folding (the mountain-valley assignment), and their spanned planar angles (the sector angles). The foldability of multi-vertex patterns, however, has been proven to be an NP-hard problem (Demaine et al., 2016).

Nevertheless, with the Origamizer, Demaine & Tachi (2017) propose the state-of-the-art algorithm for solving the inverse origami problem for arbitrary target shapes. In particular, solving the inverse problem means being able to generate a crease pattern, which when folded, approximates the surface of the target up to a fixed precision. As earlier pointed out by He & Guest (2018), the Origamizer has remarkable abilities to approximate complex surfaces, but also limitations regarding our scope of rigid origami along with the extended problem definition: the generated patterns have many DOF, require a sequential folding process and favor tugged panels when approximating non-developable surfaces.

It has not yet been fully understood how to design rigidly foldable patterns with limited DOF (He & Guest, 2018; Turner et al., 2016), which are more suitable for engineering applications. Thus, other state-of-the-art methods approach shape approximation from a different paradigm by modifying a few well-known (rigidly foldable) tessellations, such as Miura Ori, Waterbomb-, Yoshimura-, and Resch's pattern (Miura, 1969; Hanna et al., 2014; Yoshimura, 1955; Resch, 1970). Their properties are summarized and described by Tachi (2010). In comparison to the Origamizer, methods based on known tessellations (Tachi, 2010; He & Guest, 2018; Callens & Zadpoor, 2018; Dudte et al., 2016) have an inherent rigid foldability and limited DOF. However, the main limitation of these methods is that they rely on the modified tessellations with a highly constrained design space. This leads to some authors restricting themselves to approximating particular kinds of surfaces such as cylindrical ones (Dudte et al., 2016). As pointed out by Turner et al. (2016); Meloni et al. (2021), the future prospect of origami for engineering applications relies on methods for the design of new and original rigid foldable patterns with properties specific to their field of application.

A general design method for rigidly foldable patterns has not yet been proposed (Meloni et al., 2021; Turner et al., 2016). Nevertheless, the recently proposed Principle of Three Units (PTU) method (Zimmermann et al., 2020; Zimmermann & Stanković, 2020) allows the rule-based generation and efficient simulation of rigidly foldable patterns. The PTU implies a set of rules, such that a pattern complying with those rules is guaranteed to fold rigidly. Although the PTU does have limitations, in particular a restriction to acyclic origami graphs, it still enables the rule-based generation and efficient folding simulation of rigidly foldable patterns in a large configuration space.

We seek to formulate the graph design problem in a reinforcement learning (RL) setting to leverage the spectacular success of RL methods on various problems with large search spaces (Silver et al., 2016; 2018; Vinyals et al., 2019). RL has also been applied to various design tasks, such as designing molecules (Simm et al., 2020a;b) or designing computer chips (Mirhoseini et al., 2021). However, to the bests of our knowledge, there have been no attempts so far to apply RL to rigid origami pattern discovery.

Whereas the majority of computational origami design methods focus on the efficient and accurate approximation of shapes, design methods for embodied functional properties, such as a given surface area and height of the folded shape, have not received much focus. In a very basic example of this, a genetic algorithm has been used to discover a simple pattern that folds a square sheet into a smaller flat sheet that has a predetermined area using a limited number of crease lines (McAdams & Li, 2014). We explore a similar direction but open up the design space to a much wider range of applications.

B. Search Methods

Here we discuss the search methods considered in our experiments. Specifically, we compare the following approaches in the shape approximation task:

- Random search (RDM)
- Depth-first tree search (DFTS)
- Breadth-first tree search (BFTS)
- Monte Carlo tree search (MCTS)
- Proximal policy optimization (PPO)
- Evolutionary search (EVO)

All methods act within the *playable area* B of the board. That is, we limit the area in which the agent can act on the board according to the symmetries defined: If we consider the y axis as a symmetry line, we limit the playable area to the left half of the board. If additionally to the symmetry across the y axis we also enforce symmetry across the x axis, we limit the playable area to the top left quarter of the board. The additional xy symmetry axis used for the bowl, bucket, and table is enforced with the corresponding action masking that shrinks the playable area to $\sim 1/8$ of the original board size. Note that we keep the board positions on the symmetry axes within the playable area.

B.1. RDM

The random baseline simply samples in every step t an action a_t uniformly from the set of available actions in the state s_t. That is, as in all methods, we mask actions according to constraints and domain knowledge (see Section 2) and sample from the remaining actions. In Figure 2 we visualize an example of this action set in green.

B.2. DFTS

To explore a larger part of the search space, we run 10 searches of 500'000 steps each. In each trial, we start by randomly playing a game until the end and storing all states that were visited along the way. From the final state, we back up and sample a new action in the penultimate state, traversing the game tree in a depth-first manner. Throughout the search, we keep track of the best return seen so far and immediately back up from a state if the reward trace leading to it sums to less than the best-seen return. To further encourage exploration, we limit the number of children sampled in each state to 10, backing up once this threshold is crossed.

B.3. BFTS

The setup for breadth-first tree search is similar to the depth-first tree search: we run 10 searches of 500'000 steps each, backing up if either the current reward trace is less than the best seen so far or the branching threshold of 10 explored children is crossed. However, instead of traversing the tree depth-first, we traverse it breadth-first. That is, in every new state we first sample up to 10 actions uniformly at random, from the available ones, evaluate the states which result from performing these actions and then move to the state which yielded the highest evaluation. If we back up to a visited state, we move to the child that scored second-highest and so on.

B.4. MCTS

The Monte Carlo Tree Search algorithm by Chaslot et al. (2008) is divided into four phases: selection, expansion, simulation, and back-propagation. This algorithm has been successfully applied to game problems (Silver et al., 2018). We augment the implementation provided by Liang et al. (2018) with the following default configuration settings: For each step in the actual environment, 100 full episode simulations are performed. During the simulation phase, we traverse the tree by sampling available actions from a uniform distribution, which is distorted by an additive Dirichlet noise of $0.03 * \mathrm{Dir}(0.25)$. For the traversal in the actual environment, we follow a greedy policy, by always selecting the most visited child. The simulation game tree is preserved throughout an episode, such that the following simulations can build on the previous findings. During the back-propagation phase, the reward is normalized as follows: $r_{norm} = \frac{2 \cdot r_t}{r_{min}} + 1$, where r_{min} is the penalty received in a non-foldable state. Since the rewards in the shape approximation are upper bounded by zero, this ensures that $r_t \in [-1, 1]$ for all t. Additionaly we apply the following settings: $temperature = 1.5$ and a coefficient $c_{puct} = 1.0$ to balance exploration and exploitation.

B.5. PPO

Proximal policy optimization (Schulman et al., 2017) is a policy gradient-based deep reinforcement learning approach that optimizes a surrogate objective inspired by trust-region optimization (Schulman et al., 2015).

We use the implementation provided by Liang et al. (2018), modeling the agent as a simple multilayer perceptron with two layers of 256 neurons each. We keep most hyperparameters at the default values and only adjust some to fit our hardware. That is, we set the SGD mini-batch size to 64 and the training batch size to 2'050. We also set the entropy coefficient to 0.01 (default is 0) to encourage better exploration.

As input to the agent we provide an encoding of the state s in a third order state tensor $O_{i,j,k} = \{-1, 0, 1\}^{w \times h \times (wh+2)}$, where w is the board width and h is the board height. In more detail:

- $O_{i,j,k=0}$. The first slice of the elements encodes the rigid body mode $M \in \{-1, 1\}$, where i, j denotes the vertex position on the grid.
- $O_{i,j,k=n}$. The last slice indicates the current vertex i, j, which is selected for extension in the second phase.
- $O_{i,j,k=1,\ldots,k=n-1}$. In addition to their position, vertices are enumerated along k in order of their appearance. For a given vertex position i, j, the vector $O_{i,j,k=1,\ldots,wh}$ encodes the vertex adjacency. The values $\{-1, 1\}$ indicate incoming, or outgoing edges respectively.

Similar to the other approaches we mask invalid actions from the probability distribution output over the action space.

B.6. EVO

In the evolutionary algorithm, we keep track of a population of 128 agents, each represented by a list of action values $\mathbf{q} \in \mathbb{R}^m$ where $m = 4d$ and d is the size of the playable area. To give an example, the playable area for a 13×13 board with the x and y axis as symmetry lines has a size of $d = 7^2 = 49$. The size of the action value vector m is

$4 \cdot d$ to have $2d$ action values for the node selection phase and $2d$ action values for the node extension phase, where factor 2 accounts for the selection of the rigid body mode M. As such, every action value corresponds to a specific action that becomes available or masked depending on the state of the game. Agents act by always choosing the action which has the highest value among the available actions.

At initialization, we sample each agents' value vector $\mathbf{q}$ from an m-dimensional isotropic normal distribution $\mathbf{q} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ where $\mathbf{I}$ represents the $m \times m$ identity matrix. In each iteration of the evolution, we then

1. evaluate each agent in the population by playing out an episode. Here we set the evaluation of agents which yielded the same action sequence as another agent in the current generation to $-\infty$. This is done to encourage diversity. We then
2. order agents according to their achieved evaluations
3. take the top 25% as parent population for the next generation, discarding the rest
4. mutate the parents by adding a small random perturbation $\epsilon_1 \sim \mathcal{N}(\mathbf{0}, \sigma_1 \mathbf{I})$
5. copy the mutated parents twice as offspring, where each copy is subject to another additive random perturbation with $\epsilon_2 \sim \mathcal{N}(\mathbf{0}, \sigma_2 \mathbf{I})$ and $\epsilon_3 \sim \mathcal{N}(\mathbf{0}, \sigma_3 \mathbf{I})$ respectively.
6. Fill up the remaining 25% of the population with newcomers $\mathbf{q} \sim \epsilon_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and start the next iteration (go back to 1).

We iterate until the total sum of steps taken by all agents crosses the predefined step limit of 500 thousand interactions.

The perturbation hyperparameters were set to $\sigma_1 = 0.1$, $\sigma_2 = 0.5$ and $\sigma_3 = 1.0$. These hyperparameters, as well as the population size of 128 agents and the threshold at 25% were initial guesses that worked out well enough. We did not tune these hyperparameters in any way, therefore results might even improve for a different set of hyperparameters.

C. Shape Approximation

To validate our formulation, we compare all search methods in four shape approximation tasks (Figure 4).

The objective of the shape approximation (inverse origami problem) is to find a crease pattern s such that the folded polyhedral manifold $\mathcal{M}_{origami} \subset \mathbb{R}^3$ approximates a given target shape $\mathcal{M}_{target} \subset \mathbb{R}^3$. As a measure for the accuracy of approximation, one commonly takes the Hausdorff distance $d(X, Y)$ between sets of points X and Y given by

$$d(X, Y) = \max\{d_{\rightarrow}(X, Y), d_{\leftarrow}(X, Y)\}$$

$$d_{\rightarrow}(X, Y) = \max_{x \in X} \left(\min_{y \in Y} ||x - y||_2 \right)$$

$$d_{\leftarrow}(X, Y) = \max_{y \in Y} \left(\min_{x \in X} ||x - y||_2 \right)$$

We fix a set of sampled target points $Y \sim \mathcal{M}_{target}$ and choose $X \subset \mathcal{M}_{origami}$ as detailed later. Note that the two terms $d_{\rightarrow}(X, Y)$ and $d_{\leftarrow}(X, Y)$ model two distinct aspects: The first measures how far the folded shape is from the target shape while the second measures how well the whole target shape is covered by the folded shape. As we only add points during the construction of a crease pattern, the furthest point from the target in the folded shape can only get worse. We can therefore set $\Phi(s_t, \rho_0) = -d_{\rightarrow}(P_t, Y)$, where $P_t \in \mathbb{R}^{3 \times |V_t|}$ denotes the spatial positions of the vertices V_t in s_t under driving angle ρ_0. We can use the vertex positions here as they reflect the corners and thereby the extremes of the folded polyhedran. The rewards are then given by $r_t = \Delta\Phi_t$ for $t \neq T$ and

$$r_T = -\max\{d_{\rightarrow}(P_T, Y), d_{\leftarrow}(X, Y)\} - \sum_{t=0}^{T-1} r_t$$

For r_T we sample X from the folded shape.

Generally, we can shape rewards as $\tilde{r}_t = r_t + \Delta\Phi_t$ by adding a difference in potential $\Delta\Phi_t = \Phi(s_t, \rho_0) - \Phi(s_{t-1}, \rho_0)$ without changing the objective (Ng et al., 1999). However, to fulfill requirement (1) for game tree pruning from Section 2 we seek a potential function Φ that is monotonically decreasing as the game progresses, i.e. $\Phi(s_t, \rho_0) \leq \Phi(s_{t-1}, \rho_0)$. Luckily, many objectives in our setup, including the shape approximation objective, can be naturally decomposed into a part that is monotonically decreasing with t and a remainder, which can be given at the end of the episode. Note that rewards can be augmented with certain desiderata such as a term that discourages points from sticking too far out. This augmentation can also be used for reward shaping with the same argumentation as before, points only get added, so the worst point can only get worse.

The four target shapes in Figure 4 were chosen to reflect distinct surface characteristics:

1. First we aim at a proof of concept. To this end, we chose two simple target shapes, a pyramid, and a cube. These targets are benevolent to our method since: (a) their symmetry allows us to shrink the search space, and (b) their surfaces are developable if cut open along the corners, i.e. we know that there exist solutions that can close to perfectly approximate these targets.

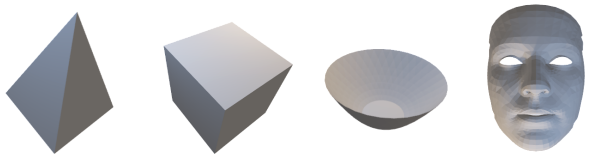


Figure 4: Visualization of the four target shapes used in the shape approximation task: cube, pyramid, bowl, and human face.

Target	Board Size	Start Pattern	Symmetry Axes	cl_{max}
pyramid	(9,9)	square	x,y	∞
cube	(9,9)	square	x,y	∞
bowl	(25,25)	square	x,y,xy	2.9
human face	(25,25)	single crease	y	2.9

Table 1: Experimental settings overview: we impose domain specific constraints on the four target shapes, regarding board size, symmetry, initial configuration and the maximal permitted length of crease lines cl_{max}.

Hence we expect that there exists an optimal pattern that can be modeled within the scope of our discrete environment and the PTU-based model.

2. At the second stage we address the question of approximating a non-developable surface, where the existence of an optimal pattern is not obvious. Here we try to approximate the curved surface of a bowl target shape. The approximation of curved shapes by means of modified rigid foldable tessellations is common in the literature (Dudte et al., 2016; He & Guest, 2018). Note that our setup here is more challenging, as we limit ourselves to a single DOF and acyclic crease pattern graphs.

3. At a third stage we aim to find the limitations of our formulation as we try to approximate the shape of a human face. Note that besides the complex topography, this target also only allows for one symmetry axis.

The experimental settings for shape approximation vary slightly between the targets but remain strictly constant throughout an experiment, i.e. all search methods face the same setup per target. As outlined in Section 2, domain knowledge may be imposed as additional target-specific constraints. An overview of the experimental settings across the four target shapes is provided in Table 1. See Appendix D for a more detailed explanation of these settings.

We ran each search method for 500 thousand interactions with the environment, always keeping track of the best pattern found so far. We evaluate the following performance measures to compare the search methods:

1. the return $f(s^*, \rho_0)$ of the best pattern s^* found

	Pyramid	Cube	Bowl	Human Face
RDM	$\mathbf{-0.09 \pm 0.00}$	-0.76 ± 0.13	-1.20 ± 0.16	-4.98 ± 0.35
DFTS	$\mathbf{-0.09 \pm 0.00}$	$\mathbf{-0.07 \pm 0.00}$	-0.85 ± 0.08	-4.38 ± 0.47
BFTS	$\mathbf{-0.09 \pm 0.00}$	-0.07 ± 0.01	$\mathbf{-0.75 \pm 0.05}$	-7.94 ± 0.16
MCTS	$\mathbf{-0.09 \pm 0.00}$	-0.79 ± 0.04	-1.09 ± 0.09	-5.01 ± 0.32
PPO	-0.09 ± 0.01	-0.68 ± 0.28	-1.24 ± 0.30	-5.70 ± 0.61
EVO	-0.17 ± 0.25	-0.44 ± 0.30	-1.09 ± 0.25	$\mathbf{-3.49 \pm 0.61}$

Table 2: Summary of best episode returns $f(\rho_0, s^*)$ across shapes and search methods. The highest returns are highlighted in bold.

2. the total number of environment-interactions taken to reach the best pattern

Since all our search methods are randomized algorithms, we run all searches for ten different random seeds and report the mean and the standard deviation. The results can be seen in Tables 2 and 3. Qualitatively, we visualize the folded shapes of the best patterns found in any of the runs in Figure 5. The best shape approximation crease patterns can be seen in Figure 8 and their folding motion can be seen in Figure 9.

The details on the shapes are as follows:

- Pyramid. The pyramid is defined by a polygonal mesh, spanned by a set of five vertices: $\{(2,2,0),(-2,2,0),(-2,-2,0),(2,-2,0),(0,0,\sqrt{8})\}$.
- Cube. A cube with edge length $b = 2$, as expressed in units of the discrete board grid.
- Bowl. The curved surface of the bowl is defined by a paraboloid: $z = (0.2x^2 + 0.2y^2 - 0.6)$, for $\sqrt{x^2+y^2} \in (1.6, 5.0)$. The bottom of the bowl ($\sqrt{x^2+y^2} < 1.6$) is an adjacent circular disk of radius 1.6.
- Human Face. The polygonal mesh of a human face, slightly edited (trimmed in the depth dimension) from Apple ARKit.[3]

Figure 5 shows that all methods are able to find the best possible approximation of the pyramid, where the slight opening at the top is an artifact of the fold angle discretization. However, for the cube, some methods struggle to find the optimal pattern. This highlights the difficulty of the optimization problem, as many crease patterns lead to well-separated local optima. Moving to the bowl we see that all methods now struggle to find a perfect approximation. We note that DFTS and BFTS perform comparably well here, as they can leverage local optima to prune many unpromising paths. From the quantitative results in Table 2 we see that especially DFTS and BFTS struggle to find any decent

[3]Apple ARKit https://developer.apple.com/augmented-reality/arkit/

	Pyramid	Cube	Bowl	Human Face
RDM	130 ± 115	73 ± 89	249 ± 187	193 ± 93
DFTS	55 ± 29	96 ± 92	371 ± 99	256 ± 112
BFTS	83 ± 24	94 ± 48	380 ± 62	335 ± 145
MCTS	22 ± 20	159 ± 169	233 ± 136	314 ± 143
PPO	13 ± 4	151 ± 162	83 ± 109	297 ± 237
EVO	59 ± 68	236 ± 150	177 ± 121	376 ± 125

Table 3: Environment-interactions (in thousands) to find the best pattern.

approximation. We believe that the larger board makes the search space too large to explore within the 500 thousand steps. This is further exaggerated in the results on the human face approximation, highlighting the need for a broader exploration that is present in the other search methods. Finally, the results from the human face approximations show that an evolutionary algorithm performs best if the search space becomes excessively large. We believe this stems from the principled exploration of multiple promising solutions in the evolutionary algorithm.

During the evaluation we also kept track of the total number of environment interactions taken to reach the best pattern and report those in Table 3. PPO and MCTS tend to find their best pattern quite quickly, but sometimes only find a local minimum (see Table 2).

All of the experiments were performed on a server with two 10-core Intel Xeon E5-2690 v2 CPUs and 125 GB of RAM.

Summarizing all results, we note:

1. If the search space is moderately large, tree-based search algorithms (i.e., DFTS and BFTS) perform well. However, these methods can become excessively expensive when the search space becomes larger and their initial exploration fails to make fast progress. In particular, BFTS struggles if the non-monotonic remainder of the objective is large relative to the monotonic part which is used to cut branches.

2. Local optima are plentiful and well separated in the search space, leading policy-based methods (MCTS and PPO) to converge to a suboptimal solution prematurely. Note that the policy-based solutions generally arrive faster at their best solution, but in the end achieve worse accuracy.

3. The evolutionary algorithm strikes a balance between exploration and exploitation, achieving the best performance in the most complex domain of approximating a human face.

The characteristics of our search space make our application also interesting for optimization research and we see the application of more recent approaches such as GFlowNets (Bengio et al., 2021) as a promising direction to further explore the possibilities here.

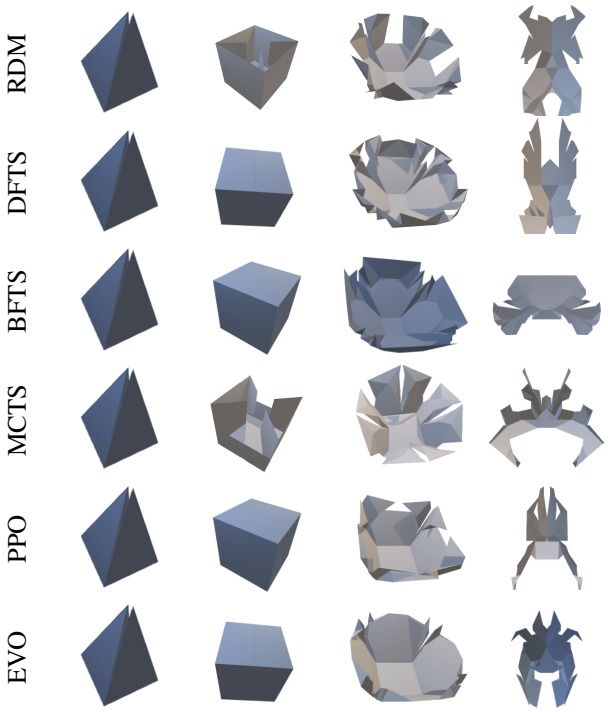


Figure 5: The figure shows the best approximations found for each method over all of the ten runs, with the overall best approximations across all methods highlighted in blue.

D. Explanation of Shape-specific Settings

As our agents cannot observe the target shape, it is beneficial to indirectly tell them some important characteristics of the shape to ease the task. This is done by introducing additional constraints based on domain knowledge as discussed in Section 2. The list below provides a detailed explanation of the experimental settings that were summarized in Table 1:

1. The board size is chosen in relation to the size of the target, and the complexity of its surface topography.

2. We initialize the pyramid, cube, and bowl with a square initial pattern at the center of the board. We fix the size of the square for the pyramid and the cube according to the target shape. For the bowl we let the agent choose the size of the initial square.

3. The human face is initialized with a single crease line just above the tip of the nose.

4. Symmetry constraints are imposed according to the

symmetry of the targets. That is, for the pyramid and the cube we impose symmetry across the x and y axis, while for the face we only have symmetry across the y axis. For the bowl, apart from the symmetry across the x and y axis, we add an additional xy-axis symmetry.

5. We set $\rho_0^{\max} = \pi$. Since the agent is also free to choose the rigid body mode M of the source creases, this effectively gives it the ability to optimize over the full range $\rho_0 \in (-\pi, \pi)$.

6. For the pyramid, the maximum permitted crease length cl_{max} remains unconstrained.

7. For the cube, the maximum permitted crease length cl_{max} is bound by its edge length.

8. For the bowl and the human face we set a maximum permitted crease length such that the agent is encouraged to build more details.

9. Additionally, we adjust the reward signal r for the two closed surfaces of the pyramid and the cube as follows: any parts of the folded surface which are fully enclosed by the target surface are excluded from the reward computation. This is done as we normally do not care about the folds which lie inside the closed surface if the closed surface is our target.

E. Abstract Reward Details

Here we provide more details on the abstract reward functions used to specify the desiderata of the everyday objects. Since the evolutionary algorithm does not rely on intermediate rewards r_t for $t < T$, we mainly focus on the formulation of objective $f(s_T, \rho_0)$ and simply set $r_t = 0$ for $t < T$ and $r_T = f(s_T, \rho_0)$. Note however that many objectives can be decomposed and shaped if one wishes to use other search algorithms.

E.1. Bucket

For the waterproof bucket, we aim to maximize the z-value of the outermost vertices. That is, we take the vertices $v \in V_T$ which have not been extended (pink/yellow vertices in Figure 2) and use the minimum of their z values in the folded shape as a reward. To encourage compact shapes we only reward buckets whose max z value of the outer points is at most twice the min z value and whose points have a max norm in the xy plane which is at most twice the min norm in the xy plane.

E.2. Shelf

The idea behind the shelf reward is to maximize the area where one can place things on the shelf. To this end, we take the surface normals of the triangulated folded mesh and evaluate the pairwise cosine-similarity to see which surfaces are parallel to each other. We further calculate the distance between parallel surfaces to distinguish triangles from the same shelf level from triangles of different shelf levels. We do not reward shelves with less than three levels to exclude trivial solutions. For a shelf with at least three levels, we sum up the areas of the triangles within a plane and take the minimum of these surface areas as a reward.

E.3. Table

For the table we set a target z value of 2.5 and calculate the mean absolute distance between the z value of the 4 highest points and this target. For all other points we calculate the mean absolute difference between their z value and 0, to keep them in the table plain. The final reward is the negative sum of these mean distances, i.e.

$$f(s_T, \rho_0) = -\frac{1}{4} \sum_{p \in P_T^{max}} |p_z - 2.5| - \frac{1}{|V_T| - 4} \sum_{p \in P_T \setminus P_T^{max}} |p_z|$$

where P_T^{max} are the points with maximal z value and $|V_T|$ is the number of vertices/points in the graph.

E.4. Chair

To encourage compact solutions, we penalize solutions that have points in the folded shape with x or y coordinate bigger than 4, that is, any point outside an area that is 4 times as large as the starting square. We further penalize solutions that do not have their 3 lowest points at approximately the same height or have them only on one side of the board. This is to ensure that we have at least three legs that provide some sort of stability. We then categorize all points above the $z = 0$ plane as backrest points and calculate their mean absolute difference to a target y value of 2.1, i.e., they should lie just next to one edge of the starting square. Further we add the absolute difference between the z value of the highest point and 4 to encourage the backrest to achieve a certain height. Finally, the leg points, i.e., the 3 lowest points are also subject to corresponding target z and y values. Formally, we have

$$L_{legs} = \frac{1}{3} \sum_{p \in P_T^{legs}} |p_z - (-4)| + ||p_y| - 2.1|$$

$$L_{rest} = |p_z^{max} - 4| + \frac{1}{|P_T^{z>0}|} \sum_{p \in P_T^{z>0}} ||p_y| - 2.1|$$

$$f(s_T, \rho_0) = -L_{legs} - L_{rest}$$

Here P_T^{legs} is the set of the 3 points with the smallest z value, p_z^{max} is the largest z value among all of the points and $P_T^{z>0}$ is the set of points with a z value larger than 0. Note the slight abuse of notation: we use $|P_T^{z>0}|$ to denote the number of points in set $P_T^{z>0}$, while the other uses of $|\cdot|$ represent the absolute value function.

F. Shape Imagination Experiment Setup

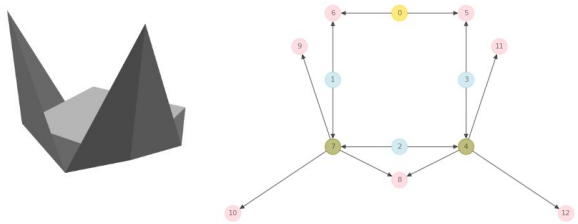


Figure 6: Seeding pattern for the chair fold. Left: the folded shape of the seeding pattern on its own. Right: The graph of the seeding pattern.

We here detail the starting configurations and symmetries enforced in the different shapes. Specifically, for all abstract rewards, we do not allow for additional source actions apart from the starting pattern. We start each pattern with 4 sources arranged in a square (as we also have in the first three shape approximation experiments), but let the algorithm choose the size of the square. Only for the chair do we fix the size of the starting square as the reward is based on the size of this square (which represents the area on which one sits, see Appendix E.4). Moreover, for the bucket and table, we enforce x, y, and xy symmetry axes. Note however that the xy symmetry here can also be omitted, see Appendix G for corresponding results. For the shelf, we enforce symmetry across the x and y axis. Note that the visualization in Figure 3 is rotated such that the parallel planes are horizontal (as they would be if one were to use it as a shelf). Also here we can drop one of the symmetry axes to get interesting alternative results, see Appendix G. Finally, for the chair, we only have one symmetry axis (the y axis). Here we also provide a seeding pattern as visualized in Figure 6. The intuition here is to flip a downward fold into upward spikes to provide a starting point for the backrest.

For all shapes except the chair, we optimize over the folding angle ρ_0. For the chair, we fix ρ_0 to the maximal folding angle admissible by the seeding pattern.

G. More Shape Imagination Results

Figure 7 shows more results on the shape imagination objectives for different symmetry constraints. The settings (apart from the symmetry) are the same as for the results in the main text. Only for the shelf, we reduced the board size to 9×9 to counter the search space increase resulting from the removed symmetry.

Figure 7: Additional results on the abstract objectives for different symmetry constraints. From left to right: Bucket with only x and y symmetry axes, shelf with only the y symmetry axis, table with only x and y symmetry axes.

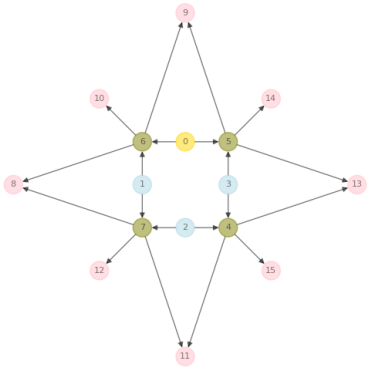
H. Furniture Folding

Figure 10 shows the crease patterns of the everyday objects designed by optimizing our abstract objectives. Meanwhile, Figure 11 provides a visualization of the folding motion of those crease patterns.

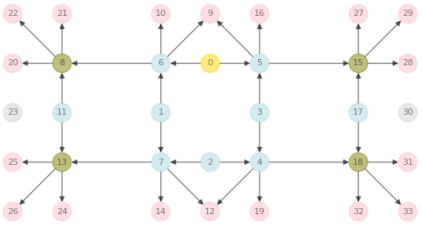
I. Limitations of the Environment

We note the following limitations in the current version of our environment:

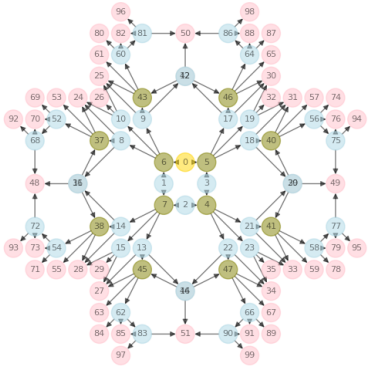
- The environment is limited to the scope of rigid foldable straight line crease patterns. Furthermore, their configuration space is limited to the scope of the PTU model: not all rigid foldable patterns can be modeled with the PTU (Zimmermann et al., 2020).
- The imposed discretization in our environment further limits the configuration space in comparison to the full continuous configuration space. However, with the board size, we can effectively trade the efficiency of the search off with the discretization errors that are introduced.



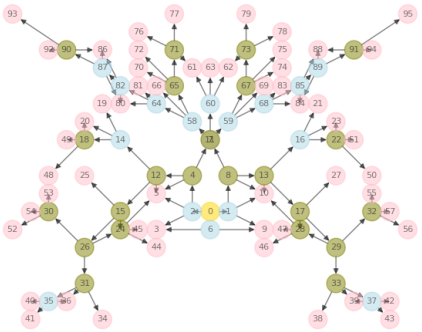
(a) Pyramid crease pattern



(b) Cube crease pattern



(c) Bowl crease pattern



(d) Face crease pattern

Figure 8: The best approximations for the four target shapes: (a) pyramid, (b) cube, (c) bowl, and (d) face.

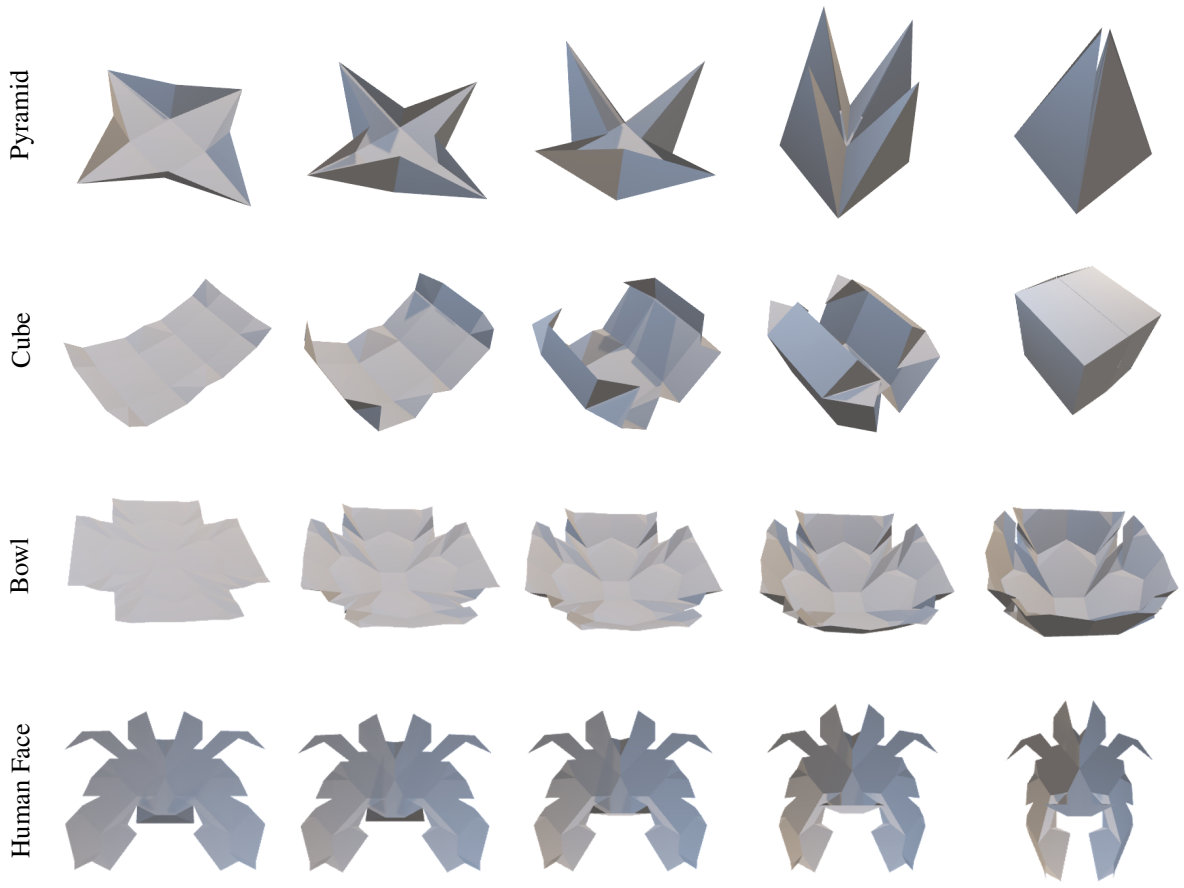
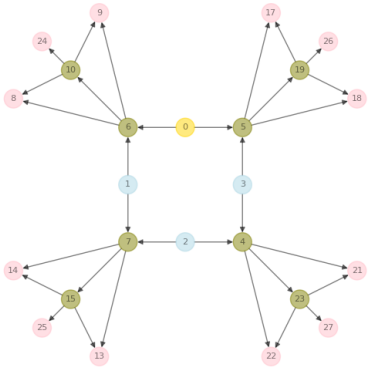
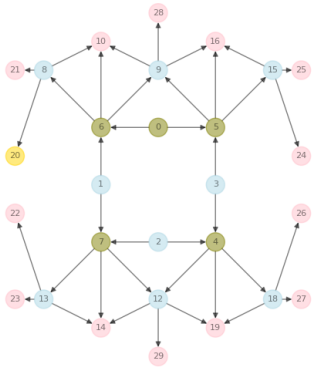


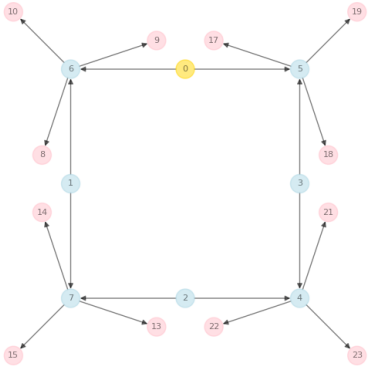
Figure 9: The folding motion of the best approximations for the four targets.



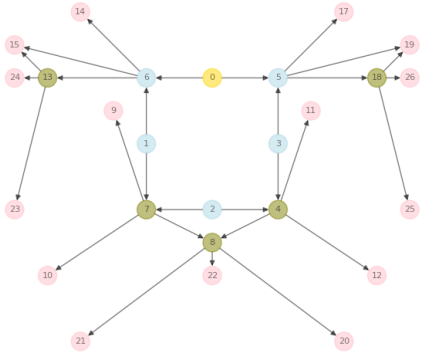
(a) Bucket crease pattern



(b) Shelf crease pattern



(c) Table crease pattern



(d) Chair crease pattern

Figure 10: The crease patterns of the imagined shapes: (a) bucket, (b) shelf, (c) table, and (d) chair.

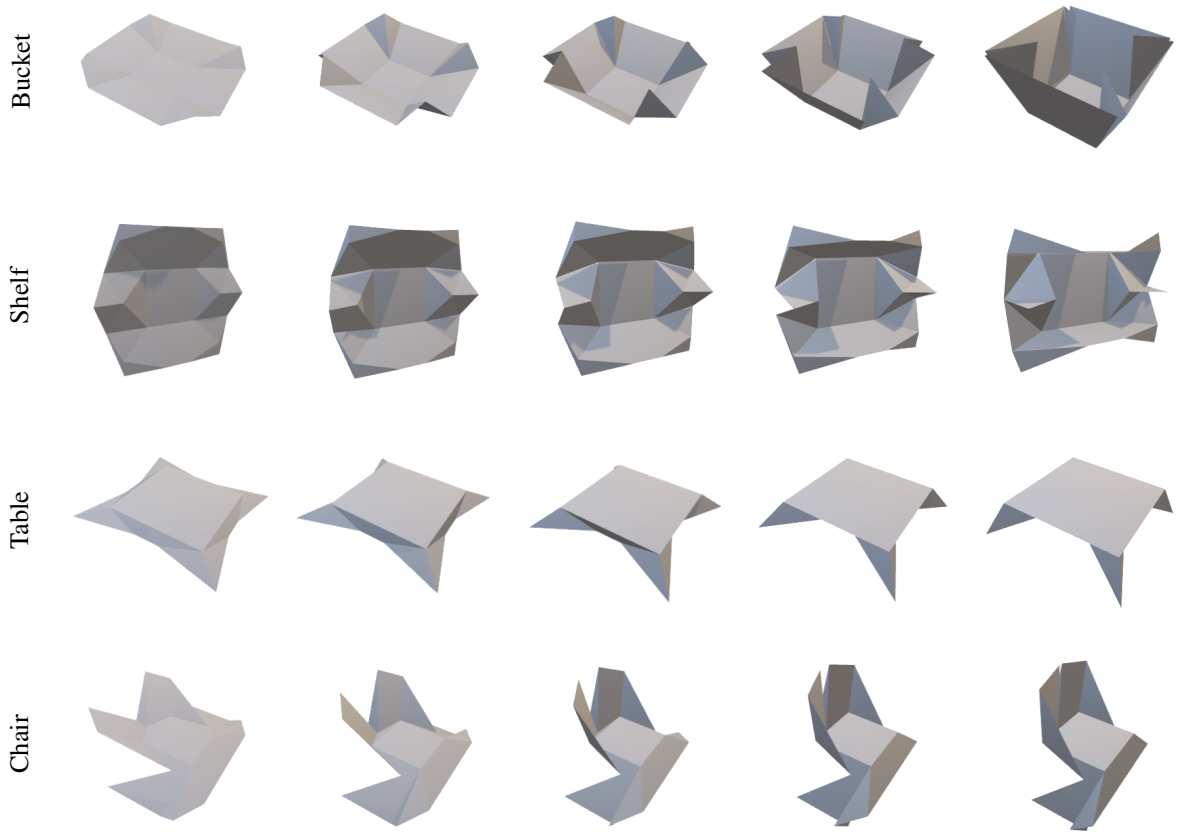


Figure 11: The folding motion of the imagined shapes: bucket, shelf, table, and chair.