Scalable and Generalizable Graph Representation for Reinforcement Learning

Mohammad Mahdi Rahimi
School of Electrical Engineering
KAIST
South Korea, Daejeon
mahi@kaist.ac.kr

Jaekyun Moon
School of Electrical Engineering
KAIST
South Korea, Daejeon
jmoon@kaist.edu

Abstract

Exploiting a system's symmetric properties has led to great successes in generalization in many fields of machine learning. While symmetric properties have traditionally been hand-coded in model architectures, more recent trends point to the systematic exploitation of symmetries in geometric deep learning, led by graph neural network (GNN) architectures. Although GNNs have been previously applied in reinforcement learning (RL), no uniform formulation and architecture address scalability in both observation and action space. In this work, we introduce a novel graph-based representation of joint action and observation (JACOB) to exploit various symmetries using the same representation and provide a constructive procedure for incorporating prior knowledge. Further, we address the limitations of the current GNN architectures in utilizing our representation and propose HERAT, a heterogeneous edge re-enhanced graph attention network, to fulfill JACOB's potential. We empirically show that, with JACOB, HERAT can generalize to out-of-distribution environments on the challenging board game of Risk and can easily be extended to other domains.

1 Introduction

Reinforcement learning (RL) has shown an empirical revolution in many challenging applications [22, 28, 39], primarily because of its ability to learn the statistical structure underlying observations and reward signals. However, systematic generalization to out-of-distribution tasks remains an open challenge [12, 50, 8]. A classical yet powerful mathematical remedy to the generalization problem is symmetry exploitation. Deep learning models are no exception [46, 1, 50]. Neural networks are designed to address the low-dimensional geometry arising from physical measurements, such as grids in images, sequences in time series, and their associated symmetries like translation or rotation.

It is shown that the cost of scalable training is significantly lower than directly solving a large-scale problem. This idea is discussed in terms of *importance of starting small* [11], and formalized as *curriculum learning* [3], which has been widely adopted in deep learning, under the assumption of having a fixed observation and action space [32, 19, 16].

Notably, board games are a suitable domain for studying scalability and generalization because rules can be trivially adjusted to make varying challenges while often involving complex strategies. Furthermore, games can introduce arbitrary symmetrical properties which typical neural network architectures cannot exploit [34, 44, 35, 36, 2].

Previous works focused on scalability in grid-based worlds and Go-inspired games. In contrast, we find that the Risk board game is an adequate challenge in which we can study unstructured scalability and out-of-distribution systematic generalization.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

We introduce JACOB, a transformation from joint action-observation to a graph representation, which exploits the task symmetries and reduces the decision tree. We further show that current GNN architectures cannot effectively utilize JACOB representation, so we propose HERAT, a heterogeneous edge re-enhanced graph attention network, to find a proper representation for graph entities.

We empirically demonstrate that JACOB-HERAT can generalize to out-of-distribution tasks and scale to a larger environment. Furthermore, we show that HERAT outperforms other GNN architectures and non-parametric baselines.

Contributions. To summarize, our key contributions are: *i*) We formulate a novel graph representation with which the observation and the action space can scale independently, achieving robust scalability and systematic generalization to a larger class of problems than previous works. *ii*) We complete the representation by proposing a GNN architecture that propagates and engages both edge and node features on a heterogeneous graph. *iii*) We empirically demonstrate that our representation and architecture can enhance an RL agent to obtain a general policy that outperforms baseline methods in out-of-distribution tasks.

2 Background

2.1 Systematic Generalization

Systematic generalization is described as a zero-shot policy transfer, where a policy is evaluated zero-shot on a collection of environments different from those it was trained on. A *controllable collection* is defined as an environment with factors of variation under user control [21]. The authors of [21] introduced four evaluation contexts for controllable collection as illustrated in Figure 1, where dimensions represent different factor sets. This text refers to interpolation and extrapolation distributions as in-distribution and out-of-distribution, respectively.

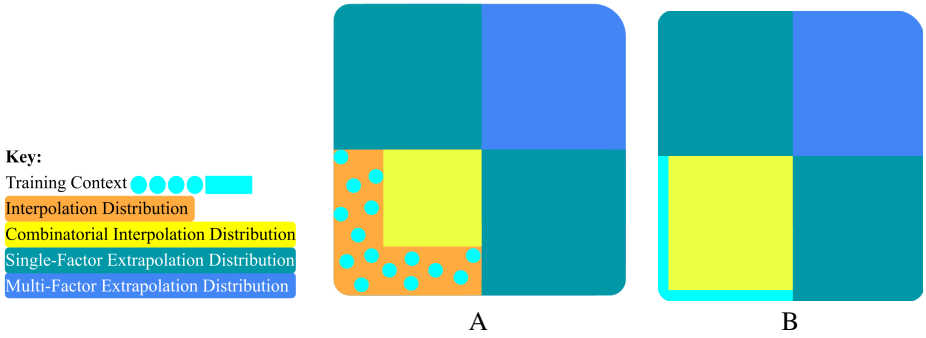


Figure 1: Evaluation context for controllable environments. Possible training (cyan), and testing (all other colors) context sets. The difference from A to B is that the training distribution samples only along a single axis; therefore policy will not be able to learn factors can vary independently at all.

2.2 Homogeneous and Heterogeneous Graph

A heterogeneous graph is defined as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, in which $\mathcal{V}$ and $\mathcal{E}$ represent the node and the edge set. Each node $v \in \mathcal{V}$ and edge $e \in \mathcal{E}$ are associated with a mapping function $\phi(v) : \mathcal{V} \to \mathcal{T}_\mathcal{V}$ and $\phi(e) : \mathcal{E} \to \mathcal{T}_\mathcal{E}$. $\mathcal{T}_\mathcal{V}$ and $\mathcal{T}_\mathcal{E}$ denote the type sets, and $|\mathcal{T}_\mathcal{V}| + |\mathcal{T}_\mathcal{E}| > 2$. A homogeneous graph can be seen as the case where $|\mathcal{T}_\mathcal{V}| = |\mathcal{T}_\mathcal{E}| = 1$ [40].

In an attributed graph, there exist mapping functions $\psi(v) : \mathcal{V} \to \mathcal{X}_\mathcal{V}$ and $\psi(e) : \mathcal{E} \to \mathcal{X}_\mathcal{E}$, where $\mathcal{X}_\mathcal{V}$ and $\mathcal{X}_\mathcal{E}$ denote the feature sets. A feature set $\mathcal{X}$ can be divided to two disjoint subsets of features: *categorical* ($\mathcal{C}_\mathcal{X} \in \mathbb{N}^+$) and *numeric* ($\mathcal{N}_\mathcal{X} \in \mathbb{R}$). A categorical feature represents a finite number of categories or distinct groups such as identifiers, while a numeric feature represents a measurable discrete or continuous value, such as weight features. This text refers to categorical and numeric features as identifiers and weights.

2.3 Graph Attention Networks

The graph attention network (GAT) [45] is one of the successful models which can engage edge features in node representation. Given a node i, the update rule for feature vector x_i is:

$$\mathbf{x}'_i = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j \tag{1}$$

and the attention coefficients for every neighboring node j connected by edge $e_{i,j}$ are computed as:

$$\alpha_{i,j} = \frac{\exp\left(\sigma\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \parallel \mathbf{\Theta}\mathbf{x}_j \parallel \mathbf{\Theta}_e\mathbf{e}_{i,j}]\right)\right)}{\sum_{k\in N_i^+}\exp\left(\sigma\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \parallel \mathbf{\Theta}\mathbf{x}_k \parallel \mathbf{\Theta}_e\mathbf{e}_{i,k}]\right)\right)} \tag{2}$$

where σ is a non-linear function (such as the ReLU) and N_i^+ denotes the neighboring nodes of i plus the node i. Θ indicates the weight matrix associated with the node and edge transformation.

3 Related Works

The current paper presents a solution for scaling RL algorithms with the help of knowledge representation and graph neural networks. It focuses on designing a general solution for board games regarding varying boards and the number of players. This section presents varying GNN architectures and how they can guide an RL agent and then briefly reviews recent advances in the Risk board game. We compare our work with existing methods at the end of each part.

3.1 Graph Neural Network

GNN is a class of machine learning models for representing graph-structured data. GNNs showed a satisfying performance in various domains [10, 30, 51, 33, 7]. They can make predictions on nodes, edges, or the graph, where every node shapes a belief by message passing within neighboring nodes.

Heterogeneous graph attention network (HAN) [47] proposed an extension to the GAT for the heterogeneous graph by specifying individual attention and convolution modules for each node type. Edge-enhanced GNN (EGNN) [15] introduced the propagation of the attention coefficients $a_{i,j}$ as the edge feature $e'_{i,j}$ to the next layer. Finally, the heterogeneous edge-enhanced graph attention network (HEAT) [29] completed HAN by involving edge attributes and types into the attention mechanism but did not propagate the transformed edge features.

Previous studies focused on engaging edge features to solve node and graph prediction tasks. In contrast, our RL policy relies on both node and edge representation. Therefore, we introduce a new architecture to propagate and transform edge attributes besides node features.

Graph convolution can not operate solely on the adjacency matrix. Therefore, we use centrality-based attributes to initialize the embeddings. It is empirically shown that using the node degree as the feature can achieve a satisfying performance on various datasets [9, 6]. Additionally, GNNs can be as strong as the Weisfeiler-Lehman (WL) graph isomorphism test [24], which means that a properly designed GNN model can learn to distinguish graph structures on par with the WL-test [49].

The current work utilizes the heterogeneous node degree, considering edge type and directions, as the initial node embedding. Moreover, edges are only initialized with a weight and a type.

3.2 Graph Neural Networks in RL

Recently, graph representations emerged in RL to establish benchmarks in enhancing generalization, transfer abilities, and data efficiency. They are particularly applied in continuous control [46, 23, 43, 4], multi-agent RL (MARL) [38, 25, 42], grid-based environments [20, 2], and robot co-adaptation [26, 17]. This section compares their methodology and objectives to our work regarding representation, architecture, and transferability.

To our knowledge, NerveNet [46] is the first work that used a GNN to represent an RL policy, achieving state-of-the-art performance. However, the authors of [23] conducted an ablation study that shows that in the presence of node attributes, morphological information encoded in the graph structure does not improve performance. They proposed a transformer-based approach, ignoring morphological information and outperforming GNN-based methods. In contrast, our work takes the opposite path and encodes the state information in a non-attributed graph structure, relying on the morphological information to achieve generalization and scalability while exploiting the symmetries.

Homomorphic networks [43, 42] are introduced to improve RL data efficiency in a single and multi-agent Markov decision process (MDP). It integrates GNNs with more equivariance constraints to exploit group-structured symmetries of the joint state-action space of an MDP. In contrast, our work focuses on exploiting graph-structured symmetries by transforming the environment space into a graph, achieving scalability and generalization.

MARL has also utilized GNNs to model a communication structure among agents. The deep graph network [18] represents dynamic multi-agent interaction as a graph convolution to learn cooperative

behaviors. More recently, MAHAC [37] was introduced to learn an efficient shared language across agents with different action and observation spaces. While prior works using GNNs in MARL have successfully modeled heterogeneous multi-agent communication, the frameworks mentioned are not designed to address multi-agent decision-making under dynamic action and observation space. On the other hand, our framework learns a representation over joint action and observation space, simultaneously leading to policy generation for all agents.

Among recent studies, SAZ [2] and Grid-2-Graph [20] are the most similar to our work. SAZ transformed the Go board game into a graph representation, achieving robust scalability regarding board size. It took advantage of having the same observation and action space to induce a policy per board squares. Grid-2-Graph transformed a grid-based environment into a multigraph, resulting in systematic generalization to out-of-distribution tasks. It incorporated external knowledge by adding auxiliary edges and generating a fixed scale policy over the entire graph representation. In contrast, our work proposes a formulation where observation and action space can scale independently, achieving robust scalability and systematic generalization not limited to grid-based environments.

3.3 Risk Strategy

To our knowledge, [31] and [48] are the first published works on the game of Risk. They applied Markov chain and decision complexity analysis to compute the branching factor of Risk. It is noteworthy to mention that the average branching factor of Risk with four players and 200 armies is approximated as 10^6. In comparison, the estimates for Chess and Go are 31 and 250, respectively.

Furthermore, [41] and [14] are the first works to use the Monte Carlo tree search (MCTS), and [27] applied temporal difference learning with one step look ahead. Following these works, [5] applied AlphaZero [39], achieving state-of-the-art performance.

All previous works considered only a fixed map two-player game. In contrast, our work solves the game for any number of players and arbitrary maps.

The rationale behind considering Risk as the benchmark is to establish JACOB's representation power in a stochastic multi-agent multi-task environment where zero-shot generalization was impossible with other baselines. Supplementary Material A shows JACOB's applicability to classic tasks, and we believe our work lays bridges for generalization studies of RL for more dynamic environments.

4 Methodology

4.1 Joint Action-Observation Space to Graph

JACOB applies three levels of transformation to observation and action space, resulting in a scalable graph representation. It starts with converting the observation space to a homogeneous graph. After that, it replaces features with auxiliary nodes, creating a non-attributed heterogeneous graph. Finally, it includes the valid action space, resulting in a non-attributed heterogeneous directed multigraph. For example the three levels of transformation are shown in Figure 2 given an initial state of a Risk game.

4.1.1 Transformation to Attributed Homogeneous Graph

Similar to recent works [20, 2], JACOB starts by transforming the observation to a homogeneous graph by considering a node per board location and representing other entities as node features. We represent identifier features with one-hot vector embedding, while weighted features are a single value. In Risk, nodes can represent territories while players and continents are identifiers, and the number of units is a weight feature.

Although the homogeneous graph can represent the observation, we argue that it does not encourage the model generalization. The primary problem is that identifier features remove the possibility to exploit symmetries. Therefore, it cannot scale and generalize from the identified aspect.

For instance, in Risk, the unique IDs of territories, players, and continents are the identifier features. Therefore, including the player's ID in the feature set will remove the players' symmetrical properties, making it impossible to add new players or perform permutations-invariant inference.

4.1.2 Transformation to Non-attributed Heterogeneous Graph

The second transformation eliminates the mentioned limitation by replacing features with auxiliary nodes. It exchanges the feature f of node v with an auxiliary node u_f and edge $e_{v \to u_f}$, where

Algorithm 1 Attributed Homogeneous Graph $\rightarrow$ Non-Attributed Heterogeneous Graph

Input: Homogeneous graph $\mathcal{G}$
Output: Heterogeneous graph $\mathcal{HG}$

for v in $\mathcal{G}.\mathcal{V}$ **do**
  Let $\mathcal{F}_v$: features of node v
  $\mathcal{HG}$.add_node(name=v, type=$\phi(v)$)
  for f in $\mathcal{F}_v$ **do**
    $\mathcal{HG}$.add_node(node=u_f, type=f)
    $\mathcal{HG}$.add_edge(edge=[v, u_f], weight=$\psi(f)$)
  end for
end for
return $\mathcal{HG}$

Algorithm 2 Non-Attributed Heterogeneous Graph $\rightarrow$ Non-Attributed Heterogeneous DiMultigraph

Input: Hetero graph $\mathcal{HG}$, Action Space $\mathcal{A}$
Output: Hetero multi di-graph $\mathcal{MG}$

$\mathcal{MG} \leftarrow \mathcal{HG}$
for a in $\mathcal{A}$ **do**
  if $\zeta(a).m$ **then**
    Let $v_a = \zeta(a).v$ # representative node
    Let $e_a = \zeta(a).e$ # representative edge
    $\mathcal{MG}$.add_node(node=v_a, type=$\phi(v_a)$)
    $\mathcal{MG}$.add_edge(edge=e_a, type=$\phi(e_a)$, w=$\psi(e_a)$)
  end if
end for
return $\mathcal{MG}$

the edge weight equals the feature f value. A zero-weight edge denotes a disconnected relation. Algorithm 1 demonstrates the construction procedure of the heterogeneous non-attributed graph.

For instance, in Risk, we add distinct nodes per player, continent, and unit numbers. They are all connected to territories, and the edge weight indicates the corresponding player, continent, or the number of units located in the territory.

This scalable representation exploits the observation symmetries. However, we still need a representative entity in the graph for each valid action to generate a scalable policy.

For instance, in Risk, a policy for attack cannot be inferred from the observation graph. Therefore, we add directed edges describing actions and feed their representation to the policy network.

4.1.3 Transformation to Non-attributed Heterogeneous Directed Multigraph

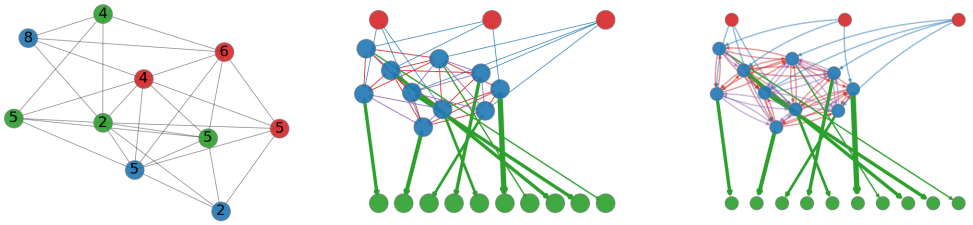
The last step introduces additional nodes and edges to represent valid actions. Given a heterogeneous non-attributed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ and an action set $\mathcal{A}$, a mapping function $\zeta(a) \rightarrow \{v, e, m\}$ exists as the domain knowledge for action $a \in \mathcal{A}$, where $v \in \mathcal{V}$, $e \in \mathcal{E}$, and $m \in \{0, 1\}$ denote representative node, edge, and validity of action a, respectively. Algorithm 2 demonstrates the construction procedure of the heterogeneous directed multigraph. Remarkably, the final graph representation includes all players' actions. Therefore, it is indifferent toward which player is utilizing the network.

Section A of Supplementary Material includes more examples of JACOB transformation for other environments. It describes how JACOB can improve symmetry exploitation and reduce the decision tree while constructing a scalable representation.

4.2 Heterogeneous Edge Re-enhanced Graph Attention Network

GNNs combine the representation and optimization process into one model. For instance, a directed edge can represent the state and affect the information flow simultaneously. HERAT tries to align representation and optimization by emphasizing edge embedding and transformation.

HERAT propagates and transforms both edge and node features. For a given graph $\mathcal{G}$ with $\mathcal{X}_\mathcal{V}$ and $\mathcal{X}_\mathcal{E}$ as node and edge features, it produces $\mathcal{X}'_\mathcal{V}$ and $\mathcal{X}'_\mathcal{E}$, where the node update rule is the same as



(a) Homogeneous Graph (b) Heterogeneous Graph (c) Heterogeneous Multi-Graph

Figure 2: Risk graph representation: In Homogeneous graph (a), nodes are territories where players and units are shown by color and label (attributes). In Heterogeneous graph (b) and multigraph (c), Players, Territories, and Units are shown with red, blue, and green nodes, respectively (types). In (b), territories are connected with neighboring edges to represent observation. In contrast, in (c), directed edges show the attack and fortify possibility among territories.

Table 1: GNN models comparison on edge engagement level.

MODEL	ATTRIBUTES	WEIGHT	TRANSFORM	PROPAGATE	TYPES
GAT	×	×	×	×	×
HAN	×	×	×	×	√
EGNN	√	√	×	√	×
HEAT	√	×	√	×	√
HERAT	√	√	√	√	√

Equation 1, while edge $\mathbf{e}_{ij}$ with type $\mathbf{e}_{ij}^{type}$ and weight $\mathbf{e}_{ij}^{w}$, will be initialized and updated as below.

$$\mathbf{e}_{ij}^{0} = \text{Linear}\left(\text{Embedding}(\mathbf{e}_{ij}^{type}) \parallel \mathbf{e}_{ij}^{w}\right) \tag{3}$$

$$\mathbf{e}_{ij}^{t+1} = \text{Linear}\left(\left[\mathbf{x}_i^t \parallel \mathbf{x}_j^t \parallel \mathbf{x}_i^t + \mathbf{x}_j^t \parallel \mathbf{e}_{ij}^t\right]\right) \tag{4}$$

The comparison on the level of edge representation engagement between GNN architectures is shown in Table 1. Section B of Supplementary Material shows the ablation study of the edge representation and the model architecture's influence on scalability.

4.3 Self-Play Reinforcement Learning

This section describes typical RL schemes in solving games. The empirical comparison is reported in the result section.

Model-Based Value Lookahead. The environment transition model is given to the agent. The model simulates the future state of actions, and the agent makes the optimal decision based on the values from the future. The look-ahead can be one or several steps into the future, empowered by search algorithms such as Minimax or MCTS.

Model-Free Actor-Critic. The agent is not aware of the transition and reward model. A critic network estimates future rewards to guide an actor network for generating an action distribution that maximizes the return value. Therefore, the actor-critic agent will generate a value and action distribution for every given state.

Model-Based Actor-Critic. The given model allows the actor-critic agent to use a look-ahead search for improving the performance. AlphaZero is one of the successful algorithms that uses MCTS in the look-ahead mechanism.

We follow the model-free scheme since we particularly look for scalability that comes from JACOB-HERAT representation, not additional search mechanisms. We further show that the model-free scheme has significantly lower inference costs, speeding up both evaluation and training procedures.

In every training iteration, we randomly initialize several environments and gather experiences through self-play. Afterward, we train the agent by batch sampling the experiences, and then we evaluate the trained agent against the previous version through a few rounds of competition. (e.g., for a two-player game, we evaluate ten times as the first-player and ten times as the second-player). If the agent passes the minimum threshold, we assign the agent to gather experience in the next iteration.

Table 2: Environment distribution configuration.

DISTRIBUTIONS	TERRITORIES	PLAYERS	UNIT COEFF.	EDGE CONNECTIVITY
TRAIN	$[2, 10]$	2	2	$[0.4, 0.6]$
EVAL($\mathcal{T}$)	**[2,50]**	2	2	$[0.4, 0.6]$
EVAL($\mathcal{TP}$)	**[2,50]**	**[2,5]**	2	$[0.4, 0.6]$
EVAL($*$)	**[2,50]**	**[2, 5]**	**[1,4]**	**[0.1, 0.9]**

4.4 Policy and Value Inference

The policy inference occurs differently in actor-critic and value look-ahead. In actor-critic, we designate a prediction network to transform the representation to logits and values for each node and edge type. Therefore, we can write the action probability and value estimation as:

$$\mathbf{P}(a|\mathcal{X}) = \text{softmax}\left(\text{MLP}\left(g\left(\mathcal{X}, \mathcal{G}; \theta_1\right)[i_a]; \theta_2\right)\right) \tag{5}$$

$$\hat{v}(\mathcal{X}, p) = \text{MLP}\left(g\left(\mathcal{X}, \mathcal{G}; \theta_1\right)[i_p]; \theta_3\right) \tag{6}$$

where $\mathcal{X}$ denotes node and edge features, g is a stack of GNN layers, and θs are neural network parameters. During inference, g returns a representation array of nodes and edges. i_a and i_p are the corresponding index of the action a and the player p in the array.

In model-based value lookahead, the value estimation for every possible future state occurs as Equation 6. Therefore action selection probability will be written as:

$$\mathbf{P}(a|\mathcal{X}) = \text{softmax}\left(\{\hat{v}\left(\mathcal{X}'\right)|\mathcal{X}' = \tau(\mathcal{X}, a); \forall a \in \mathcal{A}\}\right) \tag{7}$$

where τ is the transition function and $\mathcal{A}$ denotes the set of valid actions.

5 Experiment

The training and evaluation setups rely on randomly generating environments with controllable factors. In Risk, maps are randomly generated as Erdős-Rényi graphs, where territories are uniformly distributed between players, and player's units are binomially scattered over their territories. Therefore, each distribution of environments can be decided by controlling the number of players $\mathcal{P}$, territories $\mathcal{T}$, units $\mathcal{U}$, and edge existence probability $P_{\mathcal{E}}$.

Table 2 shows the environment generation distributions of training and evaluation. We refer to Table 2:Train as in-distribution and Table 2:Evals as out-of-distribution tasks.

5.1 Training Setting

Training environments are sampled from Table 2:Train configuration. The reward function returns $+1$ and -1, corresponding to the winner and loser at a terminal state.

We are evaluating zero-shot policy transfer. Therefore, no insight about out-of-distribution tasks is given to guide the training procedure, and the model training will be stopped after the 4000 training epochs, consisting of 200K Risk gameplay records. We collect accepted models during self-play and use the most recent model with the highest win rate for the evaluation.

5.2 Evaluation Setting

Test environments are sampled from Table 2:Eval distributions to evaluate out-of-distribution generalization performance. We generate ten maps per sampled parameter and evaluate the performance for all starting orders. We report the performance of model-free actor-critic empowered by HERAT beside model-based value lookahead, random policy, pure MCTS, and other GNN architectures.

In MCTS, we execute 1000 rollouts per action. It is sufficient to find an optimal policy for in-distribution tasks, but the performance drops as we move to the out-of-distribution.

In value look-ahead, we simulate the future state for every action and choose the action with the highest expected value. Value estimation for deterministic actions can be done with a single trial, while we average the value of ten sampled future states for stochastic actions.

5.3 Model Architecture

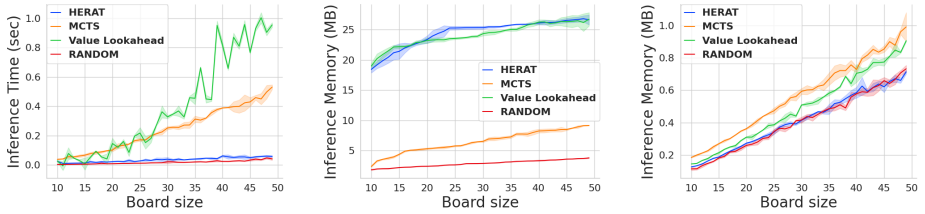
Section B of Supplementary Material shows the generalization study of varying architectures enhanced by HERAT. Our empirical results agree with [13], where a wide single-layer convolution achieves high scalability, while a deeper model fails. The HERAT baseline in the result section is a single-layer convolution with 128 neuron width and a ReLU function as non-linearity.

6 Results and Discussion

This section reports empirical results on generalization and computational complexity regarding varying methods and GNN architectures. Training curves are reported in Supplementary Material C.

Firstly, we note that increasing the board size $\mathcal{T}$ and the number of units $\mathcal{U}$ leads to the growth of game tree depth and branching factor while having more number of players $\mathcal{P}$ does not affect the decision tree (from the aspect of a single agent) and only increases the environment transition complexity. Further, edge probability $P_{\mathcal{E}}$ has a direct correlation with the branching factor and an inverse relation with game depth since the players have more freedom of decision-making.

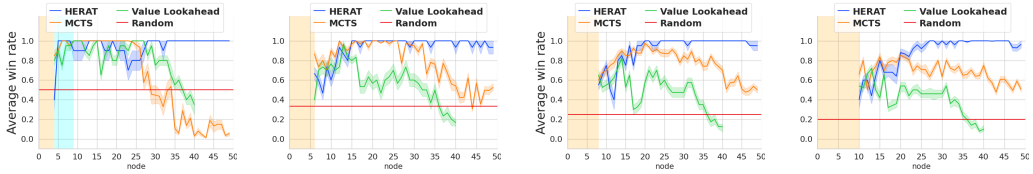
JACOB-HERAT has a constant time complexity regarding the game tree size as it can infer an action directly from the state representation. In contrast, the inference time of MCTS and value look-ahead depend on the depth and branching factor of the game tree. The HERAT's memory complexity is linear regarding graph size as the number of nodes is the batch dimension of GNN input, however the HERAT's allocated memory is larger than MCTS since it contains the neural network parameters. We conducted an experiment to measure memory usage by tracking *malloc* calls and differentiating allocated memory during inference. Figure 3 illustrates the average inference time and memory of 50 runs of HERAT, MCTS, value look ahead, and random strategy for a 2-player game where the number of territories increases from 10 to 50. We can see that HERAT inference time slightly changes, while MCTS and value look ahead increased exponentially as the environment scales up.



(a) CPU Time (b) Memory Space (c) *malloc* Calls

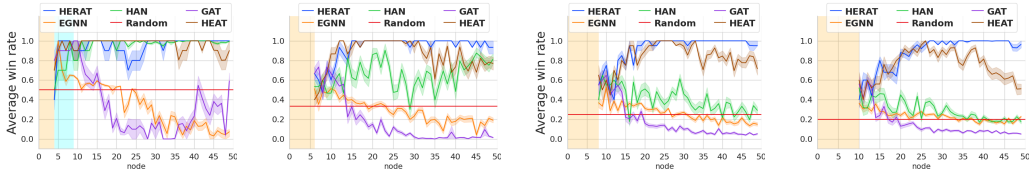
Figure 3: Inference cost comparison of different methods.

Figure 4 and 5 show the comparison of JACOB-HERAT with other methods and GNN architectures. Each figure consists of four plots showing evaluation on different numbers of players. The X-axis and Y-axis denote board size and win rate, while the cyan and orange shades indicate training and unevaluated distribution regions.



(a) 2-Player (b) 3-Player (c) 4-Player (d) 5-Player

Figure 4: OOD generalization performance of varying methods.

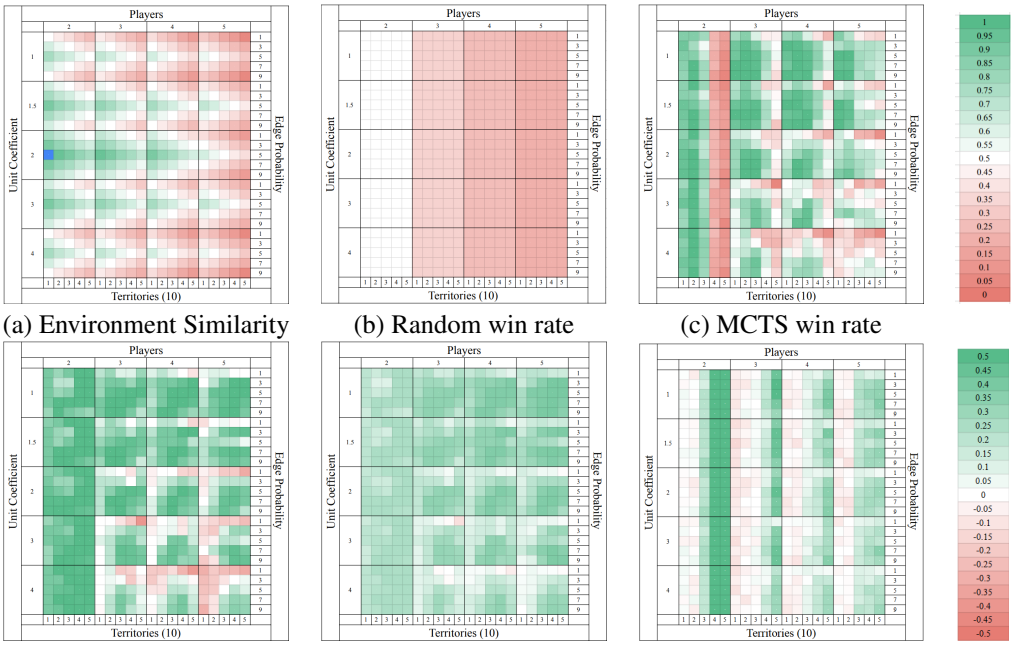


(a) 2-Player (b) 3-Player (c) 4-Player (d) 5-Player

Figure 5: OOD generalization performance of varying GNNs.

In contrast to baselines, JACOB-HERAT performance does not decline as the game tree or the number of players increases. However, all models perform poorly when fewer territories exist, pointing to the rise of randomness in Risk as the number of territories per player decreases. Therefore, in that case, having on par performance with MCTS is satisfactory since MCTS can achieve near-optimal performance given a shallow game tree. JACOB-HERAT goes beyond outperforming baselines and achieves near-optimal generalization performance.

Figure 4 shows that value look-ahead is more robust than MCTS toward the change in the number of players. Since the transition complexity of the environment does not affect the model-based methods. Figure 5 shows that GAT and EGNN fail to generalize and perform below the random baseline in out-of-distribution tasks. This points out the importance of using a heterogeneous GNN. The results also show that HAN architecture can generalize to more territories but fails as the number of players increases, indicating that the poor edge features representation cannot learn a general transition model. At the same time, HERAT utilizes a more sophisticated transformation and propagation, which outperforms HEAT architecture.



(a) Environment Similarity (b) Random win rate (c) MCTS win rate

(d) HERAT win rate (e) HERAT – Random (f) HERAT – MCTS

Figure 6: Generalization to out-of-distribution environments. The plot (a) indicates the similarity of each distribution to the training setting (shown with blue). The baseline win rate of random policy and MCTS is given in (b) and (c). The second row (d) illustrates the HERAT performance, while (e) and (f) illustrate its advantage to random and MCTS policy. MCTS underperforms in high-depth games, while it shows a near optimal behaviour dealing shallow game trees.

Figure 6 reports the evaluation on $Eval(*)$, where all controllable factors are altered from the training setup. It illustrates the generalization performance of each baseline in a heatmap, where each axis describes a controllable factor, and cell values denote the performance. The performance measures are the win rate and advantage regarding another baseline. Figure 6(a) shows the relative similarity between each setup and the training distribution. According to Figure 6, having less similarity to training distribution does not necessarily lead to lower performance, as changes in the distribution might create an easier problem than the original one. The extensive evaluation shows that JACOB-HERAT can generalize well to out-of-distribution tasks regarding all controlled parameters.

Additional Results. Hyperparameters, source code and additional experimental results regarding training curve analysis for GNN models and ablation study of edge representation and HERAT architecture are provided in the Supplementary Material.

7 Conclusion

This paper presents JACOB-HERAT, a framework for systematic generalization in RL by exploiting symmetric properties. JACOB transforms the environment's observation and action knowledge to a heterogeneous non-attributed directed multigraph input for GNN models. We discuss that the current GNN architectures underutilize JACOB representation. Therefore, we proposed HERAT to acknowledge the edge representation importance while considering the heterogeneous and directed nature of the graph. We empirically demonstrate that HERAT outperforms other GNN architectures in generalization to out-of-distribution tasks while improving the data efficiency of training at the same time. In the end, we conclude that JACOB-HERAT can generalize to the oversized board with more players by being trained on a small two-player game while having a manageable inference cost.

References

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Shai Ben-Assayag and Ran El-Yaniv. Train on small, play the large: Scaling up board games with alphazero and gnn. *arXiv preprint arXiv:2107.08387*, 2021.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[4] Charlie Blake, Vitaly Kurin, Maximilian Igl, and Shimon Whiteson. Snowflake: Scaling gnns to high-dimensional continuous control via parameter freezing. *arXiv preprint arXiv:2103.01009*, 2021.

[5] Erik Blomqvist. Playing the game of risk with an alphazero agent, 2020.

[6] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification, 2019.

[7] Nicholas Choma, Federico Monti, Lisa Gerhardt, Tomasz Palczewski, Zahra Ronaghi, Prabhat Prabhat, Wahid Bhimji, Michael M Bronstein, Spencer R Klein, and Joan Bruna. Graph neural networks for icecube signal classification. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 386–391. IEEE, 2018.

[8] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[9] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. On node features for graph neural networks, 2019.

[10] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015.

[11] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[12] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[13] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[14] Richard Gibson, Neesha Desai, and Richard Zhao. An automated technique for drafting territories in the board game risk. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 5, 2010.

[15] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9211–9219, 2019.

[16] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[17] Donald J Hejna III, Pieter Abbeel, and Lerrel Pinto. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

[18] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202*, 2018.

[19] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[20] Zhengyao Jiang, Pasquale Minervini, Minqi Jiang, and Tim Rocktaschel. Grid-to-graph: Flexible spatial relational inductive biases for reinforcement learning. *arXiv preprint arXiv:2102.04220*, 2021.

[21] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

[22] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[23] Vitaly Kurin, Maximilian Igl, Tim Rocktäschel, Wendelin Boehmer, and Shimon Whiteson. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*, 2020.

[24] AA Leman and B Weisfeiler. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya*, 2(9):12–16, 1968.

[25] Sheng Li, Jayesh K Gupta, Peter Morales, Ross Allen, and Mykel J Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.11438*, 2020.

[26] Kevin Sebastian Luck, Roberto Calandra, and Michael Mistry. What robot do i need? fast co-adaptation of morphology and control using graph neural networks. *arXiv preprint arXiv:2111.02371*, 2021.

[27] Manuela Lütolf. *A Learning AI for the game Risk using the TD (λ)-Algorithm*. PhD thesis, BS Thesis, University of Basel, 2013.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[29] Xiaoyu Mo, Yang Xing, and Chen Lv. Heterogeneous edge-enhanced graph attention network for multi-agent trajectory prediction. *arXiv preprint arXiv:2106.07161*, 2021.

[30] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[31] Jason A Osborne. Markov chains for the risk board game revisited. *Mathematics magazine*, 76(2):129–135, 2003.

[32] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015.

[33] Siyuan Qi, Wenguan Wang, Baoxiong Jia, Jianbing Shen, and Song-Chun Zhu. Learning human-object interactions by graph parsing neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 401–417, 2018.

[34] Norman Richards, David E Moriarty, and Risto Miikkulainen. Evolving neural networks to play go. *Applied intelligence*, 8(1):85–96, 1998.

[35] Thomas Philip Runarsson and Simon M Lucas. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation*, 9(6):628–640, 2005.

[36] Tom Schaul and Jürgen Schmidhuber. Scalable neural networks for board games. In *International Conference on Artificial Neural Networks*, pages 1005–1014. Springer, 2009.

[37] Esmaeil Seraj, Zheyuan Wang, Rohan Paleja, Matthew Sklar, Anirudh Patel, and Matthew Gombolay. Heterogeneous graph attention networks for learning diverse communication. *arXiv preprint arXiv:2108.09568*, 2021.

[38] Wenling Shang, Lasse Espeholt, Anton Raichuk, and Tim Salimans. Agent-centric representations for multi-agent reinforcement learning. *arXiv preprint arXiv:2104.09402*, 2021.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[40] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: a structural analysis approach. *Acm Sigkdd Explorations Newsletter*, 14(2):20–28, 2013.

[41] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. In *Advances in Computer Games*, pages 21–32. Springer, 2009.

[42] Elise van der Pol, Herke van Hoof, Frans A Oliehoek, and Max Welling. Multi-agent mdp homomorphic networks. *arXiv preprint arXiv:2110.04495*, 2021.

[43] Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[44] Erik CD van der Werf, H Jaap Van Den Herik, and Jos WHM Uiterwijk. Solving go on small boards. *ICGA Journal*, 26(2):92–107, 2003.

[45] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[46] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[47] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

[48] Michael Wolf. An intelligent artificial player for the game of risk. *Unpublished doctoral dissertation). TU Darmstadt, Knowledge Engineering Group, Darmstadt Germany. http://www. ke. tu-darmstadt. de/bibtex/topics/single/33*, 2005.

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[50] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[51] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Supplementary Material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Supplementary Material Section D
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Section 5 and 6 + Supplementary Material B
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Supplementary Material D
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Related Works
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

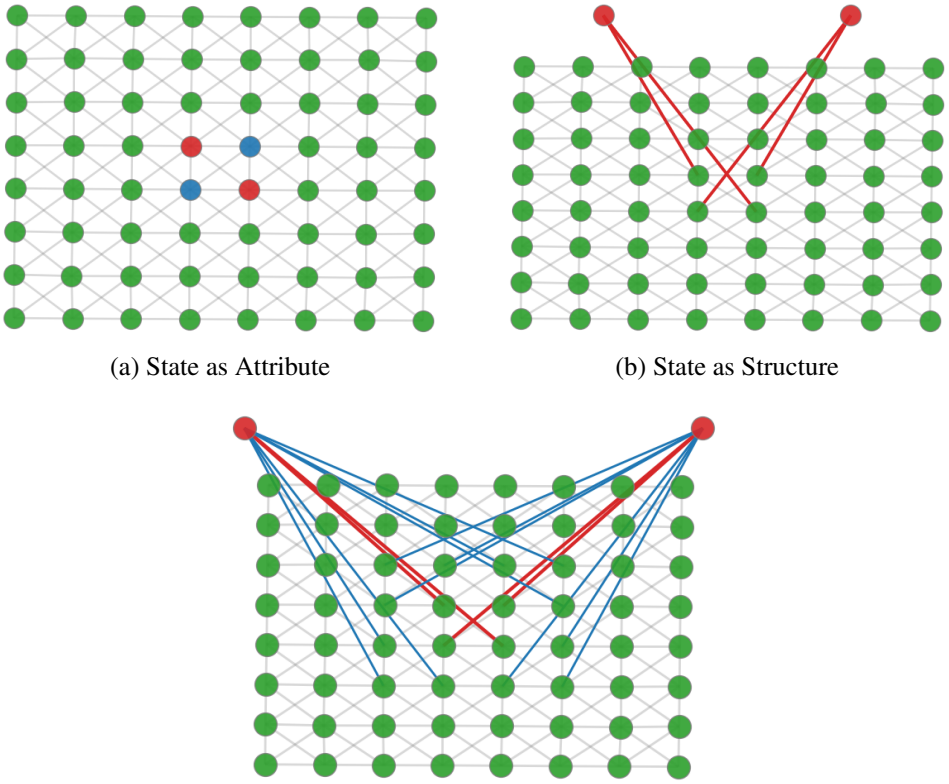
A JACOB Representation for other Games.

A.1 Go-inspired Games: Go/Othello/Gomoku

They are a sub-class of turn-based board games which consist of two players, a grid-based board, and a single piece type. At each turn, a player places one piece on the board.

Scalable AlphaZero [2] represents Go as a grid network graph where possible node attributes are 1, -1, and 0, respectively, for the player, opponent, and empty squares. The player symmetry is hand-coded by multiplying -1 into the array and having two separate representations for each state. The action space is considered to be the set of all board squares, where after the computation of policy per squares, invalid moves will be masked so that the agent decides over valid actions only.

In contrast, JACOB represents the board as a non-attributed grid network, where each player is represented by an auxiliary node connected to their pieces and possible valid actions on the board. JACOB exploits the board and player symmetries with a single representation, and therefore there is no need to compute and mask invalid actions since they are not represented in the first place. Figure A.2 illustrates an example of Othello represented at each level of JACOB transformation.



(a) State as Attribute

(b) State as Structure

(c) State-Action as Structure

Figure 7: Othello graph representation: In (a), the state information is represented as the attributes of nodes, where blue, red, and green corresponds to 1, -1, and 0. In (b), players are red nodes connected to their pieces with a red edge, and In (c), each player's valid actions are represented by blue edges.

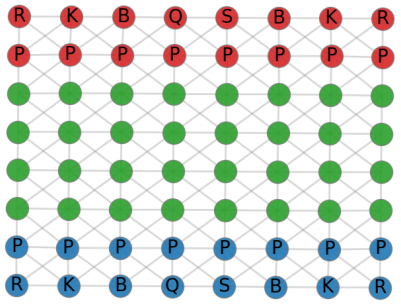
A.2 Chess Variations

Similar to Go-inspired games, chess is a turn-based, two-player, grid-based board game. However, several piece types are included. In each turn, a player moves one piece on the board, which might lead to the removal of another piece.

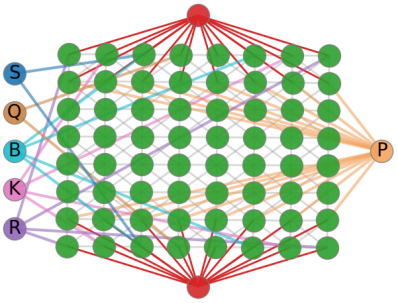
AlphaZero [39] represents each piece type with a one-hot vector and extra features for player and piece repetition identification. Similar to the previous case, the action space considers all possible movements, and after policy computation, the invalid moves will be masked.

With JACOB, we do not need to identify players, piece types, or repetition. Similar to the previous case, JACOB represents the board as a grid where each square connects to its neighbors. Each board game square is connected to its corresponding player and piece. Furthermore, valid and invalid moves are represented as distinct directed edges from the piece to ongoing locations. The intuition is that piece identification is not necessary if the possible actions of each piece are given.

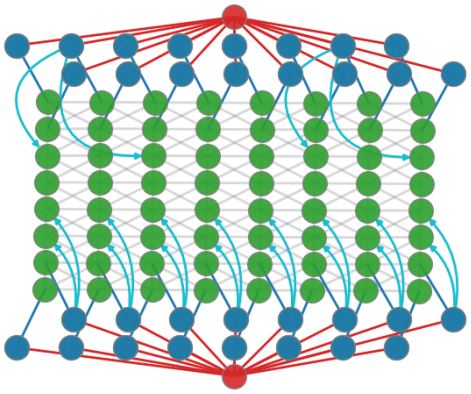
Notably, we do not discriminate between piece types, and they are represented with the same node type. Their possible actions (both valid and invalid) will tell them apart. This representation choice exploits many symmetric properties of one game and allows us to add new piece types or evaluate policy over Chess variations with the same objectives. We can also add more domain knowledge about game situations, such as adding a relation between two pieces blocking or threatening each other. Figure 8 illustrates the three level of JACOB representations for Chess.



(a) State as Attribute



(b) State as Structure



(c) State-Action as Structure

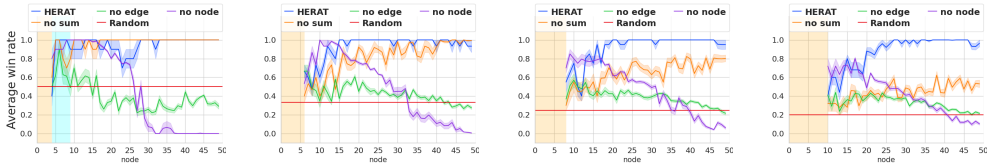
Figure 8: Chess graph representation: In (a), the state is represented as node attributes, where blue and red denote player identifier and green is neutral. Moreover, (b) and (c) are non-attributed graphs where the color denotes node type, red and green indicate players and board squares. In (b), every piece type is represented with an auxiliary node type, while (c) relies on the action set of pieces as their identifiers. For the sake of clarity, (c) represents the Pawns and Knights open actions only for one side, but in practice, representation includes both sides and blocked actions.

B Ablation Study of HERAT Architecture

B.1 Edge Representation

We suggested the edge representation in HERAT architecture as Equation 4. This section compares model performance in the absence of summation ($\mathbf{x}_i^t + \mathbf{x}_j^t$), node features ($\mathbf{x}_i^t \parallel \mathbf{x}_j^t$) and edge attribute ($\mathbf{e}_{ij}^t$).

As shown in Figure 9, removing the node features decreases the performance significantly as it removes crucial information from the environment state, which makes the edge direction invariance, while eliminating the summation term leads to minor performance loss. It also shows that edge attributes are an essential part of the representation.



(a) 2-Player (b) 3-Player (c) 4-Player (d) 5-Player

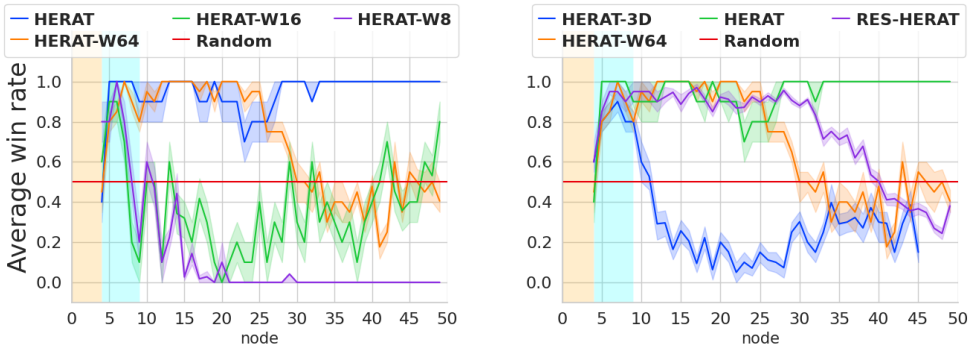
Figure 9: The performance for Random and HERAT are the same as reported in the paper. "no sum", "no node", and "no edge" denote edge representation without the summation term, node terms, and edge attributes, respectively.

B.2 Model Architecture

We study architectures enhanced by HERAT under different widths, depths, and the presence of residual connections. The detailed model configurations are shown in Table 3, and the performance regarding each configuration is reported in Figure 10. It shows the importance of having wide layers in achieving scalability. Further, it shows the failure of a deeper model for generalization in the absence of skip connections.

Table 3: HERAT architecture configurations.

MODEL	DEPTH	WIDTH	PARAMETERS
HERAT-W8	1	8	603
HERAT-W16	1	16	1819
HERAT-W64	1	64	22555
HERAT	**1**	**128**	**86043**
HERAT-3D	3	64	97859
RES-HERAT	3	64	116687



(a) Varying Width (b) Varying Depth & Residual

Figure 10: Scalability study on model architecture.

C Training Curves for In- and Out-of-distribution Performance

This section reports the performance of different GNN architectures during the training procedure. In Figure 11, each plot shows smoothed training curves for ten runs of evaluation per agent acceptance. Plot (a) represents in-distribution generalization, and plots (b) to (e) show the out-of-distribution performance of GNN models.

Finally, plot (f) shows HERAT out-of-distribution performance exclusively. It demonstrates that JACOB-HERAT achieves out-of-distribution generalization as the training continues beyond the convergence point in the in-distribution region. This phenomenon also occurs for HEAT architecture but with minor significance.

(a) In-Distribution (b) OOD: 2-Player (c) OOD: 3-Player

(d) OOD: 4-Player (e) OOD: 5-Player (f) HERAT Training

Figure 11: Generalization to in- and out-of-distribution environments during the training procedure: All models are trained for the same amount of epochs. However, the performance was evaluated per agent acceptance in the self-play update. Therefore the data points vary for each model.

D Hyperparameters and Computational Resources

For reinforcement learning and gradient descent optimization, we used the same parameters for all model architectures and they reminded fixed during the training, as shown in table 4.

Table 4: RL and Optimization Hyperparameters.

Module	Hyperparameter	Description	Value
PPO	lr	Learning Rate	$7e-4$
PPO	γ	Reward Discount Factor	0.99
PPO	$C_{entropy}$	Entropy Term Coefficient	0.01
PPO	C_{value}	Value Loss Coefficient	0.5
PPO	epochs	PPO Epochs	4
PPO	mini batch	Number of Batch for PPO	32
PPO	memory_size	Size of Replay Memory	$10e3$
PPO	target_update	Interval for Updating Target Network	2000
PPO	clip	PPO Clip Parameter	0.2
RMSProp	ϵ	Optimizer epsilon	$1e-5$
RMSProp	α	Optimizer alpha	0.99
Simulation	num_env_step	Number of Environment Steps to Train	$10e6$
Simulation	num_episode	Number of Episodes from The Environment	3000
Simulation	max_step	Maximum Number of Step During an Episode	200

Additionally, Table 5 describes parameters of varying GNN architectures that have been evaluated. We adjust the convolution width to have (≈100K) parameters.

Table 5: GNN architecture configurations.

Model	Depth	Width	Parameters
GAT	1	384	152476
HAN	1	384	154368
EGNN	1	256	138783
HEAT	1	256	139803
HERAT	1	128	86043

Lastly, we conduct our experiments on a machine with 32 cores of CPU and one NVIDIA TITAN Xp GPU with 12GB of memory. During training, parallel game simulations ran on each CPU, while policy updates occurred concurrently on a single GPU.