Exploiting Long-Range Influences in Learning Generalized Neural Policies for RDDL Relational MDPs

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* (SINET), 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 SINET 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 RDDL [Sanner, 2010] and PPDDL [Younes et al., 2005], 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 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 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

[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

Submitted to the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023). **To be used for reviewing only**.

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* (SINET), builds on existing work [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 SINET 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, (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, SINET performs better than its closest competitor SymNet2.0 in all domains with a gain of 19% relative performance in the aggregate metric. Further analysis reveals that the influence-layer of SINET learns to focus attention on key nodes in the network central to capture the long-term dependencies. We will release the code, datasets, and generators on acceptance.

2 BACKGROUND AND RELATED WORK

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 value is 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 and non-fluents as O_{SF} and 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].

2.2 RELATED WORK

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 [Toyer et al., 2018] 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.

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. 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).

2.3 SYMNET2.0

Instance-Graph(s): SymNet2.0 converts a given instance into a set of graphs each reffered 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-tuples. 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 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 the 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 are 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 a 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 SINET 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.

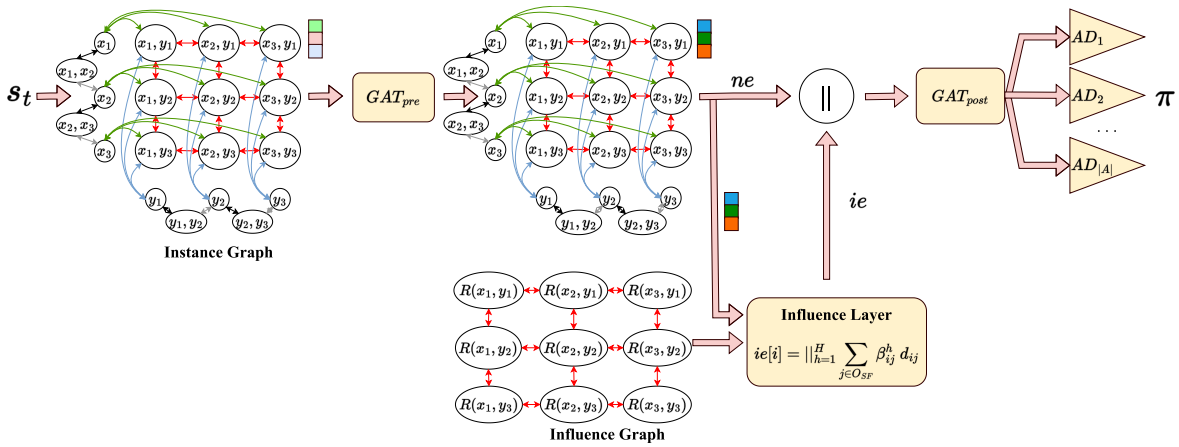

Figure 1: Figure shows the three-step process of SINET for policy prediction. The instance-graph and influence-graph are representative of the Navigation domain, and $R(x, y)$ denotes a predicate.

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 have node features f_1 and f_2, respectively. Let s_2 be another state where the node features of n_1 and n_2 are swapped with each other. 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 s_1 and s_2 as symmetric states.*

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 $\mathcal{P}_{ac_1} - \{o_i\} == \mathcal{P}_{ac_2} - \{o_i'\}$, then, the action score assigned by SymNet2.0 to ac_1 and ac_2 will be the same.*

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

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. □

Theorems 1, 2 and Corollary 1 imply that 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 $robotAt(x, y)$ denote whether the robot is at location (x, y) or not. 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 SYMBOLIC INFLUENCE NETWORK (SINET)

We next present our model Symbolic Influence Network (SINET), which addresses some of the issues faced by SymNet2.0. Intuitively, SymNet2.0 fails to represent certain desirable policies since it's 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 $(\langle u \rangle)$ in I_G (b) There is en 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 DBN.

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 SINET architecture to learn the desirable policies.

3.2.2 SINET 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, 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 SINET, 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: To compute 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,

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

Here, f_i and $\mathcal{N}_i$ denote the features and 1 hop neighbors 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.

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*, an attention-mechanism that

computes influence-embeddings (ie) as, $\forall i, j \in O_{SF}$

$$\beta_{ij}^h = softmax_{O_{SF}}(LRelu(a^T[U_1^h ne[i] || U_1^h ne[j] || U_2^h d_{ij}]))$$
$$\forall i \in O_{SF},\ ie[i] = ||_{h=1}^{H} \sum_{j \in O_{SF}} \beta_{ij}^h\ d_{ij} \quad (4)$$

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

Intuitively, in equation block 4, each state-variable object-tuple node i assigns a weight β_{ij}^h 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_{ij}^h) 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, $ne[i] = GAT_{post}(ne[i] \ || \ ie[i])$ and compute global embedding as $ge = maxpool_{i \in V} ne[i]$. The use of distance features provides nodes in SINET 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. SINET *can represent all policies that SymNet2.0 can represent.*

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

Theorem 4. SINET *can represent certain policies that SymNet2.0 can not represent.*

Proof (Sketch). In our navigation example, the influence distance captures the manhattan distance in the grid. In the influence layer, each node can use different attention heads and assign high attention weights to the nodes with the information for the Goal and four corners of the grid. Hence, the influence-embedding of various nodes can now contain a vector giving the distances from these key nodes. In reference to Theorem 1 and 2, this breaks the artificial symmetry induced among s_1 and s_2 (like in SymNet2.0), leading to the possibility of assigning different scores to ac_1 and ac_2. Thus, SINET can represent certain policies that SymNet2.0 can not. □

4 EXPERIMENTS

We design our experiments for answering three research questions. (i) How well does SINET 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 SINET'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 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 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.

	Model	SRecon	Pizza	DNav	StWall	EAcad	StNav	Mean
r_1	PROST	0.42	0.1	0.94	0.7	0.38	0.01	0.43
r_2	SymNet2.0	0.45	0.17	0.67	0.21	0.91	0	0.4
r_3	SINET-KL	0.39	**0.48**	0.87	0.32	0.92	0.01	0.5
r_4	SINET+KL	**0.56**	0.43	**0.95**	**0.5**	**0.95**	**0.17**	0.59

Table 1: Comparison of SINET with the baselines on 6 LR domains. The best-performing neural model is shown in bold.

	Model	Tam	Traffic	Sys	Skill	Nav	TT	Recon	Elev	Acad	CT	Wild	GoL	Mean
r_1	PROST	0.9	0.94	0.84	0.9	0	0.03	0.7	0.93	0.65	0.35	0.62	0.39	0.6
r_2	SymNet2.0	0.92	**0.86**	0.78	**0.88**	0.3	0.55	0.19	**0.92**	**0.85**	0.81	0.76	**0.81**	0.72
r_3	SINET-KL	**0.94**	**0.86**	**0.85**	0.77	**0.31**	**0.65**	0.21	0.88	0.81	0.78	**0.87**	0.66	0.72
r_4	SINET+KL	0.91	0.83	0.77	0.5	0.29	0.64	**0.33**	0.87	0.76	**0.9**	0.33	0.28	0.62

Table 2: Comparison of SINET with the baselines on 12 IPPC domains. The best-performing neural model is shown in bold.

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 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 100 training, 10 validation, and 40 test instances.[4] See 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. As PROST is a sampling-based solver, it can end up taking different actions for the same state; we remove this ambiguity by keeping a single state-action pair that occurs most frequently. We train for 24 hours for each of the 6 new domains and for 8 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 SINET, GAT_{pre} and GAT_{post} have GAT depth 2 each, and there are 10 attention-heads in the influence-layer for all domains (supplement has further details). We implement two variations: SINET with and without KL divergence, denoted by SINET+KL and SINET-KL, respectively. We compare SINET with SymNet2.0, the existing state-of-the-art model for this task. For fair comparison, we use a 4 depth GAT to match SINET'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 SINET, 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 also observed in other works [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 3 independent runs.

5 RESULTS

Tables 1 and 2 show our results where each $(i,j)^{th}$ entry gives the α value of ith model on the jth 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 SINET+KL and SINET-KL outperform the improved baseline SymNet2.0 on all 6 new LR domains with a margin of +19

[4]We will release code, generators, and instances on acceptance.

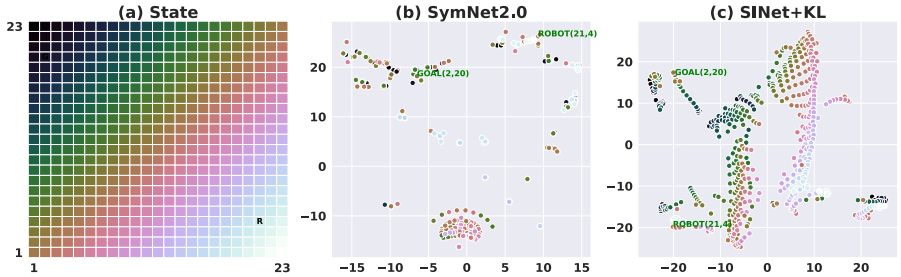

Figure 2: (a) Shows the color-coded locations of a 23×23 instance of the DNAV domain where R and G are the robot and Goal. Fig. (b) and (c) show the 2D t-SNE plot for node-embeddings of the grid locations for SymNet2.0 and SINET.

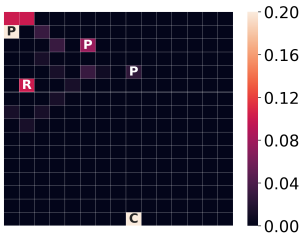

Figure 3: The attention map of the influence-layer in the Pizza domain for the R node.

$\alpha(m)$ points for SINET+KL and $+10$ for SINET-KL. Interestingly, SINET+KL is able to achieve a score of $+17$ on StNav, where both all other methods (PROST, SymNet2.0 and SINET-KL) 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 SINET+KL and SINET-KL get a score of greater than 40 as compared to SymNet2.0's 17. Overall, SINET+KL outperforms all models on such domains.

IPPC Domains: SINET-KL performs at par with SymNet2.0 on the IPPC domains. We see that, in two domains, TT and Recon, SINET+KL shows a significant increase in performance – these two domains require analyzing long-range dependencies for better solutions. However, overall, SINET+KL's performance drops in comparison to SymNet2.0, specifically on Skill, Acad, and GoL domains. We hypothesize that as the training data increases, SINET should perform at par with SymNet2.0. As a preliminary experiment, we successfully validate this hypothesis by training on 200 instances for Acad and Skill domains (see supplement for results).

Use of KL divergence loss: In general, we note that using KL-based loss is critical for domains with long-range dependency, but is not needed for domains that do not require long-range dependency analysis. 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 leads to irrelevant influence embeddings that can lead to overfitting. Our eventual goal is indeed to create *one* planner that can work for all domains. To this end, we vary the regularization parameter associated with the KL loss, and choose the best value on the validation set. We find that this selects SINET+KL and SINET-KL for most LR and IPPC domains, respectively. The overall performance of this unified model is marginally higher than SINET+KL for LR domains, and marginally higher than SymNet2.0 for IPPC (see further details in supplement). This suggests that the SINET architecture along with one hyperparameter tuning, is robust across multiple types of domains.

5.1 INSIGHTS

Visualizing influence-layer: Figure 2 shows the node-embeddings of the locations of a DNav instance of size 23×23 as computed by SINET+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. SINET'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 SINET 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 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 SINET, 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. Directions for future work include going beyond pairwise distance based features, e.g., distance among nodes in a set, for learning more sophisticated policies.

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.

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.

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.