Learn to Play Tangram with Graph-based Reinforcement Learning

Maximilian Langer, Kay Hansel, Niklas Funk

Abstract— **Tangram is a dissection puzzle with the goal of reconstructing a given shape using seven polygons of different shapes and sizes. Its inherent complexity has made it a famous game often played by children who therefore have to learn goal-defined geometric reasoning. However, its complexity also poses a challenge in constructing learning algorithms. The polygon-based composition of tangram makes it predestined for graph representations. We formalize a method to solve the game of tangram by decentral graph-based decision making using a multi-head attention graph representation. For learning, we utilize a reinforcement learning setting based on a PyBullet simulation environment**

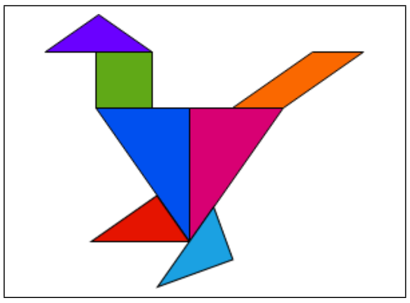
Fig. 1. Example of possible target shape in tangram.

I. Introduction

Solving the problem of autonomous robotic assembly promises a huge advance in the domain of manufacturing. Currently, even slight changes to the assembly target require human intervention. However, once robots gain capabilities to optimize assembly plans by themselves we can expect drastic increases in efficiency and reduction of cost. In general, the problem of autonomous robotic assembly lives in a continuous three-dimensional space. Consequently, one runs into the problem of high combinatorial complexity. If we are to transition into a two-dimensional space, the problem of combinatorial complexity is still present but weakened. This is where the game of tangram becomes promising. Simple enough, the rule of the game is to properly position a set of seven polygons such that they resemble a given target shape (see Figure 1). In this regard, as it is the case for autonomous assembly, solving tangram requires geometric reasoning under the guidance of a predefined goal. Additionally, its' polygon-based composition makes it predestined for graph-based approaches. Under the use of *graph neural networks* [5] (GNNs) we naturally incorporate a representation of the problem that depicts the need for geometric reasoning. Furthermore, we can conduct our studies under gradual increase of complexity, e.g., by varying the number of polygons present in the environment. If we are to find a proper representation of the problem of tangram that is generalizable to three-dimensional problems, corresponding learnings might benefit research in autonomous robotic assembly.

In this work, our focus is to formalize a concept to solve tangram based on *reinforcement learning* (RL) and graphs. To do this, we adjust a *multi-head attention graph representation* (MHA) [1] originally developed for autonomous robotic assembly. In contrast to deploying a decision making module per polygon, we propose a decentral decision making strategy to achieve the global goal of correct alignment of all polygons.

II. Related Work

Yamada & Batagelo [10] performed a comparative study on different methods to solve tangram. First, they evaluated a method utilizing heuristic programming proposed by Deutsch & Hayes [11]. The basic idea is to iteratively divide the target shape into multiple sub-problems until each sub-problem corresponds to a single polygon. Deutsch & Hayes define 10 rules using the geometric relationship of the edges to automatically dissect the target shape. Once the problem is successfully dissected into sub-problems that correspond to exactly 1 polygon each, it becomes easy to solve by trial and error. However, there is no guarantee for the algorithm to successfully dissect the problem. It is especially constrained to target shapes without holes and target polygons with specific rotations.

Another method evaluated in [10] is based on simple neural networks. Oflazer [12] proposed a representation of the position and rotation of polygons in the target shape by individual neurons. Although this approach is shown to solve common tangram targets, it assumes the target shape being placed on a regular grid and rotations of polygons as multiples of 45 degrees.

Finally, Yamada & Batagelo reviewed two techniques originally developed for solving jigsaw puzzles. Both techniques have extensions to the game of tangram. First, [14] utilizes genetic programming by representing polygons by string codes. Based on the string code representation and an iterative approach, a subset of target polygons are placed such that they lie inside the outline of the target shape while their boundaries coincide with the boundary of the target shape. The placed polygons now define a smaller subspace of the target shape. Further iterations try to fill the subspace using the remaining polygons. Again, this proposal assumes target shapes without holes. In [13], Kovalsky et al. represent tangram as an edge matching puzzle. They formulate a polynomial equation system to represent a specific problem in tangram. Their representation is based on a target shape and a definition of the corresponding polygons, assuming a fixed orientation of the individual polygons. The solution of the equation system yields the solution to the given problem. However, this approach can only deal with translations. Therefore, polygons are assumed to be given already in the final orientation. Kovalsky et al. argue that the approach can be extended to additionally support a discrete set of rotations.

Note, that all proposals evaluated in [10] pose strong assumptions on the structure of the target shape. Most important, none of the proposed techniques can solve target shapes consisting of multiple connected areas.

The first public appearance of a *deep learning* (DL) model that has successfully learned control policies based on high-dimensional image input was published in [8]. The breakthrough was achieved by combining deep neural networks with the techniques of RL, introducing Deep Q-Networks. However, the solution was constrained to small discrete actions spaces.

In general, for tasks of physical control one encounters high-dimensional continuous action spaces. Silver et al. [2] successfully proposed further advances adjusting Deep Q-Learning to the continuous action domain. By introducing the *deep deterministic policy gradient* algorithm (DDPG), we are now able to utilize deep RL to learn control policies in environments of continuous action spaces.

For the sake of generalizability, when working on optimal control problems in an RL setup we want to avoid too strict assumptions about the agents' morphology. Instead of having the policy assume a single specific geometric shape when deciding on an action, we want to learn a global policy that enables the agent to take well-informed steps. In this regard, Huang et al. [9] proposed to create a global policy based on a collection of *shared modular policies* (SMP). Consider an environment with a robot agent of some morphology. The basic idea is to deploy an SMP per actuator of the robot. By utilizing message passing between distant SMPs, a policy can be learned that masters complex coordination independent of the specific morphology of the agent.

In graph representations, it can be beneficial for individual nodes to learn different relations for different neighbors. [7] introduced the attention mechanism [6] for use in GNNs. Exploiting attention, individual nodes are able to efficiently learn different weights for different nodes present in the graph.

III. Background

Following a standard RL setting, we consider an agent acting in an environment modeled as *Markov decision process* (MDP). For each discrete time step t the agent observes a state s_t and performs action a_t. Carrying out $a_t \in \mathbb{R}^N$ yields a scalar reward $r_t \in \mathbb{R}$ and a new state s_{t+1}.

A. Deep Deterministic Policy Gradient

We concurrently learn a policy and a Q-function following the DDPG algorithm. DDPG is motivated by the fact that for any state s, given the optimal value-function $Q^*(s,a)$, the optimal action $a^*(s)$ can be computed by solving

$$a^*(s) = \underset{a}{argmax} Q^*(s,a). \tag{1}$$

DDPG specifically works on continuous action spaces for which solving Equation (1) becomes infeasible. This problem is overcome by approximation based on a policy $\pi(s)$, mapping states s to a probability distribution over actions a, yielding

$$\underset{a}{max} Q^*(s,a) \approx Q(s,\pi(s)).$$

Learning exploits a replay buffer of previous observations. Observations in the replay buffer are sets (s,a,r,s',d) of state s, action a performed in s, leading to new state s' and yielding result r. We store a Boolean value d to encode whether state s' is terminal. DDPG follows an actor-critic scheme. The critic is a neural network $Q_\phi(s,a)$ parameterized by ϕ that learns to approximate $Q^*(s,a)$. Updating the network parameters ϕ is based on minimizing the *mean-squared Bellman error* (MSBE)

$$Q_\phi(s,a) = \underset{\phi}{argmin} \underset{(s,a,r,s',d)\sim\mathbb{D}}{\mathbb{E}} \left[\left(Q_\phi(s,a) - y_i \right)^2 \right] \tag{2}$$

with observations (s,a,r,s',d) sampled from the replay buffer $\mathbb{D}$ and y_i as the target. To avoid Equation (2) directly depending on the parameters that are learned through y_i, target networks are introduced. We make use of a target policy $\pi_{\theta_T}(s)$ and a target Q-function Q_{ϕ_T}. Initially, parameters θ_T and ϕ_T are equal to the main networks parameters θ and ϕ. We update the target parameters by $\phi_T \leftarrow \rho\phi_T + (1-\rho)\phi$ and $\theta_T \leftarrow \rho\theta_T + (1-\rho)\theta$ with hyperparameter ρ. Target network updates follow main network updates. Now the targets can be defined as

$$y_i = r_i + \gamma Q_{\phi_T}(s'_i, \pi_{\theta_T}(s'))$$

with γ as discount factor. Finally, the actor is also a neural network $\pi_\theta(s)$. We learn parameters θ by maximizing the expected Q-function. Therefore, we solve

$$\pi_\theta(s) = \underset{\theta}{max} \underset{s\sim\mathbb{D}}{\mathbb{E}} \left[Q_\phi(s, \pi_\theta(s)) \right].$$

B. Hindsight Experience Replay

In order to learn how to act properly, an agent in RL requires some signal on performance. Typically, this signal is received in the form of a reward r after execution of action a in some observed state s. However, for some problems positive rewards are sparse. As a consequence, the agent rarely receives a signal indicating proper behavior which makes learning extremely difficult. *Hindsight experience replay* (HER) [3] tackles this problem. It additionally utilizes undesirable outcomes adding them as positive samples to the replay buffer. Let s_{target} be the state describing the target shape to reconstruct. Beginning in state s_0 the agent successively performs actions $a_{1:N}$ until it observes terminal state s_N. For each action a_i with $i = 1..N$ performed, the agent observes an experience $x_i = \{s_i, a_i, r_i, s_{i+1}, s_{target}\}$. Whenever the set of possible targets becomes very large, many observed terminal states do not match. These experiences alone do not help to find good policies. As a solution, whenever an undesired terminal state is observed, we create a new target $\widetilde{s}_{target} = s_N$. Each experience x_i with $i = 1..N$ along the corresponding trajectory is modified to $\widetilde{x}_i = \{s_i, a_i, r_i, s_{i+1}, \widetilde{s}_{target}\}$. Adding both, the modified and original samples to the replay buffer ensures the presence of a positive sample. Consequently, when sampling from the replay buffer during the learning phase the probability of encountering positive rewards is increased.

C. Multi-Head Attention Graph Representation

In this work, we closely follow the MHA graph representation proposed in [1], originally developed for the problem of autonomous assembly. The general idea is to use graph-based environment state representations as input to a GNN. Based on attention, the network is able to efficiently learn different weights for different nodes in the graph. This approach enables flexibility regarding the representation of the problem at hand.

Under the assumption of a Markov decision process, we formalize the graph $G = (N, E)$ with nodes N and edges E. Nodes N will encode the state of the environment. As we want to exploit the networks' ability to treat the neighbors of each node individually, N will be a set of the encoded states of differing nodes. The distinction of nodes is problem-specific, e.g., encoding some defined role in the current state of the environment.

Let C be the set of classes of nodes we distinct between. Then we have $N = \{N_c | c \in C\} = \{n_i | i = 1..N\}$.

Edges E simply define connections between nodes in the graph. To construct E we build an adjacency matrix with 1 at entry $E(i, j)$ if there is a connection between nodes n_i and n_j and 0 if not.

We utilize the graph state representation as input to the MHA architecture and project it into a higher dimensional space. Using a *fully-connected layer* (FC) as well as a ReLU activation function, we define function $g(n) = ReLU(FC(n))$ and apply it to all nodes n_i. We get

$$n_i^{(1)} = g(n_i^{(0)}) = ReLU(FC(n_i^{(0)})),$$

with $n_i^{(0)}$ corresponding to the initial nodes of the graph. The higher dimensional node embedding is further processed by multiple rounds of message passing. For each round l of message passing we pass the previous embeddings $\mathcal{N}^{(l-1)}$ of all nodes through a MHA layer with M heads. For each head we compute a key $k_i = W_k n_i^{(l-1)}$, a query $q_i = W_q n_i^{(l-1)}$ and a value $v_i = W_v n_i^{(l-1)}$. W_k, W_q and W_v are weights special to each head. We can then compute the output message of each head by

$$m_i = \sum_{j \neq i} a_{i,j} v_j,$$

with $a_{i,j}$ as attention weights of the connection from node n_i to n_j. To compute the attention weights, we first need the compatibility scores for the corresponding node connection. We compute them by

$$c_{i,j} = \begin{cases} \frac{1}{d} q_i^T k_j & E(i,j) = 1, \\ -\infty & \text{otherwise,} \end{cases}$$

with normalization constant d. The attention weights are now computed by $a_{i,j} = \frac{e^{c_{i,j}}}{\sum_{j' \neq i} e^{c_{i,j'}}}$. Finally, the MHA layers' output per node is a weighted sum of the messages of all heads

$$\text{MHA}(\mathcal{N}^{(l-1)}, i) = \sum_{o=1}^{M} W_{m,o} m_{i,o}, \tag{3}$$

with weights W_m special to each head. Using a skip connection $h(f(x)) = x + f(x)$, we compute node embeddings of round l by

$$n_i^{(l)} = h(g(h(\text{MHA}(\mathcal{N}^{(l-1)}, i)))).$$

We can view the overall result of message passing as an encoded version of the original graph. The encoded graph should now inherit a meaningful representation of the geometries, enabling the agent to take informed actions.

IV. Simulation Environment

We simulate the game of tangram in a PyBullet [4] environment. This approach allows us for fast and extensive data generation. Furthermore, the simulated environment allows the agent to access additional information useful for building expressive state representations. By exploiting exact observations of the current state, we remove the impact of errors naturally inherited by state estimations.

Simulation steps are performed relative to a fixed frequency of 240Hz. There are five different building blocks available in our current simulation. These building blocks are a parallelogram, a square and three isosceles triangles differing in size (see Figure 2). In this work, we only consider a single polygon present in the environment. Note that we treat tangram as a two-dimensional problem by ignoring the height of the three-dimensional building blocks in the simulation. This is a valid procedure as overlapping blocks are not allowed. For each simulation step, we have access to two different representations of the current state of the simulation. Generally, if we use the state as input to a neural network we encode it as matrix. The matrix representation is used to define the nodes of our graph representation. If we compute rewards, we convert a predefined space of the simulation environment into a grayscale image using an aerial view (see Figure 2). The resolution of the state image is fixed to 200x200 pixels to save memory. Our approach does not benefit from higher resolution images as they are solely used for reward computations. Movement of blocks is based on velocity control. Whenever a block is moved, we set its velocity in x- and y-direction for translation, as well as its angular velocity in z-direction for rotation (see Figure 3). Each block can only move once for exactly one simulation step unless the simulation is reset. We reset the simulation if a terminal state is encountered, i.e., once an action was performed on each polygon.

For training on GPUs, we want to exploit as much parallel computation capabilities as possible. However, as a consequence of the serial nature of the agents' observations, full parallelism is impossible. Therefore, we stick to a Master-Worker pattern. We initialize a single master running on GPU and multiple workers running on CPU. By using multiple workers, we are running simulations in parallel, even though the progress of individual simulations is serial. The task of each worker is to accumulate observations of the agent by acting in the environment. These observations are stored in the replay buffer. After passing a certain number of observations, the workers send them to the master. The job of the master is now to perform updates based on DDPG by sampling batches from the replay buffer.

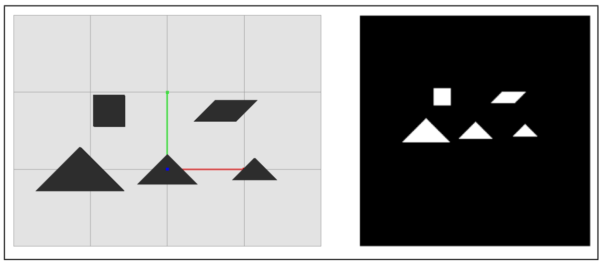
Fig. 2. All available polygons in PyBullet simulation (left) and the corresponding state representation as grayscale image (right).

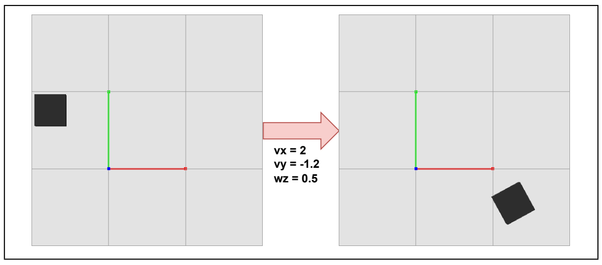

Fig. 3. Example move performed in PyBullet simulation. Floats vx and vy correspond to velocities in x- and y-direction. Float wz corresponds to angular velocity in z-direction. All values are given in cartesian coordinates. Movement is performed for exactly one simulation step.

A. *Target Sampling*

In practice, tangram target shapes usually resemble real-world objects. This imposes a difficult challenge on simulators used in an RL setup. Because of the high number of samples required for the algorithm to learn proper behavior, target shapes have to be automatically generated. Automatically generating shapes reconstructable by the polygons used for tangram that additionally follow the distribution of real-world target shapes is a non-trivial problem. However, in this work, we only consider targets of a single polygon. The presence of only one polygon drastically reduces the problem of target sampling. Therefore, our approach is to sample the target position and orientation from a two-dimensional Gaussian (see Figure 4). This strategy can be extended easily when generating target shapes of multiple polygons. We can introduce a curriculum learning approach by shrinking the covariance over time. At the beginning of learning, this strategy will produce targets of unconnected polygons. Ultimately, with a shrinking covariance, the polygons will begin to connect.

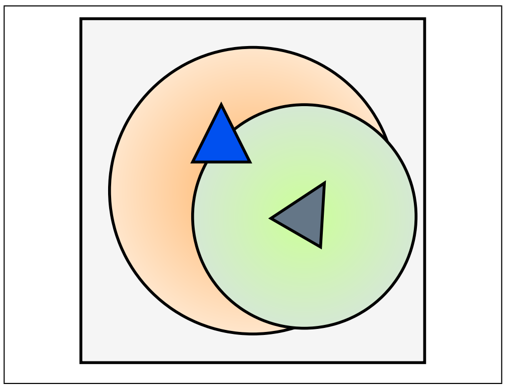
Fig. 4. Illustration of position sampling of target (gray) and non-target (blue) polygons. The target polygons' center point can take any position inside the orange circle. The center point of any non-target polygon is sampled from a two-dimensional Gaussian around the center of the target (green circle).

B. *Initial Polygon Position Sampling*

Our goal is to have a trained model that is able to properly position a polygon independent of its initial position on the plane. This goal imposes two prerequisites for spawning polygons in the simulation. First, the target position has to be reachable from the initial position. Second, we have to sample initial positions from a large set of candidates. At the beginning of learning, the agent will not know how to choose proper actions. As a consequence, actions leading to positive performance feedback will be very rare. However, instead of solving the problem of sparse rewards through suitable spawn position sampling, we perform learning based on DDPG with HER.

In this work, we sample initial positions of all non-target polygons from a two-dimensional Gaussian around the target (see Figure 4). Note that this sampling strategy can lead to initial positions already matching parts of the target shape.

V. Proposed Method

As stated earlier, the proposed method in this work closely follows [1]. The fundamental difference lies in an adjusted version of the multi-head attention graph representation to match our problem.

The task we aim to solve is to reconstruct a sampled target shape of non-overlapping polygons. For reconstruction, a learning agent has access to the same polygons that shaped the target. It should learn proper positioning of these polygons by moving them from some initial spawn point in the environment.

We want to create a graph-based representation of the state of the environment. This graph is formalized as $G = (N, E)$ with nodes N and edges E. At every time step, we represent the state as set of nodes $s = \{n | n \in N\}$. We consider both, the target and the polygons available to reconstruct it as part of the state. However, we want to introduce some

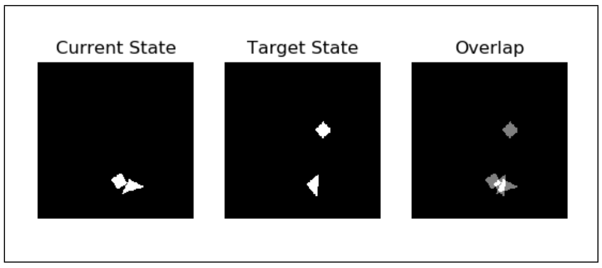

Fig. 5. Rewards are computed by the normalized sum of overlapping pixels between images of the target and the current state of the simulation.

distinctions. First, targets are fixed structures. We cannot perform any actions on them. Second, for each simulation, the agent is granted only a single action per non-target polygon. After a polygon has been moved, we consider it placed for the rest of the simulation. Consequently, we get a natural distinction between three classes of nodes. To capture all three classes, we define the following representation. Each node is represented by its x- and y-coordinate, as well as two Boolean values indicating whether the node belongs to a placed polygon and whether it is part of the target. Target nodes are always encoded as placed. In deciding which parts of the polygon are represented as nodes, we differentiate between target and non-target polygons. Equal to both, we include their corners as nodes in the state. But for non-target polygons, we additionally include their base, i.e., their center as a node. This node is regarded as control node and will later bridge the connection to other polygons. Furthermore, all actions computed will refer to these base nodes. In total, the size of the state equals the number of corners of all polygons plus a single node per non-target polygon.

The graph-based representation allows us to define connections between nodes, encoded by the edges E of the graph. In this work, we define an adjacency matrix with a 1 at entry $E(i,j)$ if there is a connection between nodes n_i and n_j and 0 otherwise. We model the connections as follows. For each polygon, we connect all corners with each other. For non-target polygons, all corners are additionally connected to the base node. To enable information passing between polygons, we connect all base nodes with all other nodes in the environment. See Figure 6 for an illustration.

Learning is based on DDPG with HER. The reward signal is computed as normalized sum of overlapping pixels between images of the target and the current state of the simulation (see Figure 5). With DDPG, we are making use of an actor-critic scheme. Consequently, we need two slightly differing network architectures. In general, both architectures resemble the MHA architecture. The task of the actor is to output actions $a(s) \in \mathbb{R}^3$, depending on the current state s. Actions describe polygon movements as velocities in x- and y-direction for translation, as well as an angular velocity in z-direction for rotation. We can feed the network the state in the representation described above without adjustments. Note that we can only apply one action to the base node of a polygon and only if the polygon has not been placed yet. To keep it simple, the actor computes actions for all nodes and we extract the corresponding action afterwards.

The critics' task is different in that it has to compute values $Q(s,a) \in \mathbb{R}$ depending on both, the current state s and some action a. We perform two adjustments compared to the actor. First, we condition on action a by appending it to each node in our state representation. Second, instead of computing a value per node we compute a global feature and output a single value only.

VI. Experimental Results

At the time of submission, no experimental results following the multi-head attention graph representation are available. Due to unfixed bugs, training is currently not possible.

Nonetheless, we have access to results from previous studies following a different approach. We use these results as a baseline to assess the performance of our approach. Both studies carry out learning based on DDPG with HER and rewards computed by the normalized sum of overlapping pixels of the target and resulting shapes. The fundamental difference lies in the representation of the state. This baseline uses grayscale images of the simulation environment as inputs to *convolutional neural networks* (CNNs). Using DDPG, the policy and Q-function are learned concurrently by taking an actor-critic approach. Both, the actor and the critic share equivalent CNN architectures as a backbone. We feed two images of the current state of the simulation and the target state to the network. The difference between the actor and the critic network begins after the last layer of the CNN backbone. While in both cases we append a multi-head module with one head per polygon present in the simulation, the Q-function has to be conditioned on the action taken. To satisfy this requirement we flatten the output of the last convolutional layer and extend the features with the action. The extended features are passed through additional fully-connected layers finally computing the Q-function. For the actor, we proceed analogously but without altering the output of the CNN backbone before passing it to the fully-connected layers.

A second difference is how initial positions of polygons are sampled. Instead of sampling the initial position by a two-dimensional Gaussian around the target, the baseline uses a hardcoded set of possible positions.

VII. Conclusions

In this work, we present a graph-based approach to the game of tangram. We follow the multi-head graph attention representation proposed by [1]. Adjusting the state representation to match our goals allows us to utilize the proposal for learning how to properly position polygons to reconstruct a target shape. The flexibility of GNNs enables us to gradually increase the difficulty of the problem by varying the number of polygons present in the simulation environment. Utilizing a PyBullet simulation environment further increases our control of the difficulty of the problem by allowing the agent to access exact observations instead of relying on state estimations.

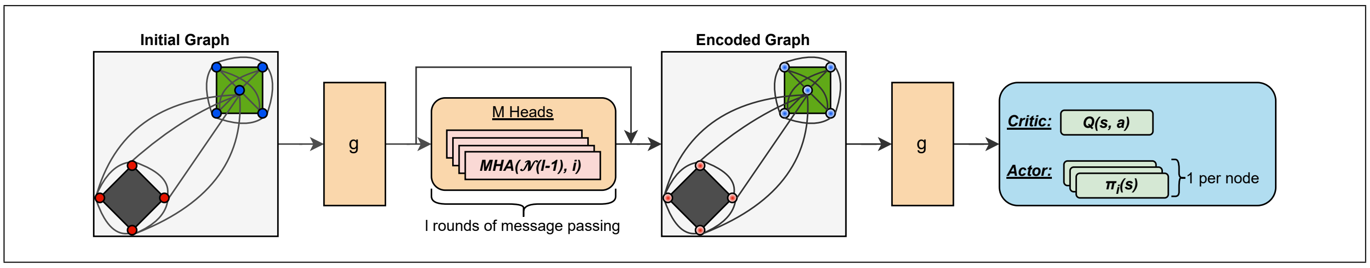

Fig. 6. Multi-head attention graph representation for tangram with one polygon. The initial graph is projected into a higher dimensional space. Multiple rounds of message passing yield an encoded version of the graph with a meaningful representation of the geometries. Based on this encoding, the agent can take informed actions.

VIII. Future Work

Future work should approach fully implementing and evaluating the proposed method. Ultimately, we want a learning algorithm to solve the game of tangram for the original setup with seven polygons.

Another important problem subject to future work is a suitable sampling strategy of target shapes. Currently, it is unclear how to automatically generate targets close to the real distribution of target shapes in tangram. This problem becomes more severe with increasing number of polygons in the environment.

Lastly, we exploit information only accessible through our PyBullet simulation environment. Using this additional information is a massive simplification of the problem. To represent the state of the simulation, we include the corner coordinates of each polygon, even for polygons building the target shape. In our simulation environment, we can access the corresponding coordinates directly. Ideally, we want to solve the problem with grayscale images as the only source of information. However, converting images into our proposed state representation requires reliable corner detection. We can expect a drastic increase in difficulty as the state representation loses some of the corner nodes in case of connected surfaces. Having access to all corner nodes of each polygon in the target shape gives strong geometric cues to the agent on where to position which polygon. Additionally, we now face the risk of false positives when detecting corners. As a consequence, errors are introduced to the state representation.

References

[1] Niklas Funk, Georgia Chalvatzaki, Boris Belousov, Jan Peters. Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction. *Proceedings of the 5th Conference on Robot Learning*, PMLR 164:1401-1411, 2022.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, Wojciech Zaremba. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, pages 5048-5058, 2017.

[4] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org. 2016–2021.

[5] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, Gabriele Monfardini. The Graph Neural Network Model. In *IEEE Transactions on Neural Networks Volume 20, Issue 1*, pages 61-80, 2009.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998-6008, 2017.

[7] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, Yoshua Bengio. Graph Attention Networks. *arXiv preprint arXiv:1710.10903*, 2017.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. In *arXiv preprint arXiv:1312.5602*, 2013.

[9] Wenlong Huang, Igor Mordatch, Deepak Pathak. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control. In *Proceedings of the 37th International Conference on Machine Learning*, PMLR 119:4455-4464, 2020.

[10] Yamada, Fernanda Batagelo, Harlen. A Comparative Study on Computational Methods to Solve Tangram Puzzles. In *30th Conference on Graphics, Patterns and Images*, 2017.

[11] E. S. Deutsch and K. C. Hayes Jr. A heuristic solution to the tangram puzzle. In *Machine Intelligence, vol. 7*, pages 205–240, 1972.

[12] K. Oflazer. Solving tangram puzzles: A connectionist approach. In *International journal of intelligent systems, vol. 8, no. 5*, pages 603–616, 1993.

[13] S. Z. Kovalsky, D. Glasner, and R. Basri. A global approach for solving edge-matching puzzles. In *SIAM Journal on Imaging Sciences, vol. 8, no. 2*, pages 916–938, 2015.

[14] D. Bartoněk. A genetic algorithm how to solve a puzzle and its using in cartography. In *Acta Scientiarum Polonorum. Geodesia et Descriptio Terrarum, vol. 4, no. 2*, pages 15–23, 2005.