SymNet 3.0: Exploiting Long-Range Influences in Learning Generalized Neural Policies for Relational MDPs

Vishal Sharma[*1] **Daman Arora**[*1] **Mausam**[1] **Parag Singla**[1]

[1]Indian Institute of Technology Delhi {vishal.sharma, cs5180404, mausam, parags}@cse.iitd.ac.in

Abstract

We focus on the learning of generalized neural policies for Relational Markov Decision Processes (RMDPs) expressed in RDDL. Recent work first converts the instances of a relational domain into an *instance graph*, and then trains a Graph Attention Network (GAT) of fixed depth with parameters shared across instances to learn a state representation, which can be decoded to get the policy [Sharma et al., 2022]. Unfortunately, this approach struggles to learn policies that exploit long-range dependencies – a fact we formally prove in this paper. As a remedy, we first construct a novel *influence graph* characterized by edges capturing one-step influence (dependence) between nodes based on the transition model. We then define *influence distance* between two nodes as the shortest path between them in this graph – a feature we exploit to represent long-range dependencies. We show that our architecture, referred to as *Symbolic Influence Network* (SymNet3.0), with its distance-based features, does not suffer from the representational issues faced by earlier approaches. Extensive experimentation demonstrates that we are competitive with existing baselines on 12 standard IPPC domains, and perform significantly better on six additional domains (including IPPC variants), designed to test a model's capability in capturing long-range dependencies. Further analysis shows that SymNet3.0 automatically learns to focus on nodes that have key information for representing policies that capture long-range dependencies.

1 INTRODUCTION

Recent work has shown the successful application of neural models for the task of automated planning [Hafner et al., 2018, Groshev et al., 2018]. Of particular interest are relational domains – which are characterized by objects, predicates, and a first-order transition model. These are typically represented in the form of a Relational Markov Decision Process (RMDP) [Boutilier et al., 2001], where grounding an RMDP with a set of objects results in a specific problem instance. A state is represented by an assignment to the groundings of predicates, referred to as state variables. Multiple different languages have been proposed to represent RMDPs, the popular ones being Relational Dynamic influence Diagram Language [Sanner, 2010] (RDDL) and Probabilistic Planning Domain Definition Language [Younes et al., 2005] (PPDDL), with our focus in this work being the former [1]. Given an RMDP expressed in RDDL, the goal then is to learn a single generalized policy that is applicable to any instance of the domain. Recent works [Garg et al., 2019, 2020, Sharma et al., 2022] have made progress in this direction, showing that it is possible to train a neural model on smaller instances, which generalizes to (unseen) instances from the same domain.

Existing approaches convert a given instance into an *instance graph* where nodes represent object tuples [2], and edges represent influence based on the transition model [3]. A Graph Attention Network (GAT) is used to compute node embeddings in this graph, and the state embedding is typically computed as a combination of node embeddings along with an aggregation function such as maxpool applied over them. A decoder network takes the state representation and gives a distribution over actions in the current state, resulting in a policy. Though this approach has met with initial success, some important problems remain. The key one that

[*]Equal Contribution

[1]other approaches using PPDDL are discussed in related work

[2]a tuple of objects appearing as argument of some predicate

[3]additional nodes and edges are created representing singleton objects and their connections to object tuples that they are part of

Proceedings of the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023), PMLR 216:1921–1931.

we tackle in this paper is that of capturing long-range dependencies. For instance, consider the problem of navigating in a large grid where the objective is to reach a cell designated as the goal starting from the current location of the agent. Assume that each non-goal (non-agent) cell has an identical set of features, and the node embeddings are learned using a GAT with d layers. Then, consider the goal being in the middle of the grid, and two different states s_1 and s_2, such that robot is to the left of the goal in s_1 and to the right in s_2 at a distance $2d + 1$. Then, ignoring any edge effects, the score for any given action (say 'left') will be identical in the two states. This is because any node in the network has view either of the robot or the goal, but not both. Further, one can establish a one-to-one mapping between the node embeddings in the resulting states, hence taking any aggregate function which is permutation invariant will result in identical state embeddings. Since the optimal actions in s_1 and s_2 are different, i.e., left and right respectively, there is no way for the model to learn the optimal policy in this case. We formally prove this deficiency for existing architectures. Increasing GAT depth d is not a solution either, due to blow-up in the number of parameters and other learnability issues with long-distance message passing [Li et al., 2018, Wu et al., 2020, Alon and Yahav, 2021].

As a remedy, we propose constructing a novel graph, referred to as the *influence graph*, whose nodes represent state variables, and two nodes are connected by an edge if they can influence each other in one step, based on the transition model. Intuitively, the distance between two nodes in the influence graph represents the minimum number of steps it would take for the influence of one node to reach the other via the transition dynamics of the model. The pairwise distances thus computed in the influence graph can be useful features for capturing long-range dependencies. We show that the addition of these distance-based features gives the model the representational power to capture long-range dependencies for a large class of problems. Our architecture referred to as *Symbolic Influence Network* (SYMNET3.0)[4], builds on Sharma et al. 2022, and adds distance-based features at every node in the instance graph to capture the long-range dependencies. A multi-head attention is used for learning to focus on relevant nodes based on these features resulting in a distance-aware state representation, enabling our model to capture long-range dependencies. The resulting state is then decoded to get the policy as before.

Similar to earlier works [Garg et al., 2020, Sharma et al., 2022], we operate in the offline planning setting and train SYMNET3.0 by imitation learning using the data generated from an online planner PROST [Keller and Eyerich, 2012]. Our extensive experimental evaluation shows that (a) we are competitive with existing baselines on 12 IPPC domains which do not necessarily require capturing long-range dependencies for learning the optimal policy, and (b) we are significantly better on 6 new domains (4 of them being IPPC variants) specifically designed to test the efficacy of the models when long-range dependencies need to be exploited for learning a good policy. Specifically, in the latter case, SYMNET3.0 performs better than its closest competitor SYMNET2.0 in all domains with a gain of 18% relative performance in the aggregate metric. Further analysis reveals that the influence-layer of SYMNET3.0 learns to focus attention on key nodes in the network central to capture the long-term dependencies.

[4]The code for SYMNET3.0 and the RDDL instance generators can be found at https://github.com/dair-iitd/symnet3

2 BACKGROUND

2.1 RELATIONAL MARKOV DECISION PROCESSES USING RDDL

Relational Dynamic influence Diagram Language [Sanner, 2010] (RDDL) defines a first-order Relational Markov Decision Process (RMDP) in two parts, 1) a domain description that represents the object types (C), state-fluent predicates (SF), non-fluent predicates (NF), action predicates (A), first-order transition functions (T) and first-order reward functions (R); and 2) an instance description that represents a specific instance of the domain by describing its ground objects ($\mathcal{O}$), initial state (s_0), discount factor (γ) and horizon (H). State-fluents are predicates that can change over time, whereas, non-fluents are predicates whose values are fixed for a given instance but can vary from instance to instance. Together they form the set of state predicates (SP). Grounding a predicate implies replacing each argument of the predicate with an object-tuple having type-consistent objects. Grounding state predicates forms a set of state-variables ($SP_{\mathcal{O}}$), and grounding action predicates forms the set of ground actions ($A_{\mathcal{O}}$). A state is defined as an assignment to all state-variables, denoted by $s \in \mathcal{PS}(SP_{\mathcal{O}})$, where $\mathcal{PS}$ is the power set. We denote the set of object tuples appearing in state-fluents as O_{SF}. The set of object tuples for which either a numeric non-fluent is defined or a true boolean non-fluent is defined is denoted as O_{NF}. Let Ar denote the maximum arity of any Predicate in SP. Throughout the paper, we denote any (state-fluent or non-fluent) ground predicate $P(u_1, ...u_k)$ as $P(\langle u \rangle)$ where $\langle u \rangle = \langle u_1, ...u_k \rangle$ is an object tuple.

Each instance has an underlying Dynamic Bayesian Network (DBN) capturing its transition dynamics, which is a bipartite graph with (i) a set of nodes for each state-variable and each ground action at time t and (ii) a set of nodes for each state-variable at time $t + 1$ and a reward node. There exists an edge from a node in the first node-set (i.e., at time t) to a node in the second node-set (i.e., at time $t + 1$) if the value of the former affects the value of the later [Mausam and Kolobov, 2012].

A sequence of works [Bajpai et al., 2018, Garg et al., 2019,

2020, Sharma et al., 2022] learns generalized neural policies for RDDL RMDPs. All these works have their limitations: Torpido [Bajpai et al., 2018] can not transfer across instance size, TrapsNet [Garg et al., 2019] only handles domains with limited state and action predicate arities, and SymNet [Garg et al., 2020] ignores most non-fluents leading to generalization limitations. We build on the most recent of these, SymNet2.0 [Sharma et al., 2022], that improves on SymNet (described in the next section).

2.2 SYMNET2.0

Instance-Graph(s): SymNet2.0 converts a given instance into a set of graphs each referred to as an instance-graph, each having two types of nodes, (i) singleton object nodes: for each object $o \in \mathcal{O}$, a node o is added to all graphs. (ii) object-tuple nodes: for each unique object-tuple, i.e., for each $\langle u \rangle \in O_{SF} \cup O_{NF}$, a node $\langle u \rangle$ is added to all graphs. We will use n to denote a node corresponding to either an object or an object-tuple. There are three types of graphs that capture different types of interactions among state-variables via edges that are created as follows,

1. DBN-based graph (G_d): An edge is added between nodes $\langle u \rangle$ and $\langle v \rangle$ if there is a state-fluent $P(\langle u \rangle)$ that affects another state-fluent $Q(\langle v \rangle)$.
2. Action-based graphs ($\{G_{a1}, ..., G_{a|A|}\}$): An edge is added to graph G_{ai} between nodes $\langle u \rangle$ and $\langle v \rangle$ if there is a state-fluent $P(\langle u \rangle)$ that affects another state-fluent $Q(\langle v \rangle)$ via action ai.
3. Position-based graphs ($\{G_{p1}, ..., G_{p|Ar|}\}$): A bidirectional edge is added between a singleton object node o and an object tuple node $\langle u \rangle$, in the graph G_{pi}, if o comes at position i of object tuple $\langle u \rangle$.

Node features: SymNet2.0 adds node features in each graph as (i) For each predicate $P(\langle u \rangle) \in SF \cup NF$, a feature is added to node $\langle u \rangle$. (ii) For each unparameterized predicate $Q \in SF \cup NF$, a feature is added to all nodes. (iii) For each node, a one-hot encoding representing the type of the node is added. For object-tuple $\langle u \rangle = \langle u_1, ..., u_k \rangle$, $type(\langle u \rangle) = (type(u_1), ..., type(u_k))$. The values for features corresponding to SF and NF come from the current state and the RDDL descriptions, respectively.

Node Embeddings: Next, SymNet2.0 computes node embeddings for each of these graphs by using a Graph Attention Network (GAT) [Veličković et al., 2018]. Each graph is passed through an independent GAT with fixed neighborhood size. The node embeddings from each graph are then merged into a single node embedding as $\forall v \in V, ne(v) = concat(ne_{G_d}[v], ..., ne_{G_{p|Ar|}}[v])$, where V is the set of all nodes. To capture the complete state, a global embedding is also computed as $ge = maxpool_{v \in V}(ne[v])$. The set of node embeddings, along with the global embeddings, represent the state-representation.

Next, a set of action decoders is created for each action type, denoted by $\{AD_1, ..., AD_{|A|}\}$. For a global action ac and for a ground action $ac(\langle o \rangle)$, where $o = (o_1, ..., o_k)$, that affects a set of state-variables $\mathcal{P}_{a(\langle o \rangle)}$, the score is given as,

$$score(a(o)) = AD_{type(a)}\big(ne[o_1], ..., ne[o_k],\ ge,\ maxpool_{P \in \mathcal{P}_{a(\langle o \rangle)}}(ne[args(P)])\big) \quad (1)$$

$$score(ac) = AD_{type(ac)}(ge) \quad (2)$$

Here, $args(P)$ returns the arguments of predicate P. The scores of all actions are then normalized to get a policy. For training, imitation learning is used where the data is generated using the state-of-the-art online planner PROST [Keller and Eyerich, 2012].

3 TECHNICAL CONTRIBUTIONS

The organization of this section is as follows. We first highlight the deficiencies of SymNet2.0 in representing long-range dependencies. We then present the architecture for SymNet3.0 which addresses these limitations by incorporating the notion of distance-based features.

3.1 LIMITATIONS OF SYMNET2.0

We note that SymNet2.0 uses fixed depth GAT, each node can only access information in its immediate neighborhood and ignores the information beyond its field of view. The only way to have information access beyond the field of view is through global embedding. However, SymNet2.0's max pool-based global embedding is a commutative operation that ignores the structure of the graph. Hence, if we swap node embeddings of any two nodes, SymNet2.0 will not be able to identify this change leading to sub-optimal policies. Next, we will explain this formally.

Theorem 1. *Let G be an instance graph with two nodes n_1 and n_2 representing object-tuples of state-variables, having identical $2d$ hop neighborhoods. Let s_1 be some state where n_1 and n_2 are distinct nodes having node features f_1 and f_2, respectively, such that $f_1 \neq f_2$. Let s_2 be another state where the node features of n_1 and n_2 are swapped with each other, and the remaining node features are same as in s_1. Let the state-representation (set of node embeddings and maxpool global embedding) be computed by using a GAT of fixed-depth d. Then, the state-representations of s_1 and s_2 will be the same and we refer to s_1 and s_2 as symmetric states with respect to nodes n_1 and n_2.*

Proof (Sketch). Only the nodes in the d hop neighborhood of n_1 and n_2 notice the swap, and as their neighborhoods are identical, there is a one-to-one correspondence between node embeddings before and after the swap. Node embeddings of all nodes outside the d hop neighborhood of n_1

and n_2 will be unchanged. Next, a commutative function like maxpool will return the same value before and after the swap. Hence the set of node-embeddings and global embedding that formulate the state-representations remains unchanged. □

Theorem 2. *In reference to Theorem 1, let there be two singleton nodes o_i and o_i' whose features have been swapped in symmetric states s_1, s_2. Let $ac_1 = A_C(o_1, ...o_i, ..., o_k)$ be an action applicable in s_1 and $ac_2 = A_C(o_1, ..., o_i', ..., o_k)$ be an action applicable in s_2, that differ only in the arguments o_i and o_i'. Further, if $\mathcal{P}_{ac_1} \setminus \{o_i\} = \mathcal{P}_{ac_2} \setminus \{o_i'\}$, then, the action score assigned by* SymNet2.0 *to ac_1 and ac_2 will be the same.*

Proof (Sketch). As node-embeddings of o_i and o_i' are the same, the action scores for ac_1 and ac_2 will be the same. □

Corollary 1. *The action score assigned by* SymNet2.0 *to any global action is the same in both s_1 and s_2.*

Proof (Sketch). The score for a global action (Eqn. 2) is a function of only ge, and as from Theorem 1 ge is the same for s_1 and s_2; hence the score will remain the same. □

Theorem 1 and Corollary 1 imply that, when all actions are unparameterized, SymNet2.0 can not represent policies that need to differently treat states s_1 and s_2 that are identical to each other, except for the features of n_1 and n_2 being swapped with each other in s_1 and s_2.

Example: Consider the deterministic Navigation domain where a robot has to locate a goal in a 2D-grid (say 23×23) with no obstacles. Let the Boolean predicate `robot_at(x,y)` denote whether the robot is at location (x, y) or not (See supplement for the RDDL domain description). Let us assume SymNet2.0 uses GATs with depth 2. Let the goal be at location $l_g = (10, 10)$. Consider a state s_1 that has the robot at location $l_1 = (5, 10)$ and another state s_2 where the robot is at location $l_2 = (15, 10)$. In either of these states, no node in the network has a view of both the goal and the agent location. Further, it is easy to see due to symmetry, there is one to one correspondence between node embeddings in s_1 to the node embeddings in s_2, resulting in identical global embeddings computed via maxpool. The above theorems state that SymNet2.0 considers both s_1 and s_2 as the same and hence, results in decoding of the identical action in both these states. In other words, it has no way to represent the optimal policy, which corresponds to taking 'move right' action in s_1 and 'move left' action in s_2.

3.2 SymNet3.0: INCORPORATING INFLUENCE

We next present our model Symbolic Influence Network (SymNet3.0), which addresses some of the issues faced by SymNet2.0. Intuitively, SymNet2.0 fails to represent certain desirable policies since its view is limited by the depth of the underlying GAT. Two nodes that are more than $2d$ distance away, with d being the depth of GAT, do not share any neighborhood and hence, have no way to propagate relevant information to each other. Further, maxpool being a permutation invariant operation has no way to capture the relative ordering of nodes in the network. A combination of these issues results in the learning of sub-optimal policies. A natural way to address this would be to simply increase the depth of the GAT. But unfortunately, this leads to blow-up in the number of parameters, potentially causing overfitting. Another approach would be to consider a GAT with parameters tied across layers [Palm et al., 2018] but that still requires passing messages for a long distance, potentially resulting in learnability issues as observed by Zambetta and Thangarajah [2022].

Motivated by these shortcomings, we take a different approach and ask the following question: "Since we have full knowledge of the transition model, is there a way to apriori encode some information in the network which would break the symmetry of states which should actually be different from each other?" Presumably doing so would also help us in learning policies which can discriminate between such similar looking states based on a fixed depth GAT. One way to encode such information would be to capture the distance between two nodes in a graph, where the nodes represent state variables (predicates), and edges represent transitions from one state variable to another, via an action. We note that this may not be possible to do it on the original instance graph, due to presence of a larger number of additional nodes (e.g., singletons) making it too dense, and unsuitable for capturing such a notion of distance.

In the navigation example, this kind of graph would capture the underlying grid structure, since robot can move in either direction in one step via the transition model. This means that if the model is given access to this distance information as a feature, it could represent policies not earlier representable by SymNet2.0, e.g., in our navigation domain, it is better to move in the direction, which minimizes the distance to the goal. In general, some other complicated function of the distance could also be learned, as we show in our experiments. Next, we formally introduce the notion of influence distance followed by changes to the SymNet2.0 architecture to exploit the distance-based features.

3.2.1 Influence Graph and Influence Distance

To succinctly represent the influence among state-variables of a given instance I, we define an *influence graph* I_G as follows: (a) There is a node for each state-variable $P(\langle u \rangle)$ in I_G, and (b) There is a directed edge $(P(\langle u \rangle), Q(\langle v \rangle))$ if state-variable $P(\langle u \rangle)$ affects the state-variable $Q(\langle v \rangle)$ in the following time step based on the transition model in the

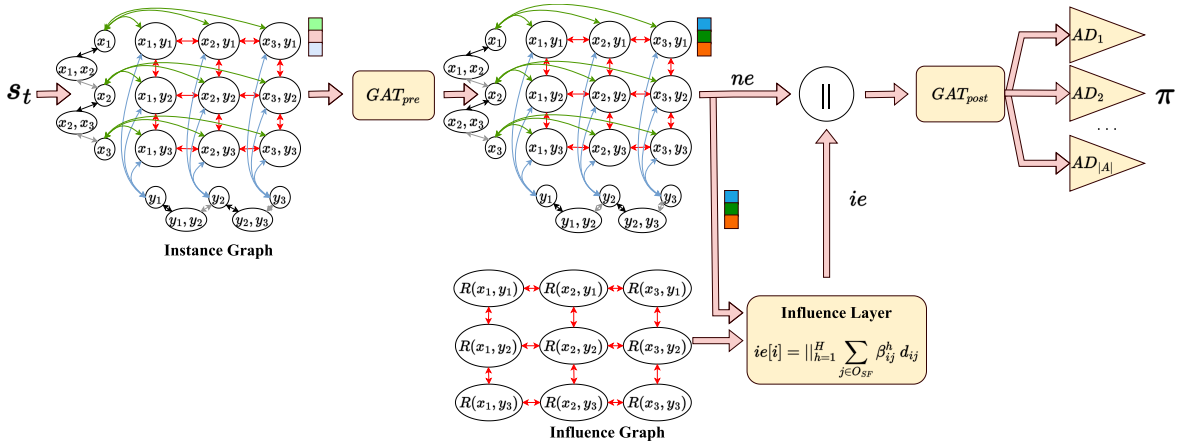

Figure 1: Figure shows the three-step process of SymNet3.0 for policy prediction. The instance graph and influence graph are representative of the Navigation domain (See supplement for domain description). The instance graph has nodes for object-tuples $((x_i, x_j), (y_i, y_j), (x_i, y_j), x_i, y_i)$ and the influence graph has nodes for predicates ($R(x_i, y_j)$ denoting the `robot_at` predicate). In the case of SymNet2.0, only instance-graph is present.

DBN. Intuitively, the influence graph removes the notion of time from the *nodes* present in the DBN and captures dependencies among the state-variables. In the Navigation domain, it will have nodes for `robot_at` state-variables and edges for each neighboring cell (see Fig 1)

Definition 1. *Given an Influence Graph I_G, we define influence distance between two nodes $n_1, n_2 \in I_G$, as the length of the shortest path from n_1 to n_2 in I_G.*

Note that a distance of k between nodes $P(\langle u \rangle)$ and $Q(\langle v \rangle)$ implies that it takes at least k time steps for state-variable $P(\langle u \rangle)$ to influence state-variable $Q(\langle v \rangle)$. Since the influence distance is computed in the influence graph, which is based on the transition model, in general, it will be the distance between two nodes in a directed graph. Next, we describe how this influence distance is incorporated in the SymNet3.0 architecture to learn the desirable policies.

3.2.2 SymNet3.0 Architecture

We use the same instance graph as used in SymNet2.0 i.e., it has the same set of nodes, edges, and input features, modulo one important distinction. As SymNet2.0 has multiple adjacencies in its instance graph, on large instances, the memory requirements become too high, leading to an out-of-memory error. As a simple remedy, we use a single adjacency (in both SymNet2.0 and SymNet3.0), but with edge-types where each edge-type represents the original adjacency it comes from. Therefore, there will be $1+|A|+|Ar|$ edge types corresponding to all the original instance graphs.

To compute node-embeddings in SymNet3.0, we use a three-step process (Figure 1), (i) Compute initial node-embeddings using a fixed-depth pre-process GAT, (ii) compute influence distance among nodes and incorporate it as a feature in instance graph, and (iii) combine initial node-embeddings and distance feature to get final node-embeddings using a fixed-depth post-process GAT. We provide the details below.

Pre-Processing: The information about an object-tuple is provided either as state fluents or non-fluents or both. In the instance graph, the non-fluent based nodes are connected to singleton nodes which are in turn connected to the state fluent based nodes; hence to collate the information on the state variable nodes, we need an initial message-passing step. To compute the initial node-embeddings (ne), we use a single Graph Attention Network [Veličković et al., 2018] called pre-process GAT (GAT_{pre}) that can incorporate edge types as,

$$\alpha_{ij}^{h} = softmax_{\mathcal{N}_i}(LRelu(a^T[W_1^h f_i || W_1^h f_j || W_2^h e_{ij}]))$$
$$ne[i] = ||_{h=1}^{H} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{h} W_3^h f_j \quad (3)$$

Here, f_i and $\mathcal{N}_i$ denote the features and one-hop neighbours of node i. e_{ij} is the one-hot encoding of each edge type. H and $||$ denote the number of attention heads and concatenation operators, respectively. In our experiments, we use GAT_{pre} of depth 2 as this is the minimum number of message passing steps required for information from non-fluent nodes to reach the state fluent nodes.

Incorporating Influence: Since the influence-distance is defined over state-variables, whereas each node in the instance graph is either an object or an object-tuple, we have to first define a mapping from the influence-distance to nodes of instance graph, which is followed by computation of distance feature based *influence-embeddings* (see below).

1. For two nodes $i, j \in O_{SF}$ in the instance graph, we define d_{ij} as the minimum influence distance between any two

state-variables mapped to these nodes. We normalize d_{ij} by dividing it by the maximum value of d_{ij} for that instance. Note that computation of d_{ij} is a static process, done once for each instance. Then, we introduce a novel layer called *influence-layer*, with the goal of capturing the notion of the distance of each node from other nodes (like the goal node in the navigation domain). Since we do not know which nodes are relevant, we use an attention mechanism to figure this out. The influence-embeddings (ie) are thus computed as, $\forall i, j \in O_{SF}$

$$\beta^h_{ij} = softmax_{O_{SF}}(LRelu(a^T[U^h_1 ne[i]||U^h_1 ne[j]||U^h_2 d_{ij}]))$$
$$\forall i \in O_{SF},\ ie[i] = ||^H_{h=1} \sum_{j \in O_{SF}} \beta^h_{ij}\ d_{ij} \quad (4)$$

2. For any other remaining node k in the instance graph, $ie[k] = ||^H_{h=1} 0$.

Intuitively, in equation block 4, each state-variable object-tuple node i assigns a weight β^h_{ij} based on the information on i and j, and how far away they are in the influence space. Further, to diversify the long-range information localization, we encourage our attention heads to assign different weights to different nodes. For this, during training, we add a loss term that maximizes the KL divergence between attention scores (β^h_{ij}) of any two random attention heads of a randomly sampled node $i \in O_{SF}$.

State-Representation: We update the node-embeddings using influence-embeddings and a post-process GAT as, $ne[i] = GAT_{post}(ne[i] \mid\mid ie[i])$ and compute a global embedding as $ge = maxpool_{i \in V} ne[i]$. The use of distance features provides nodes in SymNet3.0 with the capability to focus on some key nodes and learn node-embeddings that break the symmetry induced by fixed-depth GAT as in SymNet2.0.

Action Decoding: Similar to SymNet2.0, we compute action scores using a set of action decoders $\{AD_1, ...AD_{|A|}\}$, and take softmax over all scores to get the policy.

3.3 REPRESENTABILITY

Theorem 3. SymNet3.0 *can represent all policies that* SymNet2.0 *can represent.*

Proof (Sketch). SymNet3.0 subsumes SymNet2.0 as each node can write an 0 vector as its influence-embedding, rendering the weights that process ie inactive, thus reducing SymNet3.0 to SymNet2.0. □

Theorem 4. *For a node n in the influence graph, let $L(n, k)$ denote the multi-set of node features of nodes that are exactly k hops away from node n in the influence graph. In reference to theorem 1, given the features of nodes n_1 and n_2, if there exists a $k > 0$ such that $L(n_1, k) \neq L(n_2, k)$, then, given a sufficiently powerful attention function* SymNet3.0 *has the power to learn the parameters that break the symmetry induced between s_1 and s_2 which have the features of nodes n_1 and n_2 swapped.* [see Supplement for a proof sketch]

Theorem 5. *In reference to Theorem 4, let there be two singleton nodes o_i and o'_i whose features have been swapped in states s_1 and s_2, making these states symmetric to each other with respect to o_i and o'_i. Let $ac_1 = A_C(o_1, ...o_i, ..., o_k)$ be an action applicable in s_1 and $ac_2 = A_C(o_1, ..., o'_i, ..., o_k)$ be an action applicable in s_2, that differ only in the arguments o_i and o'_i. Further, if $\mathcal{P}_{ac_1} \setminus \{o_i\} = \mathcal{P}_{ac_2} \setminus \{o'_i\}$, then,* SymNet3.0 *has the power to assign different action scores to ac_1 and ac_2.*

Proof (Sketch). From theorem 4 SymNet3.0 can learn different node-embeddings of o_i and o'_i thus having the power to give different action scores for ac_1 and ac_2. □

4 EXPERIMENTS

We design our experiments for answering three research questions. (i) How well does SymNet3.0 handle the long-range influence problem in comparison to SymNet2.0? (ii) How do these models compare on domains that do not have long-range dependences? (iii) Can we identify SymNet3.0's strengths and limitations?

4.1 EXPERIMENTAL SETUP

Previous works [Garg et al., 2020, Sharma et al., 2022] have experimented with twelve IPPC 2011 and 2014 domains. Our preliminary analyses indicated that most of those domains do not require solving the long-range dependence problem: either the instances are too small, or policies are too localized. So, we use these domains to answer question (ii) above. We additionally create six new domains, which we name as LR domains, that necessitate recognizing the long-range influences for computing good solutions.

Domains: We now briefly describe the new LR domains (see supplement for further details on all domains),

1) **Deterministic Navigation (DNav):** A robot in a 2D-grid has to reach a far away goal cell in a minimum number of steps. A reward of -1 is given at every time step and 0 on reaching the goal.
2) **Stochastic Corridor Navigation (StNav):** This is a variant of IPPC's Navigation domain. Given a 2D grid, a robot has to reach a goal location, but it can die with a certain probability at each cell, except the bottom and the topmost rows are safe. The robot and goal locations are sampled randomly in the bottom and top rows, respectively. There is a single randomly sampled safe vertical corridor from bottom to top. The IPPC Navigation is a special case of

Model	SRecon	Pizza	DNav	StWall	EAcad	StNav	Mean
PROST	0.34	0.09	0.94	0.69	0.37	0	0.41
SymNet2	0.47	0.26	0.55	0.27	0.9	0.03	0.41
SymNet3-KL	**0.68**	**0.62**	0.84	0.33	0.87	0.08	0.57
SymNet3+KL$_D$	0.62	0.58	0.91	**0.38**	**0.92**	**0.15**	**0.59**
SymNet3+KL	0.61	0.18	**0.95**	0.35	0.91	0.05	0.51

Table 1: Comparison of SymNet3.0 with the baselines on 6 LR domains (bold denotes the best-performing neural model)

.

Model	Tam	Traffic	Sys	Skill	Nav	TT	Recon	Elev	Acad	CT	GoL	Wild	Mean
PROST	0.86	0.91	0.76	0.84	0.00	0.03	0.59	0.91	0.64	0.34	0.32	0.57	0.56
SymNet2	0.90	**0.88**	0.79	**0.82**	0.54	**0.78**	0.35	**0.92**	**0.83**	**0.81**	0.62	0.78	**0.75**
SymNet3-KL	**0.91**	0.85	0.81	0.81	0.53	0.70	**0.42**	0.87	0.73	0.8	**0.76**	**0.79**	**0.75**
SymNet3+KL$_D$	0.90	0.85	**0.83**	0.77	**0.85**	0.67	0.29	0.71	0.78	0.78	0.61	0.77	0.73
SymNet3+KL	0.90	0.85	0.82	0.70	0.71	0.74	0.24	0.91	0.80	0.80	0.41	0.18	0.67

Table 2: Comparison of SymNet3.0 with the baselines on 12 IPPC domains (bold denotes best-performing neural model)

StNav where the safe corridor is always the first column.
3) **Extreme Academic Advising (EAcad):** A variant of IPPC's Acad, EAcad has a set of courses arranged in a directed acyclic graph with some courses as program requirements that the agent has to complete in order to complete the degree. The probability of completing a course without completing all its pre-requisites is very low. Therefore, in the optimal policy, a course should be taken only if it is an ancestor of some far-away program requirement.
4) **Safe Recon (SRecon):** In this modification of IPPC's Recon, there is 2D-grid with multiple objects, and the robot has to locate an object to apply a tool to get a reward. The action may damage the object, so it may need to locate and try on the next object.
5) **Pizza Delivery (Pizza):** This is a new domain, in which a robot in a 2D-grid has to pick pizza from one of the outlets and deliver it to a customer in the shortest time in a windy (stochastic) environment. The robot should choose an outlet that minimizes the total distance, rather than going to the closest one.
6) **Stochastic Wall (StWall):** Another new domain, where a robot has to reach a goal location in a 2D grid, where the grid contains either a horizontal or a vertical wall. Each cell in the wall has a high death probability except for one randomly selected safe passage in between. An agent has to locate the safe passage in the wall and reach the goal.

Training Details: In the spirit of domain-independent generalized planning, we use a single architecture (with fixed hyperparameter setting) on all domains, and the validation set is used only for early stopping. For each LR domain, we generate 1000 training, 100 validation, and 200 test instances with size (#state-fluents) increasing from train to validation to test instances. And for standard IPPC domains, we generate 200 training, 10 validation, and 40 test instances.[5]. See the supplement for details on the instance sizes.

Similar to SymNet2.0, we use state-of-the-art online planner PROST and generate 30 trajectories for each training instance, and train using imitation learning on the first 300 transitions . As PROST is a sampling-based solver, it can end up taking different actions for the same state; we remove this ambiguity by choosing the most frequent action for each state. We train for 48 hours for each of the 6 new domains and for 24 hours each on the 12 IPPC domains. Each checkpoint is evaluated on validation instances, and we pick the one with the best average reward on the validation instances.

Comparison Algorithms: For SymNet3.0, GAT_{pre} and GAT_{post} have depth 2 each, and there are 10 attention-heads in the influence-layer for all domains (supplement has further details). We use the loss function $L_{imit} - \lambda L_{KL}$, where L_{imit} and L_{KL} denote the imitation, KL-based loss and λ is a hyperparameter. We implement three variations: firstly SymNet3.0-KL where $\lambda = 0$, and SymNet3.0+KL where $\lambda = 0.1$. Our eventual goal is indeed to create *one* planner that can work for all domains. To this end, we develop a third variation, SymNet3.0+KL$_D$ where we keep $\lambda = 0.1$ for first 2000 training batches and then linearly decay λ from 1 to 0 in the next 1000 batches. We compare SymNet3.0 with SymNet2.0, the existing state-of-the-art model for this task. For fair comparison, we use a 4 depth GAT to match SymNet3.0's total depth. In addition we also report results of PROST in its default setting. We note that a direct comparison is not meaningful, as PROST uses interleaved planning and execution, whereas other models are offline planners.

As mentioned in Section 3, for both SymNet2.0 and Sym-

[5]Code released at https://github.com/dair-iitd/symnet3

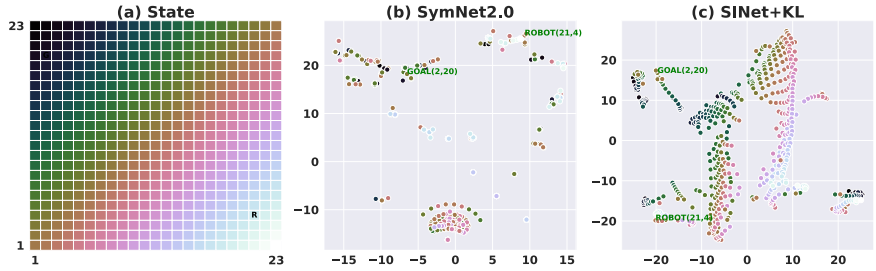

Figure 2: (a) Shows the color-coded locations of a 23×23 instance of the DNav domain where R and G are the robot and goal. Fig. (b) and (c) show the 2D t-SNE plot for node embeddings of the grid locations for SymNet2.0 and SymNet3.0+KL.

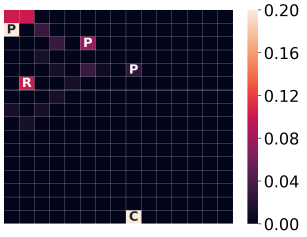

Figure 3: The attention map of the influence-layer in the Pizza domain for the R node.

Net3.0, we use edge-types in one instance-graph, rather than having multiple instance-graphs. This allows both models to avoid high memory requirement issues on larger instances. We tried the original SymNet2.0 setting of multiple instance graphs, but it does not scale to large instances. We also tried dynamically varying the GAT depth proportional to the instance size in SymNet2.0, but this also leads to training issues due to high computational requirements (as observed earlier [Zambetta and Thangarajah, 2022]).

Evaluation metric: Following SymNet2.0, for a given domain, we calculate a relative performance score for a method m on an instance i as, $\alpha(m,i) = \frac{V_m(i) - V_{rand}(i)}{V_{max}(i) - V_{rand}(i)} \in (-\infty, 1]$, where $V_m(i)$ and $V_{rand}(i)$ denote method m's and random policy's reward, respectively. And, $V_{max}(i)$ denotes the best reward by any method on instance i. Here, a value of 0 marks the random policy score, and 1 implies the best performance across all methods on all runs. Next, we calculate $\alpha(m) = \frac{1}{|I_{test}|} \sum_{i \in I_{test}} \alpha(m,i)$, which is method m's score averaged over all test instances (I_{test}). Finally, we report $\alpha(m)$ averaged over 5 independent runs.

5 RESULTS

Tables 1 and 2 show our results where each $(i,j)^{th}$ entry gives the α value of i^{th} model on the j^{th} domain. The bold numbers show the best-performing neural method. The results for PROST are in gray as it is not a direct comparison. The last column reports the average over all domains.

Long Range Domains: Table 1 shows that all variations of SymNet3.0 outperform the improved baseline SymNet2.0 on all 6 new LR domains on the mean aggregate metric with a margin of $+10$ $\alpha(m)$ points for SymNet3.0+KL, $+16$ for SymNet3.0-KL, and $+18$ for SymNet3.0+KL$_D$. Interestingly, SymNet3.0+KL$_D$ outperforms SymNet2.0 on all 6 LR domains. Interestingly, SymNet3.0+KL$_D$ is able to achieve a score of $+15$ on StNav, where PROST and SymNet2.0 fail to give any meaningful policy, highlighting the inherent difficulty of the domain. Similarly, for Pizza domain, PROST again fails to perform well, and both SymNet3.0-KL and SymNet3.0+KL$_D$ get a score of greater than $+55$ as compared to SymNet2.0's $+26$.

IPPC Domains: SymNet3.0-KL performs at par with SymNet2.0 on the IPPC domains. However, overall, SymNet3.0+KL's performance drops in comparison to SymNet2.0, specifically on Skill, GoL, and Wild domains. We hypothesize that as the training data increases, it should perform at par with SymNet2.0and we leave this analysis for future work.

Use of KL divergence loss: In general, we note that using KL-based loss improves performance in some domains. However, having a KL-based loss doesn't give better performance consistently across all domains. We hypothesize that this is because the KL loss enforces a strong inductive bias, where all attention heads must focus on different nodes in the graph. Hence, in the case of domains without long-range dependency, it could lead to attention on irrelevant nodes causing overfitting. We also observed in certain domains that sometimes the KL loss could lead to convergence problems during training. The investigation of this phenomenon is left for future work. The overall performance of unified model SymNet3.0+KL$_D$ is best among all baselines for LR domains, and marginally lower than SymNet2.0 for IPPC domains suggesting that the SymNet3.0+KL$_D$ architecture is robust across multiple types of domains.

Model selection based on validation reward: Further, when we select the best among SymNet3.0's variations based on the validation set, we notice that for both IPPC and LR settings, this model gives a slight boost in the overall performance in comparison to the earlier best model. Additionally, SymNet3.0 marginally outperforms SymNet2.0 in the IPPC setting (see Supplement for results).

5.1 INSIGHTS

Visualizing node embeddings: Figure 2 shows the node embeddings of the locations of a DNav instance of size 23×23 as computed by SymNet3.0+KL and SymNet2.0. Each grid location is color-coded (Figure 2(a)) and is marked as a circle in Figure 2(b) and (c) where its 2-dimensional t-SNE

embedding is used as the circle's location. SymNet3.0's node-embeddings retain the structure of a 2D grid. In comparison, SymNet2.0 does not exhibit any such structure.

Visualizing influence-layer: Figure 3 shows an instance of the Pizza domain with 3 pizza outlets (P), one customer (C), and a robot (R). Figure 3 shows the attention map (β_{ij}) of the influence-layer, averaged over all heads where i is the node with the robot (R). We observe that SymNet3.0 automatically learns to assign a high attention score to the key nodes having information of Pizza outlets (P) and the customer (C). This provides deeper insight into the learned policy. Further, we observe that in this instance, the learned policy by our model is the one that takes the robot to the P, which minimizes the total distance to C. We observe similar qualitative behavior for other domains (see supplement).

6 RELATED WORK

Generalized Planning: Earlier works for learning generalized policies for relational planning focus on learning generalized features that can be transferred across instances [Fern et al., 2003, Guestrin et al., 2003, Mausam and Weld, 2003]. Recent works try to learn generalized policies using deep neural networks for both PPDDL [Toyer et al., 2018, Ståhlberg et al., 2022a,b] and RDDL [Issakkimuthu et al., 2018, Garg et al., 2019, 2020, Sharma et al., 2022]. Ståhlberg et al. [2022a,b] argue that the policies that can not be written in two variable counting logic can not be represented using Graph Neural Networks. They also highlight the problem of long-range dependencies; however, they do not propose any solution. ASNet [Toyer et al., 2018] also focuses on PDDL (rather than RDDL), and has a tight coupling with an online planner to learn generalized neural policies for PPDDL. PPDDL and RDDL differ in their modeling choices; for example, PPDDL provides an explicit goal state definition, whereas RDDL does not. Neural solvers for both of these depend heavily on these facts; for example, ASNet relies on the availability of a goal state. Further, automatically converting a domain from one to another would first require grounding the representation, losing the first-order semantics. Hence, it is difficult to have a direct comparison. Another work by Silver et al. [2021] learns to predict objects' importance with the goal of pruning the number of objects. However, their target is to speed up planning rather than generalize and hence not directly comparable to ours.

To the best of our knowledge, work by Issakkimuthu et al. [2018] was the first to learn policies using neural networks for RDDL RMDPs; however, they do not learn generalized policies. A sequence of works [Bajpai et al., 2018, Garg et al., 2019, 2020, Sharma et al., 2022] learns generalized neural policies for RDDL RMDPs.

General Graph Neural Network techniques: Skip connections-based approaches like JK-net [Xu et al., 2018] focus on improving the learnability when the depth of the GNN is increased but do not affect the representation problem of long-range dependence. Another approach is to use hierarchical GNNs based on Pooling approaches like Diffpool [Ying et al., 2018] that stack blocks of message passing and pooling blocks. However, these approaches select and deselect nodes to be grouped together based on a learned score and hence alter the notion of distance among nodes – a notion critical in the planning problems. Moreover, to handle size transfer, an architecture with a varying number of message passing and pooling blocks are needed to handle large instances. Hence, for any fixed-sized hierarchical GNN, there is always a large enough instance such that the given network does not have the capacity to capture all the long-range dependencies.

7 CONCLUSION AND FUTURE WORK

We have studied the problem of capturing long-range dependencies in neural architectures for learning policies in RDDL RMDPs. We have proposed SymNet3.0, which defines the novel notion of influence graph defined over state variables, with edges representing transitions between them. The distance in the influence graph is incorporated as a feature in the instance graph, to represent long range dependencies, and the corresponding policies are learned using a multi-headed attention architecture. Extensive experimentation shows that our approach is competitive on 12 IPPC domains, and does significantly better on six domains designed to test long range dependencies, in comparison with SOTA baselines.

One of the limitations of our work is the dependence on an existing planner to generate training dataset for imitation learning. Integrating SymNet3.0 with RL for learning generalised policies is a direction for future work. Another potential limitation is that we only consider pairwise distances between nodes - it may not capture policies which simultaneously depend on distances among a set of nodes; this is another direction for future work. Applying our approach to the PPDDL-based RMDPs is also a direction for future work.

Acknowledgements

This work is supported by IBM AI Horizon Networks (AIHN) grant. Parag Singla is supported by IBM SUR awards. Mausam is supported by grants from Huawei, Google, Verisk, and a Jai Gupta Chair Fellowship. We thank the IIT Delhi HPC facility for computational resources. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views or official policies, either expressed or implied, of the funding agencies.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=i80OPhOCVH2.

Aniket Bajpai, Sankalp Garg, and Mausam. Transfer of deep reactive policies for MDP planning. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 10988–10998, 2018.

Craig Boutilier, Raymond Reiter, and Bob Price. Symbolic dynamic programming for first-order mdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 690–700, 2001.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. *Advances in neural information processing systems*, 16, 2003.

Sankalp Garg, Aniket Bajpai, and Mausam. Size independent neural transfer for RDDL planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 631–636, 2019.

Sankalp Garg, Aniket Bajpai, and Mausam. Symbolic network: generalized neural policies for relational mdps. In *International Conference on Machine Learning*, pages 3397–3407, 2020.

Edward Groshev, Aviv Tamar, Maxwell Goldstein, Siddharth Srivastava, and Pieter Abbeel. Learning generalized reactive policies using deep neural networks. In *2018 AAAI Spring Symposium Series*, 2018.

Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational mdps. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1003–1010, 2003.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Murugeswari Issakkimuthu, Alan Fern, and Prasad Tadepalli. Training deep reactive policies for probabilistic planning problems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 2018.

Thomas Keller and Patrick Eyerich. Prost: Probabilistic planning based on uct. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

Mausam and Daniel Weld. Solving relational MDPs with first-order machine learning. In *Proceedings of the Workshop on Planning under Uncertainty and Incomplete Information, at ICAPS*, 2003.

Rasmus Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. *Advances in neural information processing systems*, 31, 2018.

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32:27, 2010.

Vishal Sharma, Daman Arora, Florian Geißer, Mausam, and Parag Singla. Symnet 2.0: Effectively handling non-fluents and actions in generalized neural policies for rddl relational mdps. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1771–1781. PMLR, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/sharma22a.html.

Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Planning with learned object importance in large problem instances using graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11962–11971, 2021.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. *Proceedings of the 32nd International Conference on Automated Planning and Scheduling*, 2022a.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning generalized policies without supervision using gnns. *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, 2022b.

Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2018.

Tailin Wu, Hongyu Ren, Pan Li, and Jure Leskovec. Graph information bottleneck. *Advances in Neural Information Processing Systems*, 33:20437–20448, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5453–5462. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/xu18c.html`.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Håkan LS Younes, Michael L Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24:851–887, 2005.

Fabio Zambetta and John Thangarajah. Oracle-sage: Planning ahead in graph-based deep reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2022.

SymNet 3.0: Exploiting Long-Range Influences in Learning Generalized Neural Policies for Relational MDPs (Supplementary Material)

Vishal Sharma[*1] **Daman Arora**[*1] **Mausam**[1] **Parag Singla**[1]

[1]Indian Institute of Technology Delhi {vishal.sharma, cs5180404, mausam, parags}@cse.iitd.ac.in

1 PROOFS

Theorem 4. *For a node n in the influence graph, let $L(n, k)$ denote the multi-set of node features of nodes that are exactly k hops away from node n in the influence graph. In reference to theorem 1, given the features of nodes n_1 and n_2, if there exists a $k > 0$ such that $L(n_1, k) \neq L(n_2, k)$, then, given a sufficiently powerful attention function* SymNet3.0 *has the power to learn the parameters that break the symmetry induced between s_1 and s_2 which have the features of nodes n_1 and n_2 swapped.*

Proof (Sketch). The high level intuition of the proof is that their will exist certain "key-nodes" in the graph which have unique features. For example, in the Navigation domain, it will be the Goal location. For the Pizza domain, it would be the location of the Pizza and the Customer. If a node is using distance from these key-nodes, then it is possible to break symmetries induced by a fixed-depth GAT. The more the number of key-nodes, the easier it is to break the symmetry. The formal proof is as follows:

Let $C_{f,n,d}$ denote the number of times f occurs in $L(n, d)$. As $L(n_1, k) \neq L(n_2, k)$, $\exists f'$, s.t. $C_{f',n_1,k} \neq C_{f',n_2,k}$. With a sufficiently powerful attention function, in the state s_1, node n_1 can focus attention on f' to learn a node embedding different from that of n_2.

We construct one such set of parameters for SymNet3.0 that will break the symmetry among s_1 and s_2 with respect to nodes n_1 and n_2.

1. GAT_{pre} can learn an identity mapping for each node by focusing all attention on itself and those nodes in its neighbourhood that have exactly the same features as itself while ignoring all other neighbours.
2. Next, consider the following (un-normalized) attention function in the influence layer.

$$e(f_i, f_j, d_{ij}) = \begin{cases} 0 & d = 0 \\ 0 & f_j = f', d_{ij} = k \\ -INF & \text{otherwise} \end{cases}$$

   where INF is a very large positive number.
3. Next, GAT_{post} can also learn an identity mapping (similar to GAT_{pre}).

The above parameters ensure that, in the influence layer, any given node gives a non-zero attention weight (after normalization) to itself and to any other node at a distance k having features f'. In state s_1, n_1's attention is spread over n_1 and those nodes at a distance k that have f' as their features. Therefore the influence embedding for n_1 in s_1 will be $\frac{C_{f',n_1,k}}{C_{f',n_1,k}+1} * k$.

[*]Equal Contribution

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

Similarly for n_2 in s_2, it will be $\frac{C_{f',n_2,k}}{C_{f',n_2,k}+1} * k$. Since $C_{f',n_1,k} \neq C_{f',n_2,k}$, the embeddings will be different when the features are swapped, thus breaking the symmetry among n_1 in s_1 and n_2 in s_2. □

An example of attention function in the influence layer that can break symmetry: Additionally, we also provide an explicit construction of attention weights of the influence layer, that is independent of f'. Assume that the features of nodes come from a finite-ordered set F and there exists a function $idx_F : F \to \mathbb{N}$ that returns the index of a feature in the ordered-set F. Consider the un-normalized attention function for $m, n \geq 1$,

$$a_{m,n}(f_i, f_j, d_{ij}) = \begin{cases} 0 & d_{ij} = 0 \\ 0 & idx(f_j) = m \text{ and } d_{ij} = n \\ -INF & \text{otherwise} \end{cases} \tag{1}$$

where $-INF$ is a very large negative number. Since the influence layer has multi-head attention, we can assign each head with a different attention function. Specifically, we assign $a_{m,n}$ to the $(n|F| + m)^{th}$ attention head. Note that if a graph has $|G|$ nodes, this ensures there are atmost $|G||F|$ attention heads.

Note that given these attention heads, it is possible to encode the multi-set of neighbours at a distance k in the influence embedding! Let $C_{f,n,d}$ denote the number of times feature f occurs in the d-hop neighbour of node n. If we're given that $L(n_1, k) \neq L(n_2, k)$, we can say that $\exists f' \in F$ such that $C_{f,n_1,k} \neq C_{f,n_2,k}$.

Consider the embedding of node n_1 in state s_1, specifically the $a^{th}_{idx(f'),k}$ attention head which would correspond to the $(k|F| + idx(f'))^{th}$ element of the influence embedding. Equal attention would be spread over n_1 and $C_{f',n_1,k}$ nodes. Therefore the aggregated distance would be $\frac{C_{f',n_1,k}}{1+C_{f',n_1,k}} * k$. Correspondingly for n_2 in s_2, this element would be $\frac{C_{f',n_2,k}}{1+C_{f,n_2,k}} * k$. Since $C_{f',n_1,k} \neq C_{f',n_2,k}$ the embedding for n_1 in s_1 would not equal the embedding for n_2 in s_2. A similar argument can be made for n_2 in s_1 and n_1 in s_2. In practice this kind of a construction would require the dimension of the heads to scale with the size of the graph, however this is an exaggeration in the practical setting. In practical domains, there are only a fixed-small number of key features, and just considering the distance from them is sufficient for computing the policy.

2 RDDL EXAMPLE

The IPPC domain of Navigation is a 2D grid world where a robot has to reach a goal cell. Each cell in the grid has a death probability with which the robot can die. The agent receives a $+1$ reward for reaching the goal and 0 otherwise.

Object Types: `x, y`

Non-Fluents: `north(y, y), south(y, y), east(x, x), west(x, x), min_x(x), max_x(x), prob(x, y), goal(x, y)`

State-Fluents: `robot_at(x, y)`

Actions: `move_north, move_south, move_east, move_west`

3 EXPERIMENTAL DETAILS

- **Data generation:** For each domain, we generate 1000 training, 100 validation, and 200 test instances with size increasing from train to val to test instances. Similar to SYMNET2.0, we use state-of-the-art online planner PROST and generate 30 trajectories of each training instance using the default settings.
- **Architectural Details**: For our experiments with SYMNET2.0, we use a GAT with depth 4, having shared weights across layers, each layer having 10 attention heads. For SYMNET3.0, both the pre-processing and post-processing GATs are of depth 2 and have 10 attention heads, with shared weights. For SYMNET3.0, the influence layer uses 10 attention heads. The final node embedding dimension for both models is 20, and action decoders used are MLPs with 1 hidden layer of dimension 20.
- **Training details**: We train all models for 48 hours on a $K40$ GPU using imitation learning for LR domains and for 24 hours for IPPC domains. Each checkpoint is evaluated on validation instances and we pick the one with best average for testing.

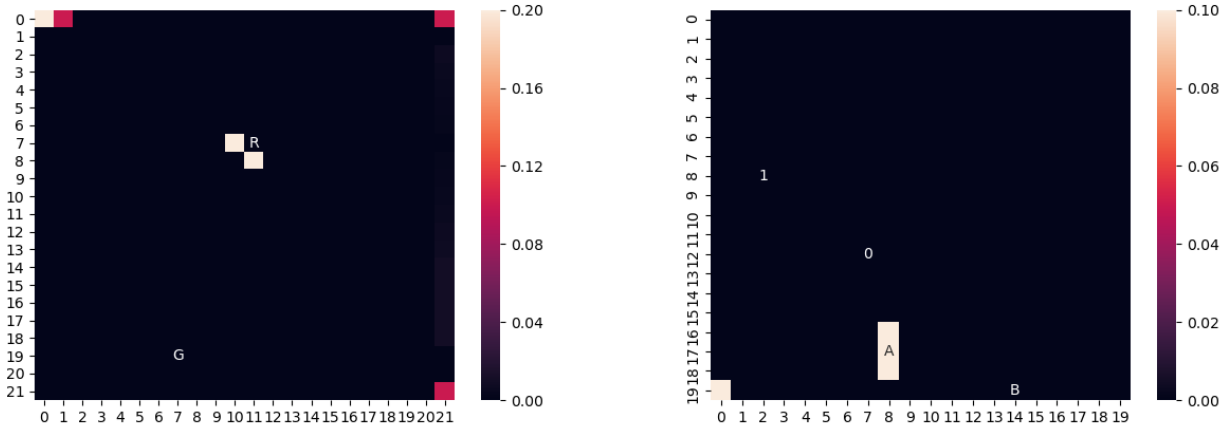

Figure 1: (left) Figure shows the attention map averaged across all heads for the robot's location computed by SYMNET3.0+KL for the DNav domain. We note that the attention heads focus on the corners of the grid helping in the localization of all nodes. (right) Figure shows the attention map averaged across all heads for the robot's location computed by SYMNET3.0+KL for the SRecon domain. Here, 0 and 1 denote the object 0 and object 1. We note that the attention heads focus on one of the corners of the grid.

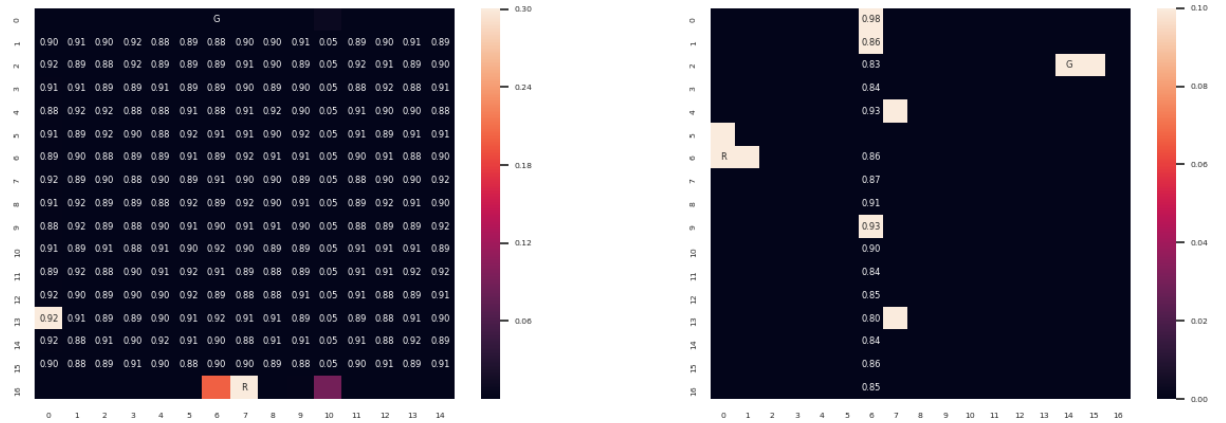

Figure 2: (left) Figure shows the attention map averaged across all heads for the robot's location computed by SYMNET3.0+KL for the StNav domain. Here, the probability of death of each cell is written on the cell. We note that the attention is focused on the entrance of the column which is safest. (right) Figure shows the attention map averaged across all heads for the robot's location computed by SYMNET3.0+KL for the StWall domain. Here, the attention is focuses on the goal and the cells near the safe passage cell.

4 DETAILED RESULTS AND ATTENTION MAPS

The detailed results of experiments for each run of various models for LR domains is shown in Table 1 and for IPPC domains is shown in Table 2

Tables 3 and 4 show the results when the best model among SYMNET3.0-KL, SYMNET3.0+KL$_D$ and SYMNET3.0+KL is chosen based on validation scores..

Model	SRecon	Pizza	DNav	StWall	EAcad	StNav	Mean
PROST	0.34	0.09	0.94	0.69	0.37	0	0.67
SymNet2[1]	0.49	0.22	0.74	0.26	0.89	0	0.63
SymNet2[2]	0.47	0.35	0.57	0.23	0.89	0.01	0.56
SymNet2[3]	0.49	0.27	0.46	0.27	0.9	0	0.54
SymNet2[4]	0.43	0.11	0.43	0.31	0.9	0.13	0.55
SymNet2[5]	0.47	0.33	0.57	0.27	0.9	0.03	0.58
SymNet3-KL[1]	0.63	0.65	0.86	0.47	0.95	0	0.76
SymNet3-KL[2]	0.63	0.43	0.83	0.31	0.72	0.2	0.62
SymNet3-KL[3]	0.73	0.69	0.87	0.42	0.81	0	0.7
SymNet3-KL[4]	0.7	0.64	0.8	0.24	0.9	0.08	0.65
SymNet3-KL[5]	0.72	0.66	0.86	0.24	0.95	0.09	0.68
SymNet3+KL$_D$[1]	0.64	0.42	0.96	0.44	0.96	0.43	0.79
SymNet3+KL$_D$[2]	0.55	0.82	0.91	0.31	0.9	0.01	0.71
SymNet3+KL$_D$[3]	0.59	0.55	0.88	0.4	0.86	0.11	0.71
SymNet3+KL$_D$[4]	0.72	0.44	0.92	0.31	0.94	0.04	0.72
SymNet3+KL$_D$[5]	0.61	0.67	0.88	0.41	0.95	0.17	0.75
SymNet3+KL[1]	0.46	0.09	0.92	0.43	0.92	0.05	0.76
SymNet3+KL[2]	0.65	0.31	0.93	0.33	0.91	0.02	0.72
SymNet3+KL[3]	0.67	0.14	0.97	0.27	0.94	0.03	0.73
SymNet3+KL[4]	0.66	0.18	0.93	0.36	0.95	0.15	0.75
SymNet3+KL[5]	0.61	0.18	0.98	0.36	0.83	0	0.72

Table 1: Performance of all runs of different models on 6 LR domains.

5 SIZES OF INSTANCES

In the spirit of transfer, the sizes of instances increase from training to validation to test instances. A measure of size is the number of state fluents present in the instance. We report the minimum and maximum of train, validation and the test sets for LR domains in table 5 and for IPPC domains in table 6.

6 DOMAINS AND GENERATORS

1. **Deterministic Navigation (DNav)**
   Deterministic Navigation involves a Robot and a Goal cell located in a square grid. For each step that the Robot is not in the Goal cell, it receives a reward of -1. The optimal policy requires the Robot to reach the Goal in the minimum number of timesteps. To generate instances, first the grid size is sampled uniformly from $[D_{min}, D_{max}]$ and then the goal and start cell of the robot is samples uniformly from the grid cells. Parameters for generation:
   (a) D_{min}: Minimum allowed grid dimension
   (b) D_{max}: Maximum allowed grid dimension.
   To generate the train, validation, and test sets, we use the parameters mentioned in Table 7
2. **Extreme Academic Advising (EAcad)** Extreme Academic Advising consists of various courses which are arrange in a Directed Acyclic Graph. Certain courses are program requirements. For each time step, every program requirement that is not completed adds a negative reward to the total reward. In order to get high reward, an agent must complete program requirements in the shortest amount of time possible. If all the pre-requisites of a course have been completed then the probaility of completion of the course when attempted is 0.95. Otherwise, the probability of completion is 0.05. This incentivizes an agent to complete courses in the DAG order specifically leaving out courses which are not ancestors to a requirement course. To generate the courses, we set L which is the number of levels, and C the number

Model	Tam	Traffic	Sys	Skill	Nav	TT	Recon	Elev	Acad	CT	Wild	GoL	Mean
PROST	0.86	0.91	0.76	0.84	0	0.03	0.59	0.91	0.64	0.34	0.32	0.57	0.56
SymNet2[1]	0.91	0.9	0.76	0.84	0.09	0.76	0.41	0.9	0.88	0.65	0.71	0.82	0.72
SymNet2[2]	0.92	0.88	0.72	0.78	0.69	0.79	0.3	0.91	0.85	0.91	0.51	0.83	0.76
SymNet2[3]	0.89	0.87	0.85	0.86	0.59	0.82	0.3	0.92	0.9	0.76	0.47	0.72	0.75
SymNet2[4]	0.89	0.9	0.81	0.83	0.45	0.76	0.35	0.95	0.64	0.89	0.63	0.8	0.74
SymNet2[5]	0.89	0.86	0.82	0.79	0.88	0.77	0.37	0.93	0.86	0.82	0.78	0.75	0.79
SymNet3-KL[1]	0.92	0.85	0.87	0.85	0.78	0.77	0.21	0.92	0.89	0.76	0.88	0.82	0.79
SymNet3-KL[2]	0.91	0.89	0.82	0.84	0.29	0.59	0.41	0.88	0.68	0.84	0.8	0.75	0.73
SymNet3-KL[3]	0.93	0.83	0.75	0.84	0.2	0.8	0.61	0.88	0.62	0.81	0.64	0.78	0.72
SymNet3-KL[4]	0.91	0.84	0.79	0.77	0.88	0.53	0.48	0.89	0.65	0.77	0.74	0.8	0.75
SymNet3-KL[5]	0.89	0.82	0.83	0.73	0.5	0.79	0.41	0.76	0.79	0.8	0.73	0.82	0.74
SymNet3+KL$_D$[1]	0.9	0.85	0.81	0.76	0.87	0.77	0.23	0.94	0.92	0.77	0.66	0.79	0.77
SymNet3+KL$_D$[2]	0.89	0.86	0.83	0.83	0.84	0.82	0.36	0.28	0.55	0.86	0.66	0.83	0.72
SymNet3+KL$_D$[3]	0.9	0.83	0.85	0.75	0.86	0.76	0.19	0.81	0.78	0.81	0.58	0.69	0.73
SymNet3+KL$_D$[4]	0.91	0.87	0.84	0.84	0.8	0.8	0.37	0.86	0.8	0.72	0.7	0.76	0.77
SymNet3+KL$_D$[5]	0.9	0.86	0.8	0.67	0.87	0.22	0.31	0.66	0.86	0.76	0.47	0.77	0.68
SymNet3+KL[1]	0.91	0.86	0.82	0.63	0.85	0.81	0.18	0.93	0.8	0.76	0.53	0.65	0.73
SymNet3+KL[2]	0.9	0.86	0.78	0.7	0.85	0.8	0.29	0.9	0.89	0.72	0.56	-0.27	0.67
SymNet3+KL[3]	0.91	0.86	0.84	0.68	0.87	0.66	0.27	0.9	0.75	0.89	0.21	0.76	0.72
SymNet3+KL[4]	0.89	0.81	0.83	0.74	0.72	0.74	0.21	0.92	0.88	0.83	0.4	-0.1	0.66
SymNet3+KL[5]	0.9	0.86	0.83	0.75	0.27	0.7	0.26	0.9	0.67	0.8	0.36	-0.13	0.6

Table 2: Performance of all runs of different models on 12 IPPC domains.

of courses per level. Additionally, each courses has, on average p number of prerequisites from the previous level. The number of course requirements is R, and the are sampled with a probability proportional to the square of their level. This is done so as to choose courses which require a lot of pre-requisites to be completed in order. Parameters of generation are:

(a) L_{min}: Minimum number of levels
(b) L_{max}: Maximum number of levels
(c) C_{min}: Minimum number of courses per level
(d) C_{max}: Maximum number of courses per level
(e) p: Average number of pre-requisites

To generate train, val, and test sets, we use the parameters mentioned in Table 8.

3. **Safe Recon (SRecon)** In Safe Recon, an agent has to traverse on a rectangular grid and take pictures of object where it detects life. Once an agent reaches at an object, it must apply tools("water" and "life"), in the correct order(first water, then life). Tools can fail with some probability. Once life has been detected, the agent can take pictures which gives it positive reward until the end of the episode. If an tool is damaged, it can go back to BASE to repair its tool or use damaged tools. Using damaged tools is risky because the a negative reward is given for each photo clicked which doesn't have life. This domain is identical to the one used for IPPC 2014, with the difference being that we do not use HAZARDS in our version. The parameters for instance generation are:

(a) D_{min} Minimum grid size
(b) D_{max} Maximum grid size
(c) O_{min} Minimum number of objects
(d) O_{max} Maximum number of objects
(e) p_{min} Minimum threshold for tool damage probability
(f) p_{max} Maximum threshold for tool damage probability

Model	SRecon	Pizza	DNav	StWall	EAcad	StNav	Mean
PROST	0.34	0.09	0.94	0.69	0.37	0	0.41
SymNet2	0.47	0.26	0.55	0.27	0.9	0.03	0.41
SymNet3-KL	**0.68**	**0.62**	0.84	0.33	0.87	0.08	0.57
SymNet3+KL$_D$	0.62	0.58	0.91	**0.38**	**0.92**	**0.15**	0.59
SymNet3+KL	0.61	0.18	**0.95**	0.35	0.91	0.05	0.51
SymNet3(best val)	0.68	0.58	**0.95**	0.38	0.91	0.15	**0.61**

Table 3: Comparison of SymNet3.0 variants with the baselines on 6 LR domains. The last row denotes the score of the best among SymNet3.0+KL, SymNet3.0-KL and SymNet3.0-KL_D chosen on the basis of average validation reward.

Model	Tam	Traffic	Sys	Skill	Nav	TT	Recon	Elev	Acad	CT	GoL	Wild	Mean
PROST	0.86	0.91	0.76	0.84	0	0.03	0.59	0.91	0.64	0.34	0.32	0.57	0.56
SymNet2	0.9	**0.88**	0.79	**0.82**	0.54	**0.78**	0.35	**0.92**	**0.83**	**0.81**	0.62	0.78	**0.75**
SymNet3-KL	**0.91**	0.85	0.81	0.81	0.53	0.7	**0.42**	0.87	0.73	0.8	**0.76**	**0.79**	0.75
SymNet3+KL$_D$	0.9	0.85	**0.83**	0.77	**0.85**	0.67	0.29	0.71	0.78	0.78	0.61	0.77	0.73
SymNet3+KL	0.9	0.85	0.82	0.7	0.71	0.74	0.24	0.91	0.8	0.8	0.41	0.18	0.67
SymNet3(best val)	0.91	0.85	0.83	0.81	0.85	0.67	0.42	0.91	0.78	0.8	0.76	0.79	**0.78**

Table 4: Comparison of SymNet3.0 variants with the baselines on 12 IPPC domains. The last row denotes the score of the best among SymNet3.0+KL, SymNet3.0-KL and SymNet3.0-KL_D chosen on the basis of average validation reward.

To generate the train, validation, and test sets we use the parameters mentioned in Table 9.

4. **Pizza Delivery (Pizza)** Pizza Delivery consists of a rectangular grid of width w and height h. In addition, the grid contains d pizza shops which are subgoals. The agent must collect the pizza from any pizza shop and deliver it to the customer in the minimum time possible. Additionally, a wind blows which can randomly push you to any neighbouring cell. To generate domains, a start location s, customer location c and a special pizza shop location p' is sampled uniformly randomly. Next, to sample the remaining $d-1$ goals, we remove all cells p such that

$$dist(s,p) \geq dist(s,p')$$

or

$$dist(s,p) + dist(c,p) \leq dist(s,p') + dist(c,p')$$

This is done so that the planner doesn't go to the nearest pizza shop but learns to minimize the sum of both distances(distance to shop+distance to customer). Out of the candidate cells, we sample with probability proportional to the distance from s. Parameters for generation:

(a) w_{min}: Minimum grid width
(b) w_{max}: Maximum grid width
(c) h_{min}: Minimum grid height
(d) h_{max} Maximum grid height
(e) d_{min}: Minimum number of pizza shops
(f) d_{max} Maximum number of pizza shops

To generate the train, validation, and test sets we use the parameters mentioned in Table 10.

Domain	Train(min)	Train(max)	Val(min)	Val(max)	Test(min)	Test(max)
SRecon	48	193	208	249	373	924
Pizza	34	153	205	265	409	493
EAcad	8	36	42	192	120	320
StWall	25	100	121	225	256	400
DNav	81	196	225	324	400	625

Table 5: Number of state fluents for LR domains for training, validation, and test sets.

Domain	Train(min)	Train(max)	Val(min)	Val(max)	Test(min)	Test(max)
Acad	4	50	60	96	120	240
CT	12	84	112	144	180	312
GoL	4	36	42	64	81	100
Skill	6	42	12	42	30	60
Recon	29	81	68	107	120	257
TT	12	75	108	147	192	300
Wild	10	72	72	128	128	288
Tam	2	48	28	84	48	140
Elev	9	32	18	32	24	60
Traffic	32	80	32	80	56	104
Sys	2	15	2	14	15	25
Nav	9	49	25	90	120	120

Table 6: Number of state fluents for IPPC domains for training, validation and test sets.

	D_{min}	D_{max}	Horizon
Train	9	14	40
Val	15	18	60
Test	20	25	60

Table 7: Table shows the parameters used in to generate instances in the DNav domain.

	L_{min}	L_{max}	C_{min}	C_{max}	p	Horizon
Train	2	8	2	8	1	40
Val	7	12	3	8	1	100
Test	12	20	5	8	1	200

Table 8: Table shows the parameters used in to generate instances in the EAcad domain.

	D_{min}	D_{max}	O_{min}	O_{max}	p_{min}	p_{max}	Horizon
Train	6	13	2	4	0	0.5	40
Val	13	14	2	4	0	0.5	80
Test	18	19	2	4	0	0.5	80

Table 9: Table shows the parameters used in to generate instances in the SRecon domain.

	w_{min}	w_{max}	h_{min}	h_{max}	d_{min}	d_{max}	Horizon
Train	5	12	5	12	2	4	100
Val	14	16	14	16	2	4	150
Test	20	22	20	22	2	4	200

Table 10: Table shows the parameters used in to generate instances in the Pizza domain.

	n_{min}	n_{max}	Horizon
Train	5	10	40
Val	11	15	40
Test	16	20	40

Table 11: Table shows the parameters used in to generate instances in the StWall domain.

5. **Stochastic Wall (StWall)** Stochastic Wall consists of a square grid of dimension n. Uniformly randomly, a row or a column is sampled to form a barrier. The start and goal location are sampled such that they lie on opposite sides of the barrier. In the barrier, a cell is selected to be a safe passageway from one part of the grid to the other. Hitting the barrier can cause death with some probability sampled from $\mathcal{U}(0.8, 1.0)$. The agent must navigate from his initial position to the goal. The generation parameters are:

	w_{min}	w_{max}	h_{min}	h_{max}	Horizon
Train	5	10	5	10	40
Val	11	15	11	15	40
Test	15	20	15	20	40

Table 12: Table shows the parameters used in to generate instances in the StNav domain.

(a) n_{min}: Minimum size of the grid.
(b) n_{max}: Maximum size of the grid.

To refer to the generation parameters for train, val and test splits, refer to table 11.

6. **Stochastic Navigation (StNav)** This domains consists of a grid of width w and height h. The bottom and top rows are safe. The start state is sampled from the bottom row and the goal state is sampled in the top row. In the middle rows, the robot can die with some probability. However, there exists a single column which has very low death probability from $\mathcal{U}(0.045, 0.055)$. For all other column, the death probability in each cell is from $\mathcal{U}(0.88, 0.92)$. This task has long range dependencies because the agent has to decide which column to enter into. The column could be very far and thus SYMNET2.0might not be able to decide which direction to take. The generation parameters are:
   (a) w_{min}: Minimum width of the grid.
   (b) w_{max}: Maximum width of the grid.
   (c) h_{min}: Minimum height of the grid.
   (d) h_{max}: Maximum height of the grid.

   To refer to the generation parameters for train, val and test splits, refer to table 12