Symbolic Network: Generalized Neural Policies for Relational MDPs

Sankalp Garg [1] **Aniket Bajpai** [1] **Mausam** [1]

Abstract

A Relational Markov Decision Process (RMDP) is a first-order representation to express all instances of a single probabilistic planning domain with possibly unbounded number of objects. Early work in RMDPs outputs generalized (instance-independent) first-order policies or value functions as a means to solve *all* instances of a domain at once. Unfortunately, this line of work met with limited success due to inherent limitations of the representation space used in such policies or value functions. Can neural models provide the missing link by easily representing more complex generalized policies, thus making them effective on all instances of a given domain?

We present SYMNET, the first neural approach for solving RMDPs that are expressed in the probabilistic planning language of RDDL. SYMNET trains a set of shared parameters for an RDDL domain using training instances from that domain. For each instance, SYMNET first converts it to an instance graph and then uses relational neural models to compute node embeddings. It then scores each ground action as a function over the first-order action symbols and node embeddings related to the action. Given a new test instance from the same domain, SYMNET architecture with pre-trained parameters scores each ground action and chooses the best action. This can be accomplished in a single forward pass *without any retraining* on the test instance, thus implicitly representing a neural generalized policy for the whole domain. Our experiments on nine RDDL domains from IPPC demonstrate that SYMNET policies are significantly better than random and sometimes even more effective than training a state-of-the-art deep reactive policy from scratch.

[1]Indian Institute of Technology Delhi. Correspondence to: Sankalp Garg <sankalp2621998@gmail.com>, Aniket Bajapi <quantum.computing96@gmail.com>, Mausam <mausam@cse.iitd.ac.in>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

1. Introduction

A Relational Markov Decision Process (RMDP) (Boutilier et al., 2001) is a first-order, predicate calculus-based representation for expressing instances of a probabilistic planning domain with a possibly unbounded number of objects. An RMDP *domain* has object types, relational state predicate and action symbols that are applied over objects, first order transition templates that specify probabilistic effects associated with action symbols, and a first-order reward structure. A domain *instance* additionally specifies a set of objects and a start state, thus defining a ground MDP with a known start state (Kolobov et al., 2012)). *Relational* planners aim to produce a single *generalized* policy that can yield a ground policy for *all* instances of the domain, with little instance-specific computation. *Domain-independent* planners are representation-specific, but domain-agnostic, making them applicable to all domains expressible in the language. In this paper, we design a domain-independent relational planner.

RMDP planners, in their vision, expect to scale to very large problem sizes by exploiting the first-order structures of a domain – thereby reducing the curse of dimensionality. Traditional RMDP planners attempted to find a generalized *first-order* value function or policy using symbolic dynamic programming (Boutilier et al., 2001), or by approximating them via a function over first-order basis functions (e.g., (Guestrin et al., 2003; Sanner & Boutilier, 2009)). Unfortunately, these methods met with rather limited success, for e.g., no relational planner participated in International Probabilistic Planning Competition (IPPC)[1] after 2006, even though all competition domains were relational. We believe that this lack of success may be due to the inherent limitations in the representation power of a basis function-based representation. Through this work, we wish to revive the research thread on RMDPs and explore if neural models could be effective in representing these first-order functions.

We present **Sym**bolic **Net**Work (SYMNET), the first domain-independent neural relational planner that computes generalized policies for RMDPs that are expressed in the symbolic representation language of RDDL (Sanner, 2010). SYMNET outputs its generalized policy via a neural model whose all parameters are specific to a domain, but tied among all in-

[1]http://www.icaps-conference.org/index.php/Main/Competitions

stances of that domain. So, on a new test instance, the policy can be applied out of the box using pre-trained parameters, i.e., without any retraining on the test instance. SYMNET is domain-independent because it converts an RDDL domain file (and instance files) completely automatically into neural architectures, without any human intervention.

SYMNET architecture uses two key ideas. First, it visualizes each state of each domain instance as a graph, where nodes represent the *object tuples* that are valid arguments to some relational predicate. An edge between two nodes indicates that an action causes predicates over these two nodes to interact in the instance. The values of predicates in a state act as features for corresponding nodes. SYMNET then learns node (and state) embeddings for these graphs using graph neural networks. Second, SYMNET learns a neural network to represent the policy and value function over this graph-structured state. To learn these in an instance-independent way, we recognize that most ground actions are a first-order action symbol applied over some object tuple. SYMNET scores such ground actions as a function over the action symbol and the relevant embeddings of object tuples. After training all model parameters using reinforcement learning over training instances of a domain, SYMNET architecture can be applied on any new (possibly larger) test problem without any further retraining.

We perform experiments on nine RDDL domains from IPPC 2014 (Grzes et al., 2014). Since no planner exists that can run without computation on a given instance, we compare SYMNET to random policies (lower bound) and policies trained from scratch on the test instance. We find that SYMNET obtains hugely better rewards than random, and is quite close to the policies trained from scratch – it even outperforms them in 28% instances. Overall, we believe that our work is a step forward for the difficult problem of domain-independent RMDP planning. We release the code of SYMNET for future research.[2]

2. Background and Related Work

2.1. Probabilistic Planning

Markov Decision Process (MDP): A (ground) finite-horizon MDP (Bellman, 1957; Puterman, 1994) with a known start state is formalized as a tuple $< S, A, T, R, H, s_0, \gamma >$, where S is the set of states, A is the set of actions, T is the transition model $S \times A \times S \to [0, 1]$, R is the reward model $S \times A \times S \to \mathbb{R}$, H is the horizon and s_0 is the start state, and γ, the discount factor. Probabilistic planning problems are often expressed via a factored MDP (Mausam & Kolobov, 2012). It factors a state s into a set of state variables X, i.e., $s = \{x_i\}_{i=1}^{|X|}$. T may also be factored, defined via, e.g., a DBN, dynamic Bayesian network (Sridharan, 1989), which maintains the conditional probability table T^f of $x_i^{'}$ dependent on action a, previous state s, and lower valued $x_j^{'}$s, i.e., $T^f(x_i^{'}|s, a, x_1^{'}, \ldots x_{i-1}^{'})$. The joint probability $T(s, a, s^{'}) = \prod_{x_i^{'} \in s^{'}} T^f(x_i^{'}|s, a, x_1^{'}, \ldots x_{i-1}^{'})$. In practice, these models are compact, and an $x_i^{'}$ depends only on a small number of other state variables.

Relational Markov Decision Process (RMDP): An RMDP $< \mathcal{C}, \mathcal{SP}, \mathcal{A}, O, \mathcal{T}, \mathcal{R}, H, s_0, \gamma >$ is a first-order representation of a factored MDP (Boutilier et al., 2001), expressed via objects, predicates and functions. Here, $\mathcal{C}$ is a set of classes (types), $\mathcal{SP}$ is the set of state predicate symbols. $\mathcal{A}$ is a set of action predicate symbols, O represents a set of domain objects, each associated with single type from $\mathcal{C}$. It is a first order representation because different sets of objects O can construct different ground MDPs.

Each predicate symbol is declared to take as argument a tuple of object types. A predicate symbol (action symbol) applied over a type-consistent tuple of object variables forms a state variable (respectively, ground action). A ground-state s is, thus, a complete assignment of all predicate symbols $\mathcal{SP}$ applied on all type-consistent object tuples from O (also denoted by $\mathcal{SP}_O$). Similarly, the set of all ground actions (A) can be defined as $\mathcal{A}_O$ – all-action symbols applied on all type-consistent object tuples. We also denote the ground state space S by $\mathcal{P}(\mathcal{SP}_O))$, where $\mathcal{P}$ denotes the powerset. Transition and reward models for an RMDP are defined at the schema level through different languages, e.g., PPDDL (Younes et al., 2005) and, our focus, RDDL (Sanner, 2010).

Research in Relational MDPs explores ways to represent and construct first-order (generalized) value functions or policies, which can be used directly on a new test instance. Example representations for these include regression trees (Mausam & Weld, 2003), decision lists (Fern et al., 2006), extensions of algebraic decision diagrams (Joshi & Khardon, 2011), and linear combinations of basis functions (Guestrin et al., 2003; Sanner & Boutilier, 2005). We believe a reason for limited success of RMDP algorithms is the inherent limitation of these representations. We study the use of deep neural models for representing such generalized functions. Moreover, to the best of our knowledge, we are the first to develop relational planners for domains expressed in RDDL.

Relational Dynamic Decision Language (RDDL): RDDL has been the language of choice for the last three IPPCs. It divides $\mathcal{SP}$ into non-fluent ($\mathcal{NF}$) and fluent ($\mathcal{F}$) symbols. Non-fluents are the state variables that do not change with time in a given instance, but may be different across problem instances. Fluents represent state variables that change with time (due to actions or natural dynamics). RDDL splits an RMDP into two separate files, one for the whole domain (that has types, predicates, transitions, and rewards), and the other for the instance (that has objects,

[2] https://github.com/dair-iitd/symnet

non-fluent values, and fluent values for initial state). RDDL uses additive rewards – the total reward is the sum of local rewards collected for satisfying different properties in a state. It factors the transition function via an underlying DBN semantics. There exist algorithms that convert an RDDL instance into a ground DBN (Sanner, 2010).

Running Example: We use a simplified Wildfire domain as our example. It has a grid where each cell may have fuel, causing it to burn. The goal is to have the least damage to the grid by either putting out the fire or cutting out the fuel supply. The DBN for the domain is show in Figure 1.

There are two classes, $\mathcal{C} = \{x_{pos}, y_{pos}\}$: x and y coordinate of the grid cell. Domain has two fluent symbols $\mathcal{F} = \{burning, out\text{-}of\text{-}fuel\}$, representing the current burning state and the fuel state of the cell. Both fluent symbols take a cell (x, y) as its arguments. The non-fluents represent costs and topology, $\mathcal{NF} = \{CostTgtBurn, CostNTgtBurn, Neighbour, Target\}$. The non fluent symbol $Neighbour$ takes four arguments (x, y, x', y'), since it defines the topology of the grid. $Target$ has arguments (x, y). $\mathcal{A} = \{put\text{-}out, cut\text{-}out, finisher\}$. First two action symbols take arguments (x, y) – they put out fire and cut out fuel supply at a cell. There is one global action $finisher$, which puts out fire in all the cells simultaneously. Reward (negative) in each time step adds $CostTgtBurn$ for each target cell that is burning and $CostNTgtBurn$ for each non-target cell that is burning. In a problem instance, say there are three objects $O = \{x1, x2, y1\}$. This implies a problem with two cells $(x1, y1)$, and $(x2, y1)$. Say the target cell is $(x1, y1)$ and that these are connected, i.e., $Neighbour(x1, y1, x2, y1) = 1$.

RDDL vs PPDDL: PPDDL and RDDL have significant differences in their modeling choices. PPDDL uses correlated effects, whereas RDDL naturally models parallel effects. Thus, RDDL handles no-op actions with underlying natural dynamics better. RDDL rewards are state-dependent and sum over all objects satisfying a property, whereas PPDDL has both goals and rewards, but its rewards are associated with action transitions and do not aggregate over objects.

2.2. Reinforcement Learning

Reinforcement Learning (RL) refers to planning problems without known transition and rewards, necessitating learning from experience. State of the art approaches for RL are neural, which approximate policy and value functions through deep neural models. We use the Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) as our underlying RL algorithm. A3C uses two neural networks 1) θ_π to represent the policy (mapping from a state to distribution over actions) and 2) θ_V to estimate the state value (long term discounted reward starting in a state). Policy parameters are optimized to prefer an action that increases a state's advantage function – difference between its value when taking that action and its overall value. Value parameters optimize the MSE loss between observed and predicted long-term rewards.

2.3. Graph Neural Networks

Graph Neural Networks input a graph and learn latent space embeddings for each node, based on the its individual features and the local connectivity structure. Examples include Graph Convolution Networks (GCN) (Kipf & Welling, 2017), and Graph Attention Networks (GAT) (Velickovic et al., 2017). We use GAT, which computes a node embedding by using a weighted attention for each of neighbouring nodes. Specifically output node embedding $\overline{v}_i{}' = \sigma(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N_i}\alpha_{ij}^k \mathbf{W}^k\overline{v}_j)$, where $\overline{v}_i$ is the input feature of node v_i, N_i is its neighbours, $\mathbf{W}^k$ is a trainable weight matrix, k is the multi-head hyperparameter and $\alpha_{ij}^k = softmax_j(f(\mathbf{W}^k\overline{v}_i, \mathbf{W}^k\overline{v}_j))$ is the normalized self attention coefficient for any non-linear function (f), which in our implementation is LeakyReLU (Xu et al., 2015).

2.4. Transfer Learning for Probabilistic Planning

There having been several classical (Taylor & Stone, 2009; Sorg & Singh, 2009; Atkeson & Schaal, 1997) and neural (Parisotto et al., 2015; Matiisen et al., 2017; Garnelo et al., 2016; Higgins et al., 2017) approaches for transfer learning in RL. Recent work studies transfer learning for symbolic planning problems, e.g., Groshev et al. (2018) for deterministic planning problems. ASNets, Action-Schema Networks (Toyer et al., 2018; Shen et al., 2019), tackle a problem similar to ours but for goal-oriented subset of PPDDL. While an RDDL instance can be converted automatically into propositional PPDDL, an RDDL domain cannot always be converted into relational PPDDL – hence we cannot directly compare against ASNets. Issakkimuthu et al. (2018) devise a neural framework to learn a policy for (ground) RDDL MDPs from scratch. Their constraint on non-transferability is due to the fixed size of fully connected layers in the neural network. TORPIDO achieves transfer across RDDL problem instances of the same domain (Bajpai et al., 2018); it can only transfer over *equi-sized* problems due to its fixed size action decoder.

Our previous work TRAPSNET is closest to SYMNET, as it can transfer to different-sized instances of an RMDP (Garg et al., 2019). It constructs a graph, which uses single object as nodes, and non-fluent based edge. It encodes each node in embedding space and computes the score for a ground action based on the applied action template, and object embedding. However, TRAPSNET makes the restrictive assumptions that the domains have exactly one binary non-fluent, and all the rest are unary fluents or non-fluents, and that each action symbol is parameterized by *exactly* one object. These assumptions do not hold in several RDDL domains.

3. Problem Formulation

Given an RMDP domain $D =< \mathcal{C}, \mathcal{SP}, \mathcal{A}, \mathcal{T}, \mathcal{R}, H, \gamma >$ expressed in RDDL, we wish to learn a generalized policy π^D, which can be applied to all instances of D and maximizes the discounted sum of expected rewards over a finite horizon H. Given a test problem instance $I_t =< O, s_0 >$ from D, this generalized policy can yield an instance-specific policy $\pi^D(I_t) : \mathcal{P}(\mathcal{SP}_O) \rightarrow \mathcal{A}_O$, without any training on I_t. The RMDP learning problem can be seen in terms of multi-task learning over several problem instances in D: given N randomly selected problem instances I_1, I_2, ..., I_N (possibly of different sizes) from D, we wish to learn the weights ϕ of a neural network, such that $\pi^D(I_i; \phi)$ is a good (high-reward) policy for problem instance I_i. A good generalized policy is one which, without training, achieves high reward values on the new instance I_t.

4. The SymNet Framework

We now present SymNet's architecture for training a generalized policy for a given RMDP domain. We follow existing research to hypothesize that for any instance of a domain, we can learn a representation of the current state in a latent space and then output a policy in latent space, which is decoded into a ground action. To achieve this, SymNet uses three modules: (1) problem representation, which constructs an instance graph for every problem instance, (2) representation learning, which learns embeddings for every node in the instance graph, and for the state, and (3) policy decoder, which computes a value for every ground action, outputting a mixed policy for a given state. All parameters of representation learning and policy learning modules are shared across all instances of a domain. SymNet's full architecture is shown in Figure 2.

4.1. Problem Representation

We follow TraPSNet, in that we continue the general idea of converting an instance into an instance graph and then learning a graph encoder to handle different-sized domains. However, the main challenge for a general RMDP, one that does not satisfy the restricted assumptions of TraPSNet, is in defining a coherent graph structure for an instance. The first key question is what should be a node in the instance graph. TraPSNet's approach was to use a single object as nodes, as all fluents (and actions) in its domains took single objects as arguments. This may not work for a general RMDP since it's fluents and actions may take several objects as arguments. Secondly, how should edges be defined. Edges represent the interaction between nodes. TraPSNet defined them based on the one binary non-fluent in its domain. A general RMDP may not have any non-fluent symbol or may have many (possibly higher-order) non-fluents.

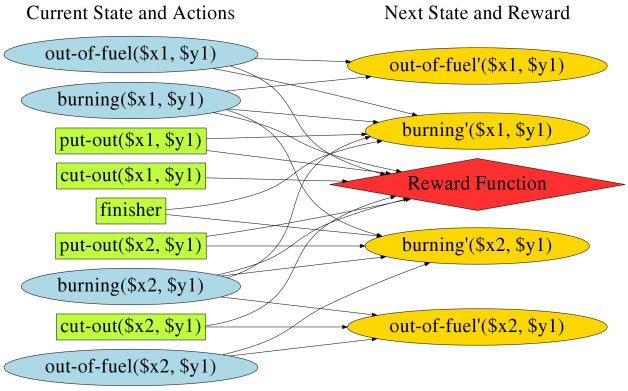

Figure 1. DBN for a modified wildfire problem.

Last but not least, the real domain-independence for SymNet can be achieved only when it parses an RDDL domain file without any human intervention. This leads to a novel challenge of reconciling multiple different ways in RDDL to express the same domain. In our running example, connectivity structure between cells may be defined using non-fluents $y\text{-}neighbour(y, y')$, $x\text{-}neighbour(x, x')$, or using a quarternary non-fluent $neighbour(x, y, x, y')$. Since both these representations represent the same problem, an ideal desideratum is that the graph construction algorithm leads to the same instance graph in both cases. But, this is a challenge since the corresponding RDDL domains may look very different. While, in general, this problem seems too hard to solve, since it is trying to judge logical equivalence of two domains, SymNet attempts to achieve the same instance graphs in case the equivalence is within non-fluents.

To solve these problems, we make the observation that dynamics of an RDDL instance ultimately compile to a ground DBN with nodes as state variables (fluent symbols applied on object tuples) and actions (action symbols applied on object tuples).[3] DBN exposes a connectivity structure that determines which state variables and actions directly affect another state variable. It additionally has conditional probability tables (CPTs) for each transition. Figure 1 shows an example of a DBN for our running example instance. Here, left column is for current time step, and right for the next one. The edges represent which state and action variables affect the next state-variable. We note that the ground DBN does not expose non-fluents since its values are fixed, and their dependence can be compiled directly into CPTs.

SymNet converts a ground DBN to an instance graph. It constructs a node for every unique *object tuple* that appears as an argument in any state variable in the DBN. Moreover, two nodes are connected if the state variables associated with two nodes influence each other in the DBN through

[3] done automatically using code from https://github.com/ssanner/rddlsim

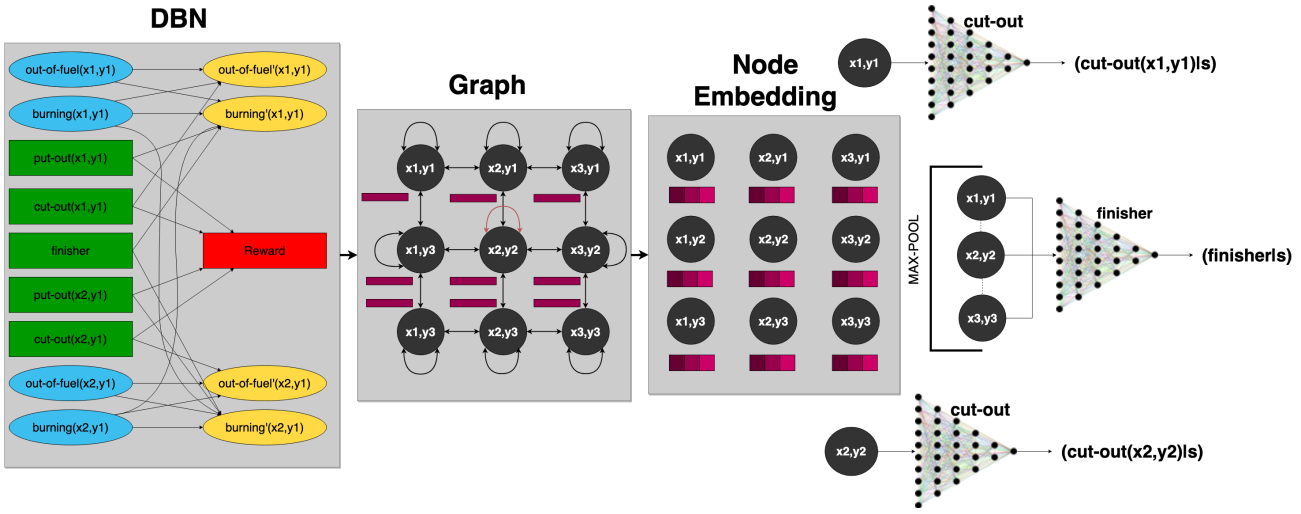

Figure 2. Policy network for SYMNET demonstrated on 2×1 wildfire domain. Fully Connected Network is used in Action Decoder.

some action. This satisfies all our challenges. First, it goes beyond an object as a node, but only defines those nodes that are likely important in the instance. Second, it defines a clear semantics of edges, while maintaining its intuition of "directly influences." Finally, it can handles some variety of non-fluent representations for the same domain. Since the DBN does not even expose non-fluent state variables, and compiles them away, same instancs encoded with different non-fluent representations often yield yield same ground DBNs and thus the same instance graphs.

Construction of Instance Graph: We now formally describe the conversion of a DBN into a directed instance graph, $G = (V, E)$, where V is the set of vertices and E is the set of edges. G is composed of $\mathcal{K} = |\mathcal{A}| + 1$ disjoint subgraphs $G_j = (V_j, E_j)$. Intuitively, each graph G_j has information about influence of each individual action symbol $\mathbf{a}_j \in \mathcal{A}$. $G_{\mathcal{K}}$ represents the influence of the full set $\mathcal{A}$, and also the natural dynamics. In our example $\mathcal{K} = 4$ since we have three action symbols: put-out, cut-out and finisher.

To describe the formal process, we define three analogous sets: O_f, O_{nf} and O_a. O_f represents the set of all object tuples that act as a valid argument for any fluent symbol. O_{nf} and O_a are analogous sets for non-fluent and action symbols. In our running example, $O_f = \{(x1, y1), (x2, y1)\}$, $O_{nf} = \{(x1, y1), (x2, y1), (x1, y1, x2, y1), (x2, y1, x1, y1)\}$, and $O_a = \{(x1, y1), (x2, y1)\}$. Nodes in the instance graph associate with object tuples. We use o_v to denote the object tuple associated with node v. SYMNET converts a DBN into an instance graph as follows:

1. The distinct object tuples in fluents form the nodes of the graph, i.e. $V_j = \{v | o_v \in O_f\}, \forall j$. For the example, each $V_j =$ different copies of $\{(x1, y1), (x2, y1)\}$.
2. We add an edge between two nodes in G_j if some state variables corresponding to them are connected in the DBN through $\mathbf{a}_j$. Formally, $E_j(u, v) = 1$, if $\exists f, g \in F, \exists o_a \in O_a, j \in \{1, \ldots, |\mathcal{A}|\}$ s.t. the transition dynamics (T^f) for state variable $g'(o_v)$ and action $\mathbf{a}_j(o_a)$ depend on state variable $f(o_u)$ or $f'(o_u)$.For the running example, there is no edge between $(x1, y1)$ and $(x2, y1)$ since cut-out, put-out or finisher's effects on one cell do not depend on any other cell.
3. We add an edge between two nodes in $G_{\mathcal{K}}$ if some state variables corresponding to them are connected in the DBN (possibly through natural dynamics). I.e., $E_{\mathcal{K}}(u, v) = 1$, if $\exists f, g \in F$ s.t. there is an edge from $f(o_u)$ (or $f'(o_u)$) and $g'(o_v)$ in the DBN. For the example, $E_4((x1, y1), (x2, y1)) = 1$ as there is an edge between $burning(x1, y1)$ and $burning'(x2, y1)$ since fire propagates to neighboring cells through natural dyanamics. Similarly, $E_4((x2, y1), (x1, y1)) = 1$.
4. As every node influences itself, self loops are added on each node. $E(v, v) = 1, \forall v \in V$.

For each node $v \in V$, we additionally construct a feature vector ($h(v)$) which consists of fluent feature vector ($h^f(v)$) and non-fluent feature vector ($h^{nf}(v)$), such that $h = concat(h^f, h^{nf})$. The feature vector for all nodes for the same object tuple is the same. The feature vector is constructed as follows:

1. The fluent features for each node is obtained from the state of the problem instance. The values of state variables corresponding to a node are added as feature to that node. Whenever a fluent symbol cannot take a node as an argument, we add zero as the feature for it. Formally, $h^f(v)_i = g_i(o_v)$ if $g_i \in \mathcal{F}, v \in V$

and o_v is an argument of g_i, otherwise, $h^f(v)_i = 0$, $\forall i = 1 \dots |\mathcal{F}|$. For the running example, we have two state-fluents. Hence, $h^f((x1, y1)) = [burning(x1, y1), out\text{-}of\text{-}fuel(x1, y1)]$.

2. The non-fluent feature vector for each node is obtained from the RDDL file. The values of non-fluents defined on the node, and additionally any unary non-fluents where the argument intersects the node are added as the features for the node. The default value is obtained from the domain file while the specific value (if available) is obtained from the instance file. Formally, $h^{nf}(v)_i = g_i(o_{nf})$ if $g_i \in \mathcal{NF}, v \in V$, $o_{nf} \in O_{nf}, ((o_v = o_{nf}) \vee (|o_{nf}| = 1 \wedge o_{nf} \subset o_v))$, otherwise, $h^f(v)_i = 0, \forall i = 1 \dots |\mathcal{NF}|$. In our example, $h^{nf}((x1, y1)) = [target(x1, y1)]$.

We note that the size of feature vector on each node depends on the domain, but is independent of the number of objects in the instance – there are a constant number of feature values per state predicate symbol. This allows variable-sized instances of the same domain to use the same representation.

4.2. Representation Learning

SYMNET runs GAT on the instance graph to obtain node embeddings $\overline{v}$ for each node $v \in V$, It then constructs tuple embedding for each object tuple by concatenating node embeddings of all associated nodes. Formally, let $O_V = \{o_v | v \in V\}$. For $o \in O_V$, the tuple embedding $\overline{o} = \text{concat}(\overline{v})$, over all v s.t. $o_v = o$. SYMNET also computes a state embedding $\overline{s}$ by taking a dimension-wise max over all tuple embeddings, i.e., $\overline{s} = MaxPool_{o \in O_V}(\overline{o})$.

4.3. Policy Decoder

SYMNET maps latent representations $\overline{o}$ and $\overline{s}$ into a state value $V(s)$ (long-term expected discounted reward starting in state s) and mixed policy $\pi(s)$ (probability distribution over all ground actions). This is done using a value decoder and a policy decoder, respectively.

There are several challenges in designing a (generalized) policy decoder. First, the action symbols may take multiple objects as arguments. Second, and more importantly, action symbols may even take those object tuples as arguments that do not correspond to any node in the instance graph. This will happen if an object tuple (in O_a) is not an argument to any fluent symbol, i.e., $\exists o_a$ s.t. $o_a \in O_a \wedge o_a \notin O_f$. We note that adding these object tuples as nodes in the instance graph may not work, since we will not have any natural features for those nodes.

In response, we design a novel framework for policy and value decoders. The decoders consist of fully connected layers, the input to which are a subset of the tuple embeddings $\overline{o}$. SYMNET uses the following rules to construct decoders:

1. The number of decoders is constant for a given domain and is equal to the number of distinct action symbols ($|\mathcal{A}|$). For the running example, three different decoders for each policy and value decoding are constructed, namely $cut\text{-}out, put\text{-}out$ and $finisher$.
2. The input to a decoder is the state embedding $\overline{s}$ concatenated with embeddings of object tuples corresponding to the state variables affected by the action in the DBN. In running example, $put\text{-}out(x1, y1)$ action takes only the tuple embedding of $(x1, y1)$ as input. However, the number of state-variables being affected by a ground action might vary across instances of the same domain. For example, the $finisher$ action affects all cells. To alleviate this, we use size-independent max pool aggregation over the embeddings of all affected tuple embeddings to create a fixed-sized input.
3. Decoder parameters are specific to action symbols and not to ground actions. In running example, $put\text{-}out(x1, y1)$ will be scored using embedding of $(x1, y1)$; similarly, for $(x2, y1)$. But, both scorings will use a single parameter set specific to $put\text{-}out$.
4. The policy decoder computes scores of all ground actions, which are normalized using softmax to output the final policy in a state. For I_t, the highest probability action is selected as the final action.
5. All value outputs are summed to give the final value for that state. This modeling choice reflects the additive reward aspect of many RDDL domains.

4.4. Learning

While construction of SYMNET architecture is heavily dependent on the RDDL domain and instance files, actual training is done via model-free reinforcement learning approach of A3C (Mnih et al., 2016). RL learns from interactions with environment – SYMNET simulates the environment using RDDL-specified dynamics. Use of model-based planning algorithms for this purpose is left as future work. We formulate training of SYMNET as a multi-task learning problem (see Section 3), so that it generalizes well and does not overfit on any one problem instance. The parameters for the state encoder, policy decoder, and value decoder are learned using updates similar to that in A3C. SYMNET's loss function for the policy and value network is the same as that in the A3C paper (summed over the multi-task problem instances).

As constructed, SYMNET's number of parameters is independent of the size of the problem instance. Hence, the same network can be used for problem instances of any size. After the learning is completed, the network represents a generalized policy (or value), since it can be directly used on a new problem instance to compute the policy in a single forward pass.

Table 1. $\alpha_{symnet}(0)$ values of SYMNET. Bold values represent over 90% the score of max performance.

	Instance	5	6	7	8	9	10
Domain	AA	**0.93 ± 0.01**	**0.94 ± 0.01**	**0.94 ± 0.01**	**0.92 ± 0.02**	**0.95 ± 0.03**	**0.91 ± 0.05**
	CT	0.87 ± 0.16	0.78 ± 0.14	**1.00 ± 0.07**	**0.98 ± 0.13**	**0.99 ± 0.04**	**1.00 ± 0.05**
	GOL	**0.96 ± 0.06**	**1.00 ± 0.05**	0.65 ± 0.05	0.83 ± 0.03	**0.95 ± 0.04**	0.64 ± 0.08
	Nav	**0.99 ± 0.01**	**1.00 ± 0.01**	**0.99 ± 0.01**	**1.00 ± 0.01**	**1.00 ± 0.02**	**1.00 ± 0.02**
	ST	**0.91 ± 0.05**	0.84 ± 0.02	0.86 ± 0.05	0.85 ± 0.05	0.81 ± 0.02	0.89 ± 0.03
	Sys	**0.96 ± 0.03**	**0.98 ± 0.02**	**0.98 ± 0.02**	**0.97 ± 0.02**	**0.99 ± 0.01**	**0.96 ± 0.03**
	Tam	**0.92 ± 0.07**	**1.00 ± 0.12**	**0.98 ± 0.06**	**1.00 ± 0.12**	**1.00 ± 0.12**	**0.95 ± 0.06**
	Tra	0.85 ± 0.18	**0.93 ± 0.06**	0.88 ± 0.21	0.74 ± 0.17	**0.94 ± 0.12**	0.87 ± 0.13
	Wild	**0.99 ± 0.01**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.00**	**1.00 ± 0.01**	**1.00 ± 0.01**

5. Experiments

Our goal is to estimate the effectiveness of SYMNET out-of-the-box policy for a new problem in a domain. Unfortunately, there are no available transfer algorithms for general RDDL RMDPs. So, we first compare it against a random policy, because that is the best we can do currently with no time to train. To further understand the overall quality of the generalized policy, we also compare it against several neural models that train from scratch on the test instance. We also compare it against state-of-the-art online planner PROST (Keller & Eyerich, 2012).

5.1. Domains and Experimental Setting

We show all our results on nine RDDL domains used IPPC 2014: Academic Advising (AA), Crossing Traffic (CT), Game of Life (GOL), Navigation (NAV), Skill Teaching (ST), Sysadmin (Sys), Tamarisk (Tam), Traffic (Tra), and Wildfire (Wild). We describe the domains, and the number of state fluents, state non-fluents, and action fluents in the supplementary material. The RL agent is trained to learn the generalized policy on smaller sized instances. We use IPPC problem instances 1, 2, and 3 of each domain for the multi-task training of SYMNET network. In the spirit of domain-independent planning, we use the *same* hyperparameters for each domain. The embedding module for GAT uses a neighborhood of 1 and an output feature size of 6. We then use a fully connected layer of output 20 dimensions to get an embedding from each of the tuple embedding outputs by GAT. All layers use a leaky ReLU activation and a learning rate of 10^{-3}. We train the network using RMSProp (Ruder, 2016) on a single Nvidia K40 GPU. SYMNET is trained for each domain for twelve hours (4 hours for each instance).

5.2. Comparison Algorithms and Metrics

As there does not exist any previous method for learning over Relational RDDL MDPs, we can only compare against a random policy. However, this experiment can only show the difference from a random policy, but cannot evaluate the overall goodness of the generalized policy. For that, we compare against several (potentially upper bound) policies that are not directly comparable to SYMNET in their experimental settings. For our first such experiment, we use TORPIDO as the state-of-the-art deep reactive policy. Note that we do not use their transfer method, but train the network from scratch on the problem instance. This is because it can only transfer across equi-sized instances. Still, it is an upper bound as TORPIDO trained on the test instance is compared against SYMNET trained on other smaller instances, but not the test instance. Similarly, we also compare against SYMNET architecture itself, trained from scratch on the test instance (named SYMNET-s). The main difference between TORPIDO and SYMNET architectures is that TORPIDO has a much higher capacity since it models each ground action explicitly. On three domains where TRAPSNET is applicable, we also compare against TRAPSNET policies out of the box. Finally, we also compare against the state-of-the-art *online* planner, PROST.

After training algorithm alg for t hours, we simulate its output policy 200 times (for H steps each) from the start state. We average the discounted rewards to estimate the expected long term discounted reward of that policy, denoted by $V_{alg}(t)$. To be able to compare across domains and problems and reward ranges, we report a normalized metric $\alpha_{alg}(t) = \frac{V_{alg}(t)-V_{min}}{V_{max}-V_{min}}$ where V_{min} and V_{max} are the minimum, and the maximum expected discounted rewards obtained at any time by any of the four comparison algorithms on a given instance. This number lies between 0 and 1, with 1 being the best-found reward, and 0 being the random policy's reward. All algorithms are trained independently 5 times and the average result is reported. During training from scratch, all networks start with a random policy and hence have their $\alpha(0)$ values as 0. However, that is not true for SYMNET as it is pre-trained on the domain. To compare against other training approaches directly, we compute $\beta_{alg}(t) = \frac{\alpha_{symnet}(0)}{\alpha_{alg}(t)}$. A value higher than 1 suggests that SYMNET out-of-the-box outperformed alg trained for t hours, and less than 1 implies SYMNET performed worse.

Table 2. Comparison of SYMNET against SYMNET-s (SYM) architecture trained from scratch and TORPIDO (TOR) architecture trained from scratch. We compare out-of-the-box SYMNET to others after 12 hours of training. INF is used when SYM or TOR achieved minimum possible reward and hence SYMNET was infinitely better.

Domain	SYM	TOR	Domain	SYM	TOR	Domain	SYM	TOR	Domain	SYM	TOR
AA 5	1.09	0.99	GOL 5	**1.35**	**1.49**	ST 5	1.11	0.94	Tam 5	**INF**	**2.33**
AA 6	1.78	0.95	GOL 6	**1.57**	**1.69**	ST 6	1.21	0.90	Tam 6	**27.71**	**8.13**
AA 7	1.21	0.98	GOL 7	1.08	0.76	ST 7	1.10	0.87	Tam 7	**17.81**	**4.83**
AA 8	1.31	0.97	GOL 8	2.22	0.87	ST 8	1.14	0.90	Tam 8	**2.74**	**15.56**
AA 9	1.39	0.95	GOL 9	**1.86**	**1.31**	ST 9	1.13	0.81	Tam 9	**24.94**	**13.07**
AA 10	1.32	0.93	GOL 10	1.25	0.68	ST 10	1.30	0.95	Tam 10	**2.35**	**7.99**
CT 5	**1.34**	**1.39**	Nav 5	**10.84**	**INF**	Sys 5	**1.03**	**2.89**	Tra 5	1.78	0.86
CT 6	**INF**	**1.56**	Nav 6	**INF**	**INF**	Sys 6	**1.33**	**1.20**	Tra 6	**1.56**	**1.39**
CT 7	**1.13**	**1.12**	Nav 7	**INF**	**INF**	Sys 7	**1.56**	**2.45**	Tra 7	**3.28**	**1.13**
CT 8	**1.55**	**1.23**	Nav 8	**INF**	**INF**	Sys 8	**1.46**	**1.60**	Tra 8	1.13	0.81
CT 9	**1.35**	**1.16**	Nav 9	**INF**	**INF**	Sys 9	**1.38**	**1.17**	Tra 9	**2.50**	**1.08**
CT 10	**1.22**	**4.99**	Nav 10	**INF**	**INF**	Sys 10	**1.18**	**1.50**	Tra 10	**1.53**	**1.86**
Wild 5	**1.03**	**1.13**	Wild 7	**1.03**	**1.13**	Wild 9	**1.01**	**13.14**			
Wild 6	**1.01**	**1.01**	Wild 8	**1.00**	**1.09**	Wild 10	**34.80**	**11.19**			

Table 3. Comparison of TRAPSNET with SYMNET on three domains as published in (Garg et al., 2019). Label: AA - Academic Advising, GOL - Game Of Life, Sys - Sysadmin

	Instance	5	6	7	8	9	10
Domain	AA	**1.12**	**1.17**	**1.12**	**1.27**	**1.26**	**1.40**
	GOL	0.96	**1.04**	0.69	**1.00**	0.97	**1.50**
	Sys	**1.01**	**1.55**	**1.33**	**1.39**	**1.21**	**1.17**

5.3. Results

Comparison against Random Policy: We report the values of $\alpha_{symnet}(0)$ in Table 1. Since the random policy is 0, we notice that on all six problem instances from the nine domains, SYMNET performs enormously better than random. We highlight the instances where our method achieves over 90% of the max reward obtained by any algorithm for that instance. We see that SYMNET with no training achieves over 90% the max reward on 40 instances and over 80% in 50 out of 54 instances. We also show that our method performs the best out-of-the-box in 28 instances. This is our main result, and it highlights that SYMNET takes a major leap towards the goal of computing generalized policies for the whole RMDP domain, and can work on a new instance out of the box.

Comparison against Training from Scratch: We now compare SYMNET against the expected discounted rewards obtained by TORPIDO and SYMNET-s, when they are trained from scratch for 12 hours on the test problem. We note that these numbers are not directly comparable, since in one case, the model has been trained on other instances of the domain, but not trained on the test problem at all, and in the other case the models are trained from scratch on the test. That said, this comparison is likely a good indicator of the absolute performance of SYMNET.

Table 2 reports the values for $\beta_{torpido}(12)$ and $\beta_{symnet-s}(12)$. We notice that, surprisingly, SYMNET policy with no training is better than both methods on several instances. Against SYMNET trained from scratch, it is better on all instances, although its edge over TORPIDO is limited to 37 out of 54. We hypothesize that this excellent performance is due to the multi-task learning aspect of SYMNET, where it is able to reach some generalized policy of a domain that is not found on the specific instance even after training for 12 hours.

In 17 out of 54 instances, SYMNET lags behind TORPIDO , which is not surprising, since TORPIDO has much higher capacity, as discussed earlier. We also notice that the performance of TORPIDO is no better than random for Navigation. We attribute this to the sparse and late reward obtained in large instances of this domain, which makes it difficult for TORPIDO to learn a good policy. Because of the late rewards, TORPIDO is not able to reach the goal state at all in 12 hours of training, and hence is not able to improve on the random policy. SYMNET trains well on small instances where path to goal is short and generalizes well. In GOL, SYMNET performs worse, because the nature of policy changes significantly in large instances (e.g. requiring new patterns to survive) which cannot be learned in smaller instances at all.

Comparison against TRAPSNET: While TRAPSNET is not applicable in many RMDPs, still, we can compare it with

Table 4. Comparison of PROST with SYMNET. INF is used when PROST returned a policy equal to or worse than a random policy.

	Domain	AA	CT	GOL	Nav	ST	Sys	Tam	Tra	Wild
Instance	5	**2.13**	0.76	0.48	**1.16**	0.95	**1.13**	0.61	0.60	**1.39**
	6	**2.14**	0.44	0.57	**1.87**	0.86	**1.24**	0.96	0.65	**INF**
	7	**2.18**	0.62	0.33	**6.42**	0.86	**1.13**	0.70	0.61	**INF**
	8	**1.79**	0.37	0.39	**45.46**	0.90	**1.50**	0.79	0.51	**INF**
	9	**1.46**	0.74	0.44	**101.23**	0.78	**1.21**	0.83	0.75	**INF**
	10	**1.46**	0.37	0.30	**INF**	0.93	**1.42**	0.84	0.64	**1.49**

SYMNET on some domains. We compare these on three domains that follow the unary fluents and binary non-fluents constraint: Academic Advising, Game of Life, and SysAdmin. We report $\beta_{trapsnet}(0)$ in Table 3. It shows that SYMNET outperforms TRAPSNET on 15 out of 18 instances, is comparable on 2 instances and worse on 1 instance. We attribute the success of SYMNET over TRAPSNET to the action-symbol specific graphs (G_j), which likely help learn better action dependencies in the embeddings.

Comparison against ASNets: Even after significant efforts, we were not able to compare against ASNets, which solves a similar problem for PPDDL domains. Converting an RDDL domain to PPDDL enumerates all the ground state-variables and loses the RMDP structure. This leads to different domain files for different instances for the same problem domain, due to which ASNets is unable to train. We also tried writing a domain file manually for a few domains, but were not successful due to the unavailability of floating non-fluent values, and due to non-additive reward structure in PPDDL.

Comparison against PROST: Finally, we compare against PROST. PROST is a state-of-the-art *online* planner, i.e., it performs interleaved planning and execution, as it builds a new search tree before taking every action, based on the specific state reached. On the other hand, SYMNET outputs an offline policy, which does not need much computation for deciding the next action. Offline and online policies are two very different settings, and these results are *not directly comparable*. Nonetheless, we report $\frac{V_{symnet}(0)-V_{min}}{V_{prost}-V_{min}}$. The code of PROST is obtained from the official repository and we use its default settings for this comparison.[4]

We compare our policy with PROST on all 9 domains, shown in Table 4. We see that on four domains SYMNET achieves a much better performance than PROST. This is rather surprising to us that even after substantial lookahead from the current state, PROST is still not able to compute a good policy. For example, in both Navigation and Wildfire, the rewards are sparse and distant, and PROST is often unable to reach the goal in its planning horizon. In other five domains, PROST is substantially better than SYMNET. This suggests that SYMNET policies are not close to optimal, and further research is needed for making them even stronger. This also points to the future possibility of applying a combination of SYMNET and PROST for the offline setting, not unlike the use of Monte-Carlo Tree Search with deep neural networks in AlphaGo (Silver et al., 2016).

Overall, we find that SYMNET's generalized policies out-of-the-box are enormously better than random, and can frequently beat other deep neural models trained from scratch on the test instance. However, comparison with PROST suggests that SYMNET policies are not close to optimal and further research is needed to make them even better.

6. Conclusion and Future Work

We present the first neural-method for obtaining a generalized policy for Relational MDPs represented in RDDL. Our method, named SYMNET, converts an RDDL problem instance into an instance graph, on which a graph neural network computes state embeddings and embeddings for important object tuples. These are then decoded into scores for each ground action. All parameters are tied and size-invariant such that the same model can work on problems of varying sizes. In our experiments, we train SYMNET on small problems of a domain and test them on larger problems to find that they out-of-the-box perform hugely better than random. Even when compared against training deep reactive policies from scratch, SYMNET without training perform better or at par in over half the problem instances.

Our work is an attempt to revive the thread on Relational MDPs and the attractive vision of generalized policies for a domain. However, ours is only one of the first steps. Further investigation is needed to assess how far are SYMNET's generalized policies from optimal. We strongly believe that there may be even better architectures that could learn near-optimal generalized policies, and the need for retraining or interleaving planning and execution could be rendered unnecessary. We release all our software for use by the research community at https://github.com/dair-iitd/symnet.

[4] https://github.com/prost-planner/prost

Acknowledgements

We would like to thank Scott Sanner for an extremely insightful discussion on DBNs, which enabled us to work out a solution to convert an RMDP into a graph using DBNs. We also thank Vishal Sharma, Gobind Singh, Parag Singla and the anonymous reviewers for their comments on various drafts of the paper. This work is supported by grants from Google, Bloomberg, IBM and 1MG, Jai Gupta Chair Fellowship, and a Visvesvaraya faculty award by Govt. of India. We thank Microsoft Azure sponsorships, and the IIT Delhi HPC facility for computational resources.

References

Atkeson, C. G. and Schaal, S. Robot learning from demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997*, pp. 12–20, 1997.

Bajpai, A., Garg, S., and Mausam. Transfer of deep reactive policies for mdp planning. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10965–10975. Curran Associates, Inc., 2018.

Bellman, R. A Markovian Decision Process. *Indiana University Mathematics Journal*, 1957.

Boutilier, C., Reiter, R., and Price, B. Symbolic dynamic programming for first-order mdps. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pp. 690–700, 2001.

Fern, A., Yoon, S. W., and Givan, R. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *J. Artif. Intell. Res.*, 25:75–118, 2006.

Garg, S., Bajpai, A., and Mausam. Size independent neural transfer for rddl planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 631–636, 2019.

Garnelo, M., Arulkumaran, K., and Shanahan, M. Towards deep symbolic reinforcement learning. *CoRR*, abs/1609.05518, 2016. URL http://arxiv.org/abs/1609.05518.

Groshev, E., Tamar, A., Goldstein, M., Srivastava, S., and Abbeel, P. Learning generalized reactive policies using deep neural networks. In *ICAPS*, 2018.

Grzes, M., Hoey, J., and Sanner, S. International Probabilistic Planning Competition (IPPC) 2014. In *ICAPS*, 2014. URL https://cs.uwaterloo.ca/~mgrzes/IPPC_2014/.

Guestrin, C., Koller, D., Gearhart, C., and Kanodia, N. Generalizing plans to new environments in relational mdps. In *IJCAI*, pp. 1003–1010, 2003.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 1480–1490, 2017.

Issakkimuthu, M., Fern, A., and Tadepalli, P. Training deep reactive policies for probabilistic planning problems. In *ICAPS*, 2018.

Joshi, S. and Khardon, R. Probabilistic relational planning with first order decision diagrams. *Journal of Artificial Intelligence Research*, 41:231–266, 2011.

Keller, T. and Eyerich, P. PROST: probabilistic planning based on UCT. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, 2012. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kolobov, A., Mausam, and Weld, D. S. A theory of goal-oriented mdps with dead ends. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pp. 438–447, 2012.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. Teacher-student curriculum learning. *CoRR*, abs/1707.00183, 2017. URL http://arxiv.org/abs/1707.00183.

Mausam and Kolobov, A. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool Publishers, 2012.

Mausam and Weld, D. S. Solving relational MDPs with first-order machine learning. In *ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, 2003.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 1928–1937, 2016.

Parisotto, E., Ba, L. J., and Salakhutdinov, R. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342, 2015. URL http://arxiv.org/abs/1511.06342.

Puterman, M. *Markov Decision Processes*. John Wiley & Sons, Inc., 1994.

Ruder, S. An overview of gradient descent optimization algorithms, 2016.

Sanner, S. Relational Dynamic Influence Diagram Language (RDDL): Language Description. 2010. URL http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf.

Sanner, S. and Boutilier, C. Approximate linear programming for first-order mdps. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pp. 509–517, 2005.

Sanner, S. and Boutilier, C. Practical solution techniques for first-order mdps. *Artif. Intell.*, 173(5-6):748–788, 2009. doi: 10.1016/j.artint.2008.11.003. URL https://doi.org/10.1016/j.artint.2008.11.003.

Shen, W., Trevizan, F., Toyer, S., Thiébaux, S., and Xie, L. Guiding Search with Generalized Policies for Probabilistic Planning. In *Proc. of 12th Annual Symp. on Combinatorial Search (SoCS)*, 2019. URL http://felipe.trevizan.org/papers/shen19b:guiding.pdf.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sorg, J. and Singh, S. P. Transfer via soft homomorphisms. In *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 2*, pp. 741–748, 2009.

Sridharan, N. S. (ed.). *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*, 1989. Morgan Kaufmann. ISBN 1-55860-094-9. URL http://ijcai.org/proceedings/1989-1.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009. doi: 10.1145/1577069.1755839. URL http://doi.acm.org/10.1145/1577069.1755839.

Toyer, S., Trevizan, F. W., Thiébaux, S., and Xie, L. Action schema networks: Generalised policies with deep learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 2018.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. *CoRR*, abs/1710.10903, 2017. URL http://arxiv.org/abs/1710.10903.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL http://arxiv.org/abs/1505.00853.

Younes, H. L. S., Littman, M. L., Weissman, D., and Asmuth, J. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res.*, 24:851–887, 2005.