The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Action Schema Networks: Generalised Policies with Deep Learning

Sam Toyer,[1] Felipe Trevizan,[1,2] Sylvie Thiébaux,[1] Lexing Xie[1,3]
[1] Research School of Computer Science, Australian National University
[2] Data61, CSIRO [3] Data to Decisions CRC
first.last@anu.edu.au

Abstract

In this paper, we introduce the Action Schema Network (ASNet): a neural network architecture for learning generalised policies for probabilistic planning problems. By mimicking the relational structure of planning problems, ASNets are able to adopt a weight sharing scheme which allows the network to be applied to any problem from a given planning domain. This allows the cost of training the network to be amortised over all problems in that domain. Further, we propose a training method which balances exploration and supervised training on small problems to produce a policy which remains robust when evaluated on larger problems. In experiments, we show that ASNet's learning capability allows it to significantly outperform traditional non-learning planners in several challenging domains.

1 Introduction

Automated planning is the task of finding a sequence of actions which will achieve a goal within a user-supplied model of an environment. Over the past four decades, there has been a wealth of research into the use of machine learning for automated planning (Jiménez et al. 2012), motivated in part by the belief that these two essential ingredients of intelligence—planning and learning—ought to strengthen one other (Zimmerman and Kambhampati 2003). Nevertheless, the dominant paradigm among state-of-the-art classical and probabilistic planners is still based on heuristic state space search. The domain-independent heuristics used for this purpose are capable of exploiting common structures in planning problems, but do not learn from experience. Top planners in both the deterministic and learning tracks of the International Planning Competition often use machine learning to configure portfolios (Vallati et al. 2015), but only a small fraction of planners make meaningful use of learning to produce domain-specific heuristics or control knowledge (de la Rosa, Celorrio, and Borrajo 2008). Planners which transfer knowledge between problems in a domain have been similarly underrepresented in the probabilistic track of the competition.

In parallel with developments in planning, we've seen a resurgence of interest in neural nets, driven largely by their success at problems like image recognition (Krizhevsky, Sutskever, and Hinton 2012) and learning to play video games (Mnih et al. 2013). This paper brings some gains of deep learning to planning by proposing a new neural network architecture, the ASNet, which is specialised to the structure of planning problems much as Convolutional Neural Networks (CNNs) are specialised to the structure of images. The basic idea is illustrated in Figure 1: rather than operating on a virtual graph of pixels with edges defined by adjacency relationships, an ASNet operates on a graph of actions and propositions (i.e. Boolean variables), with edges defined by relations of the form "action a affects proposition p" or "proposition p influences the outcome of action a". This structure allows an ASNet to be trained on one problem from a given planning domain and applied to other, different problems without re-training.

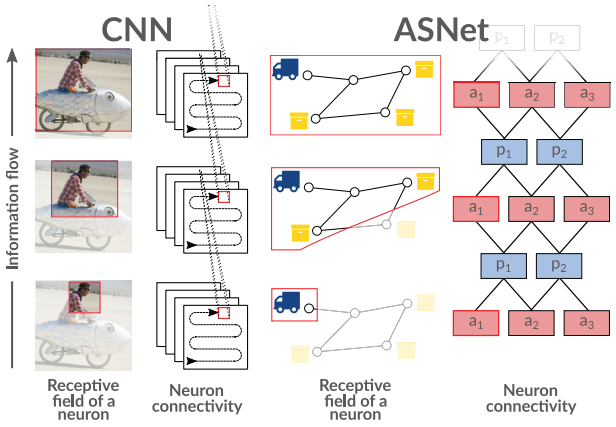


Figure 1: In a CNN, successive convolutions grow the receptive field of a neuron at higher layers; analogously for ASNet, a neuron for a particular action a or proposition p *sees* a larger portion of the current state at higher layers.

We make three new contributions. (1) A neural network architecture for probabilistic planning that automatically generalises to any problem from a given planning domain. (2) A representation that allows weight sharing among actions modules belonging to the same action schema, and among proposition modules associated with the same predicate. This representation is augmented by input features from domain-independent planning heuristics. (3) A train-

ing method that balances exploration and supervision from existing planners. In experiments, we show that this strategy is sufficient to learn effective generalised policies. Code and models for this work are available online. [1]

2 Background

This work considers probabilistic planning problems represented as Stochastic Shortest Path problems (SSPs) (Bertsekas and Tsitsiklis 1996). Formally, an SSP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{G}, s_0)$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition function, $\mathcal{C}\colon \mathcal{S} \times \mathcal{A} \to (0, \infty)$ is a cost function, $\mathcal{G} \subseteq \mathcal{S}$ is a set of goal states, and s_0 is an initial state. At each state s, an agent chooses an action a from a set of enabled actions $\mathcal{A}(s) \subseteq \mathcal{A}$, incurring a cost of $\mathcal{C}(s, a)$ and causing it to transition into another state $s' \in \mathcal{S}$ with probability $\mathcal{T}(s, a, s')$.

The solution of an SSP is a policy $\pi\colon \mathcal{A} \times \mathcal{S} \to [0,1]$ such that $\pi(a \mid s)$ is the probability that action a will be applied in state s. An optimal policy π^* is any policy that minimises the total expected cost of reaching $\mathcal{G}$ from s_0. We do not assume that the goal is reachable with probability 1 from s_0 (i.e. we allow problems with unavoidable dead ends), and a fixed-cost penalty is incurred every time a dead end is reached (Mausam and Kolobov 2012).

A *factored SSP* is a compact representation of an SSP as a tuple $(\mathcal{P}, \mathcal{A}, s_0, s_\star, \mathcal{C})$. $\mathcal{P}$ is a finite set of binary *propositions* and the state space $\mathcal{S}$ is the set of all binary strings of size $|\mathcal{P}|$. Thus, a state s is a value assignment to all the propositions $p \in \mathcal{P}$. A *partial state* is a value assignment to a subset of propositions; a partial state s is consistent with a partial state s' if the value assignments of s' are contained in s ($s' \subseteq s$ for short). The goal is represented by a partial state $s_\star$, and $\mathcal{G} = \{s \in \mathcal{S} | s_\star \subseteq s\}$. Each action $a \in \mathcal{A}$ consists in a precondition pre_a represented by a partial state, a set of effects eff_a each represented by a partial state, and a probability distribution Pr_a over effects in eff_a.[2] The actions applicable in state s are $\mathcal{A}(s) = \{a \in \mathcal{A} \mid pre_a \subseteq s\}$. Moreover, $\mathcal{T}(s, a, s') = \sum_{e \in \mathit{eff}_a | s' = res(s,e)} Pr_a(e)$ where $res(s, e) \in \mathcal{S}$ is the result of changing the value of propositions of s to make it consistent with effect e.

A *lifted* SSP compactly represents a set of factored SSPs sharing the same structure. Formally, a lifted SSP is a tuple $(\mathcal{F}, \mathbb{A}, \mathcal{C})$ where $\mathcal{F}$ is a finite set of predicates, and $\mathbb{A}$ is a finite set of action schemas. Each predicate, when grounded, i.e., instantiated by a tuple of names representing objects, yields a factored SSP proposition. Similarly, each action schema, instantiated by a tuple of names, yields a factored SSP action. The Probabilistic Planning Domain Definition Language (PPDDL) is the standard language to describe lifted and factored SSPs (Younes and Littman 2004). PPDDL splits the description into a general *domain* and a specific *problem*. The domain gives the predicates $\mathcal{F}$, action schemas $\mathbb{A}$ and cost function $\mathcal{C}$ specifying a lifted SSP. The problem additionally gives the set of objects $\mathcal{O}$, initial state s_0 and goal $s_\star$, describing a specific SSP whose propositions and actions are obtained by grounding the domain predicates and action schemas using the objects in $\mathcal{O}$. For instance the domain description might specify a predicate $\mathrm{at}(?l)$ and an action schema $\mathrm{walk}(?\mathrm{from}, ?\mathrm{to})$, while the problem description might specify objects home and work. Grounding using these objects would produce propositions $\mathrm{at}(\mathrm{home})$ and $\mathrm{at}(\mathrm{work})$, as well as ground actions $\mathrm{walk}(\mathrm{work}, \mathrm{home})$ and $\mathrm{walk}(\mathrm{home}, \mathrm{work})$.

Observe that different factored SSPs can be obtained by changing only the problem part of the PPDDL description while reusing its domain. In the next section, we show how to take advantage of action schema reuse to learn policies that can then be applied to any factored SSP obtained by instantiating the same domain.

3 Action Schema Networks

Neural networks are expensive to train, so we would like to amortise that cost over many problems by learning a *generalised policy* which can be applied to any problem from a given domain. ASNet proposes a novel, domain-specialised structure that uses the same set of learnt weights θ regardless of the "shape" of the problem. The use of such a weight sharing scheme is key to ASNet's ability to generalise to different problems drawn from the same domain, even when those problems have different goals or different numbers of actions and propositions.

3.1 Network structure

At a high level, an ASNet is composed of alternating action layers and proposition layers, where action layers are composed of a single *action module* for each ground action, and proposition layers likewise are composed of a single *proposition module* for each ground proposition; this choice of structure was inspired by the alternating action and proposition layers of Graphplan (Blum and Furst 1997). In the same way that hidden units in one layer of a CNN connect only to nearby hidden units in the next layer, action modules in one layer of an ASNet connect only to directly related proposition modules in the next layer, and vice versa. The last layer of an ASNet is always an action layer with each module defining an action selection probability, thus allowing the ASNet to scale to problems with different numbers of actions. For simplicity, we also assume that the first (input) layer is always an action layer.

Action module details. Consider an action module for $a \in \mathcal{A}$ in the lth action layer. The module takes as input a feature vector u_a^l, and produces a new hidden representation

$$\phi_a^l = f(W_a^l \cdot u_a^l + b_a^l)\,,$$

where $W_a^l \in \mathbb{R}^{d_h \times d_a^l}$ is a learnt weight matrix for the module, $b_a^l \in \mathbb{R}^{d_h}$ is a learnt bias vector, $f(\cdot)$ is a nonlinearity (e.g. $\tanh$, sigmoid, or ReLU), d_h is a (fixed) intermediate representation size, and d_a^l is the size of the inputs to the action module. The feature vector u_a^l, which serves as input to the action module, is constructed by enumerating the propositions $p_1, p_2, \ldots, p_M$ which are *related* to the action

[1] https://github.com/qxcv/asnets

[2] Factored SSPs sometimes support conditional effects and negative or disjunctive preconditions and goals. We do not use these here to simplify notation. However, ASNet can easily be extended to support these constructs.

a, and then concatenating their hidden representations. Formally, we say that a proposition $p \in \mathcal{P}$ is related to an action $a \in \mathcal{A}$, denoted $R(a, p)$, if p appears in pre_a or in an effect e where $Pr_a(e) > 0$. Concatenation of representations for the related propositions produces a vector

$$u_a^l = \begin{bmatrix} {\psi_1^{l-1}}^T & \cdots & {\psi_M^{l-1}}^T \end{bmatrix}^T ,$$

where ψ_j^{l-1} is the hidden representation produced by the proposition module for proposition $p_j \in \mathcal{P}$ in the preceding proposition layer. Each of these constituent hidden representations has dimension d_h, so u_a^l has dimension $d_a^l = d_h \cdot M$.

Our notion of propositional relatedness ensures that, if ground actions a_1 and a_2 in a problem are instances of the same action schema in a PPDDL domain, then their inputs u_1^l and u_2^l will have the same "structure". To see why, note that we can determine which propositions are related to a given ground action a by retrieving the corresponding action schema, enumerating the predicates which appear in the precondition or the effects of the action schema, then instantiating those predicates with the same parameters used to instantiate a. If we apply this procedure to a_1 and a_2, we will obtain lists of related propositions $p_1, p_2, \ldots, p_M$ and $q_1, q_2, \ldots, q_M$, respectively, where p_j and q_j are propositions with the same predicate which appear in the same position in the definitions of a_1 and a_2 (i.e. the same location in the precondition, or the same position in an effect).

Such structural similarity is key to ASNet's generalisation abilities. At each layer l, and for each pair of ground actions c and d instantiated from the same action schema s, we use the same weight matrix W_s^l and bias vector b_s^l—that is, we have $W_c^l = W_d^l = W_s^l$ and $b_c^l = b_d^l = b_s^l$. Hence, modules for actions which appear in the same layer and correspond to the same action schema will use the same weights, but modules which appear in different layers or which correspond to different schemas will learn different weights. Although different problems instantiated from the same PPDDL domain may have different numbers of ground actions, those ground actions will still be derived from the same, fixed set of schemas in the domain, so we can apply the same set of action module weights to any problem from the domain.

The first and last layers of an ASNet consist of action modules, but their construction is subtly different:

1. The output of a module for action a in the final layer is a single number $\pi^\theta(a \mid s)$ representing the probability of selecting action a in the current state s under the learnt policy π^θ, rather than a vector-valued hidden representation. To guarantee that disabled actions are never selected, and ensure that action probabilities are normalised to 1, we pass these outputs through a *masked softmax* activation which ensures that $\pi^\theta(a \mid s) = 0$ if $a \notin \mathcal{A}(s)$. During training, we sample actions from $\pi^\theta(a \mid s)$. During evaluation, we select the action with the highest probability.
2. Action modules in the first layer of a ASNet are passed an input vector composed of features derived from the current state, rather than hidden representations for related propositions. Specifically, modules in the first layer are given a binary vector indicating the truth values of related propositions, and whether those propositions appear in the goal. In practice, it is helpful to concatenate these propositional features with heuristic features, as described in Section 3.2.

Proposition module details. Proposition modules only appear in the intermediate layers of an ASNet, but are otherwise similar to action modules. Specifically, a proposition module for proposition $p \in \mathcal{P}$ in the lth proposition layer of the network will compute a hidden representation

$$\psi_p^l = f(W_p^l \cdot v_p^l + b_p^l) ,$$

where v_p^l is a feature vector, f is the same nonlinearity used before, and $W_p^l \in \mathbb{R}^{d_h \times d_p^l}$ and $b_p^l \in \mathbb{R}^{d_h}$ are learnt weights and biases for the module.

To construct the input v_p^l, we first find the predicate $\mathrm{pred}(p) \in \mathcal{F}$ for proposition $p \in \mathcal{P}$, then enumerate all action schemas $A_1, \ldots, A_L \in \mathbb{A}$ which reference $\mathrm{pred}(p)$ in a precondition or effect. We can define a feature vector

$$v_p^l = \begin{bmatrix} \mathrm{pool}(\{{\phi_a^l}^T \mid \mathrm{op}(a) = A_1 \wedge R(a,p)\}) \\ \vdots \\ \mathrm{pool}(\{{\phi_a^l}^T \mid \mathrm{op}(a) = A_L \wedge R(a,p)\}) \end{bmatrix} ,$$

where $\mathrm{op}(a) \in \mathbb{A}$ denotes the action schema for ground action a, and pool is a pooling function that combines several d_h-dimensional feature vectors into a single d_h-dimensional one. Hence, when all pooled vectors are concatenated, the dimensionality d_p^l of v_p^l becomes $d_h \cdot L$. In this paper, we assume that pool performs max pooling (i.e. keeps only the largest input). If a proposition module had to pool over the outputs of many action modules, such pooling could potentially obscure useful information. While the issue could be overcome with a more sophisticated pooling mechanism (like neural attention), we did not find that max pooling posed a major problem in the experiments in Section 5, even on large Probabilistic Blocks World instances where some proposition modules must pool over thousands of inputs.

Pooling operations are essential to ensure that proposition modules corresponding to the same predicate have the same structure. Unlike action modules corresponding to the same action schema, proposition modules corresponding to the same predicate may have a different number of inputs depending on the initial state and number of objects in a problem, so it does not suffice to concatenate inputs. As an example, consider a single-vehicle logistics problem where the location of the vehicle is tracked with propositions of the form $\mathrm{at}(\iota)$, and the vehicle may be moved with actions of the form $\mathrm{move}(\iota_{\text{from}}, \iota_{\text{to}})$. A location ι_1 with one incoming road and no outgoing roads will have only one related move action, but a location ι_2 with two incoming roads and no outgoing roads will have two related move actions, one for each road. This problem is not unique to planning: a similar trick is employed in network architectures for graphs where vertices can have varying in-degree (Jain et al. 2016; Kearnes et al. 2016).

As with the action modules, we share weights between proposition modules for propositions corresponding to the

same predicate. Specifically, at proposition layer l, and for propositions q and r with $\mathrm{pred}(q) = \mathrm{pred}(r)$, we tie the corresponding weights $W_q^l = W_r^l$ and $b_q^l = b_r^l$. Together with the weight sharing scheme for action modules, this enables us to learn a single set of weights

$$\theta = \{W_a^l, b_a^l \mid 1 \leq l \leq n+1, a \in \mathbb{A}\} \cup \{W_p^l, b_p^l \mid 1 \leq l \leq n, p \in \mathcal{F}\}$$

for an n-layer model which can be applied to any problem in a given PPDDL domain.

3.2 Heuristic features for expressiveness

One limitation of the ASNet is the fixed receptive field of the network; in other words, the longest chain of related actions and propositions which it can reason about. For instance, suppose we have I locations $\iota_1, \ldots, \iota_I$ arranged in a line in our previous logistics example. The agent can move from ι_{k-1} to ι_k (for $k = 2, \ldots, I$) with the $\mathrm{move}(\iota_{k-1}, \iota_k)$ action, which makes $\mathrm{at}(\iota_{k-1})$ false and $\mathrm{at}(\iota_k)$ true. The propositions $\mathrm{at}(\iota_1)$ and $\mathrm{at}(\iota_I)$ will thus be related only by a chain of move actions of length $I-1$; hence, a proposition module in the lth proposition layer will only be affected by at propositions for locations at most $l+1$ moves away. Deeper networks can reason about longer chains of actions, but that an ASNet's (fixed) depth necessarily limits its reasoning power when chains of actions can be *arbitrarily* long.

We compensate for this receptive field limitation by supplying the network with features obtained using domain-independent planning heuristics. In this paper, we derive these features from disjunctive action landmarks produced by LM-cut (Helmert and Domshlak 2009), but features derived from different heuristics could be employed in the same way. A disjunctive action landmark is a set of actions in which at least one action must be applied along any optimal path to the goal in a deterministic, delete-relaxed version of the planning problem. These landmarks do not necessarily capture all useful actions, but in practice we find that providing information about these landmarks is often sufficient to compensate for network depth limitations.

In this paper, a module for action a in the first network layer is given a feature vector

$$u_a^1 = \begin{bmatrix} c^T & v^T & g^T \end{bmatrix}^T.$$

$c \in \{0,1\}^3$ indicates whether a_i is the sole action in at least one LM-cut landmark ($c_1 = 1$), an action in a landmark of two or more actions ($c_2 = 1$), or does not appear in a landmark ($c_3 = 1$). $v \in \{0,1\}^M$ represents the M related propositions: v_j is 1 iff p_j is currently true. $g \in \{0,1\}^M$ encodes related portions of the goal state, and g_j is 1 iff p_j is true in the partial state $s_\star$ defining the goal.

4 Training with exploration and supervision

We learn the ASNet weights θ by choosing a set of small training problems P_{train}, then alternating between guided exploration to build up a state memory $\mathcal{M}$, and supervised learning to ensure that the network chooses good actions for the states in $\mathcal{M}$. Algorithm 1 describes a single epoch of exploration and supervised learning. We repeatedly apply this procedure until performance on P_{train} ceases to improve, or until a fixed time limit is reached. Note that this strategy is only intended to learn the *weights* of an ASNet—module connectivity is not learnt, but rather obtained from a grounded representation using the notion of relatedness which we described earlier.

Algorithm 1 Updating ASNet weights θ using state memory $\mathcal{M}$ and training problem set P_{train}

procedure ASNET-TRAIN-EPOCH(θ, $\mathcal{M}$)
  for $i = 1, \ldots, T_{\text{explore}}$ **do** ▷ Exploration
    for all $\zeta \in P_{\text{train}}$ **do**
      $s_0, \ldots, s_N \leftarrow$ RUN-POL$(s_0(\zeta), \pi^\theta)$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{s_0, \ldots, s_N\}$
      for $j = 0, \ldots, N$ **do**
        $s_j^*, \ldots, s_M^* \leftarrow$ POL-ENVELOPE(s_j, π^*)
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{s_j^*, \ldots, s_M^*\}$
  for $i = 1, \ldots, T_{\text{train}}$ **do** ▷ Learning
    $\mathcal{B} \leftarrow$ SAMPLE-MINIBATCH$(\mathcal{M})$
    Update θ using $\frac{d\mathcal{L}_\theta(\mathcal{B})}{d\theta}$ (Equation 1)

In the exploration phase of each training epoch, we repeatedly run the ASNet policy π^θ from the initial state of each problem $\zeta \in P_{\text{train}}$, collecting $N+1$ states $s_0, \ldots, s_N$ visited along each of the sampled trajectories. Each such trajectory terminates when it reaches a goal, exceeds a fixed limit L on length, or reaches a dead end. In addition, for each visited state s_j, we compute an optimal policy π^* rooted at s_j, then produce a set of states $s_j^*, \ldots, s_M^*$ which constitute π^*'s policy envelope—that is, the states which π^* visits with nonzero probability. Both the trajectories drawn from the ASNet policy π^θ and policy envelopes for the optimal policy π^* are added to the state memory $\mathcal{M}$. Saving states which can be visited under an optimal policy ensures that $\mathcal{M}$ always contains states along promising trajectories reachable from s_0. On the other hand, saving trajectories from the exploration policy ensures that ASNet will be able to improve on the states which it visits most often, even if they are not on an optimal goal trajectory.

In the training phase, small subsets of the states in $\mathcal{M}$ are repeatedly sampled at random to produce minibatches for training ASNet. The objective to be minimised for each minibatch $\mathcal{B}$ is the cross-entropy classification loss

$$\mathcal{L}_\theta(\mathcal{B}) = \sum_{s \in \mathcal{B}} \sum_{a \in A} \big[(1 - y_{s,a}) \cdot \log(1 - \pi^\theta(a \mid s)) + y_{s,a} \cdot \log \pi^\theta(a \mid s)\big]. \quad (1)$$

The label $y_{s,a}$ is 1 if the expected cost of choosing action a and then following an optimal policy thereafter is minimal among all enabled actions; otherwise, $y_{s,a} = 0$. This encourages the network to imitate an optimal policy. For each sampled batch $\mathcal{B}$, we compute the gradient $\frac{d\mathcal{L}_\theta(\mathcal{B})}{d\theta}$ and use it to update the weights θ in a direction which decreases $\mathcal{L}_\theta(\mathcal{B})$ with Adam (Kingma and Ba 2015).

The cost of computing an optimal policy during supervised learning is often non-trival. It is natural to ask whether

it is more efficient to train ASNets using unguided policy gradient reinforcement learning, as FPG does (Buffet and Aberdeen 2009). Unfortunately, we found that policy gradient RL was too noisy and inefficient to train deep networks on nontrivial problems; in practice, the cost of computing an optimal policy for small training problems more than pays for itself by enabling us to use sample-efficient supervised learning instead of reinforcement learning. In the experiments, we investigate the question of whether suboptimal policies are still sufficient for supervised training of ASNets.

Past work on generalised policy learning has employed learnt policies as control knowledge for search algorithms, in part because doing so can compensate for flaws in the policy. For example, Yoon, Fern, and Givan (2007) suggest employing policy rollout or limited discrepancy search to avoid the occasional bad action recommended by a policy. While we could use an ASNet similarly, we are more interested in its ability to learn a reliable policy on its own. Hence, during evaluation, we always choose the action which maximises $\pi^\theta(a \mid s)$. As noted above, this is different from the exploration process employed during training, where we instead sample from $\pi^\theta(a \mid s)$.

5 Experiments and discussion

In this section, we compare ASNet against state-of-the-art planners on three planning domains.

5.1 Experimental setup

We compare ASNet against three heuristic-search-based probabilistic planners: LRTDP (Bonet and Geffner 2003), ILAO* (Hansen and Zilberstein 2001) and SSiPP (Trevizan and Veloso 2014). Two domain-independent heuristics are considered for each of the three planners—LM-cut (admissible) and the additive heuristic h^{add} (inadmissible) (Teichteil-Königsbuch, Vidal, and Infantes 2011)—resulting in 6 baselines. During evaluation, we enforce a 9000s time cutoff for all the baselines and ASNets, as well as a 10Gb memory cutoff.

Since LRTDP and ILAO* are optimal planners, we execute them until convergence ($\epsilon = 10^{-4}$) for each problem using 30 different random seeds. Notice that, for h^{add}, LRTDP and ILAO* might converge to a suboptimal solution. If an execution of LRTDP or ILAO* does not converge before the given time/memory cutoff, we consider the planner as having failed to reach the goal. SSiPP is used as a replanner and, for each problem, it is *trained* until 60s before the time cutoff and then evaluated; this procedure is repeated 30 times for each problem using different random seeds. The training phase of SSiPP consists in simulating a trajectory from s_0 and, during this process, SSiPP improves its lower bound on the optimal solution. If 100 consecutive trajectories reach the goal during training, then SSiPP is evaluated regardless of the training time left. For the 6 baselines, we report the average running time per problem.

For each domain, we train a single ASNet, then evaluate it on each problem 30 times with different random seeds. The hyperparmeters for each ASNet were kept fixed across domains: three action layers and two proposition layers in each network, a hidden representation size of 16 for each internal action and proposition module, and an ELU (Clevert, Unterthiner, and Hochreiter 2016) as the nonlinearity f. The optimiser was configured with a learning rate of 0.0005 and a batch size of 128, and a hard limit of two hours (7200s) was placed on training. We also applied ℓ_2 regularisation with a coefficient of 0.001 on all weights, and dropout on the outputs of each layer except the last with $p = 0.25$. Each epoch of training alternated between 25 rounds of exploration shared equally among all training problems, and 300 batches of network optimisation (i.e. $T_{\text{explore}} = 25/|P_{\text{train}}|$ and $T_{\text{train}} = 300$). Sampled trajectory lengths are $L = 300$ for both training and evaluation. LRTDP with the LM-cut heuristic is used for computing the optimal policies during training, with a dead-end penalty of 500. We also repeated this procedure for LRTDP using h^{add} (inadmissible heuristic) to compare the effects of using optimal and suboptimal policies for training. Further, we report how well ASNet performs when it is guided by h^{add}, but *not* given the LM-cut-derived heuristic features described in Section 3.2. For the ASNets, we report the average *training time plus time to solve the problem* to highlight when it pays off to spend the one-off cost of training an ASNet for a domain.

All ASNets were trained and evaluated on a virtual machine equipped with 62GB of memory and an x86-64 processor clocked at 2.3GHz. For training and evaluation, each ASNet was restricted to use a single, dedicated processor core, but resources were otherwise shared. The baseline planners were run in a cluster of x86-64 processors clocked at 2.6GHz and each planner again used only a single core.

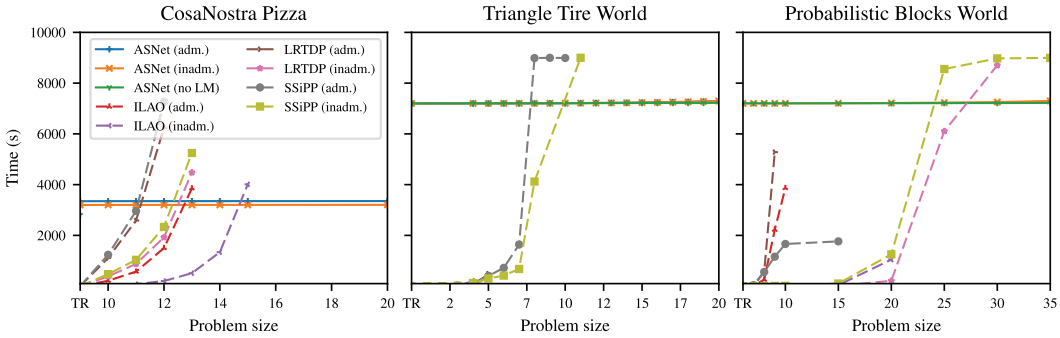
5.2 Domains

We evaluate ASNets and the baselines on the following probabilistic planning domains:

CosaNostra Pizza: as a Deliverator for CosaNostra Pizza, your job is to safely transport pizza from a shop to a waiting customer, then return to the shop. There is a series of toll booths between you and the customer: at each booth, you can either spend a time step paying the operator, or save a step by driving through without paying. However, if you don't pay, the (angry) operator will try to drop a boom on your car when you pass through their booth on the way back to the shop, crushing the car with 50% probability. The optimal policy is to pay operators when travelling to the customer to ensure a safe return, but not pay on the return trip as you will not revisit the booth. Problem size is the number of toll booths between the shop and the customer. ASNets are trained on sizes 1-5, and tested on sizes 6+.

Probabilistic Blocks World is an extension of the well-known deterministic blocks world domain in which a robotic arm has to move blocks on a table into a goal configuration. The actions to pick up a block or to put a block on top of another fail with probability 0.25; failure causes the target block to drop onto the table, meaning that it must be picked up and placed again. We randomly generate three different problems for each number of blocks considered during testing. ASNet is trained on five randomly generated problems of each size from 5–9, for 25 training problems total.

Triangle Tire World (Little and Thiébaux 2007): each



Problem			ASNet	Others
CN	14	Cover.	30/30	30/30
		Avg. C.	46.0	46.0
	15	Cover.	30/30	30/30
		Avg. C.	49.0	49.0
TTW	7	Cover.	30/30	30/30
		Avg. C.	40.8	41.4
	10	Cover.	30/30	1/30
		Avg. C.	59.0	60.0
PBW	25	Cover.	30/30	30/30
		Avg. C.	92.9	145.6
	30	Cover.	30/30	27/30
		Avg. C.	116.4	340.0
	35	Cover.	30/30	1/30
		Avg. C.	138.8	366.0

Figure 2: Comparison of planner running times on the evaluation domains. TR refers to the time used for training (zero for baselines). ASNet runs with (adm.) used optimal policies for training while (inadm.) used potentially suboptimal policies, and runs with (no LM) did not use heuristic input features. The table at right shows, for selected problems, the coverage and average solution cost for the best ASNet and baseline. We use TTW for Triangle Tire World, CN for CosaNostra Pizza, and PBW for Probabilistic Blocks World. In PBW, running times are averaged over the three problems of each size. In TTW and PBW, ASNet (no LM) occludes ASNet (inadm.). ASNet (adm.) is also occluded in TTW, but is absent entirely from PBW as the optimal planner used to generate training data could not solve all training problems in time.

problem consists of a set of locations arranged in a triangle, with connections between adjacent locations. The objective is to move a vehicle from one corner of the triangle to another. However, each move has a 50% chance of producing a flat tire, which must be replaced at the next visited location. The vehicle thus requires a sequence of moves between locations where replacement tires are available. Tires are arranged such that the most reliable policy is one which travels the *longest* path to the goal, along the outside edge of the triangle. This task can be made more challenging by scaling up the number of locations. Per Little and Thiébaux (2007), a problem of size n has $(n+1)(2n+1)$ locations. We use sizes 1-3 for training, and test with sizes from 4 onward.

5.3 Results

Figure 2 shows the time taken to train and evaluate ASNet using optimal (adm.) and suboptimal (inadm.) policies as training data. In addition, it shows coverage (proportion of runs which reached the goal) and average solution cost when the goal is reached for selected problems for the best ASNet and best baseline. The following is a summary of our results:

When is it worth using ASNet? All ASNets obtained 30 out of 30 coverage for all Triangle Tire World problems, and the ASNets with heuristic input features similarly obtained perfect coverage on CosaNostra. In contrast, the baselines failed to scale up to the larger problems. This shows that ASNet is well-suited to problems where local knowledge of the environment can help to avoid common traps, for instance: in CosaNostra, the agent must learn to pay toll booth operators when carrying a pizza and not pay otherwise; and in Triangle Tire World, the agent must learn to sense and follow the outer edge of the triangle. Not only could ASNets learn these tricks, but the average solution cost obtained by ASNets for CosaNostra and Triangle Tire World was close to that of the optimal baselines (when they converged), suggesting that the optimal solution was found.

Probabilistic Blocks World is more challenging as there is no single pattern that can solve all problems. Even for the deterministic version of Blocks World, a generalised policy requires the planner to learn a recursive property for whether each block is in a goal position (Slaney and Thiébaux 2001). The ASNet appears to have successfully learnt to do this when trained by a suboptimal teacher and given landmarks as input, and surpassed all baselines in coverage (reaching the goal on 30/30 runs on each instance). Moreover, the average solution cost of ASNet (inadm.) is similar to the optimal baselines (when they converge) and up to 3.7 times less than SSiPP (inadm.), the baseline with the best coverage. The ASNet (inadm.) policy typically obtained a mean solution cost somewhere between the US and GN1 strategies presented by Slaney and Thiébaux: it is suboptimal, but still better than unstacking and rebuilding all towers from scratch. Note that the ASNet could not obtain a policy within the allotted time when trained by an optimal teacher.

Are the heuristic features necessary? In some cases, ASNet's performance can be improved by omitting (expensive) LM-cut heuristic input features. For instance, in Triangle Tire World, ASNet (inadm.) took 2.4x as much time as ASNet (no LM) to solve problems of size 15, and 4.3x as much time to solve problems of size 20, despite executing policies of near-identical average cost. Notice that this difference cannot be seen in Figure 2 because the training time (TR) is much larger than the time to solve a test instance.

Interestingly, ASNet (no LM) was able to obtain 100% coverage on the Probabilistic Blocks World problems in Figure 2, despite not receiving landmark inputs. To gain stronger assurance that it had learnt a robust policy, we tested on 10 more instances with 10, 15, 20, 25, 30 and 35 blocks (60 more instances total). ASNet (no LM) could not solve all the additional test instances. In contrast, ASNet (inadm.)—which *was* given landmarks as input—reliably solved all test problems in the extended set, thus showing that heuristic inputs are necessary to express essential recursive properties like whether a block is in its goal position.

Heuristic inputs also appear to be necessary in CosaNostra, where ASNet (no LM) could not achieve full coverage on the test set. We suspect that this is because an ASNet without heuristic inputs cannot determine which direction leads to the pizza shop and which direction leads to the customer when it is in the middle of a long chain of toll booths.

How do suboptimal training policies affect ASNet? Our results suggest that use of a suboptimal policies is sufficient to train ASNet, as demonstrated in all three domains. Intuitively, the use of suboptimal policies for training ought to be beneficial because the time that would have been spent computing an optimal policy can instead be used for more epochs of exploration and supervised learning. This is somewhat evident in CosaNostra—where a suboptimal training policy allows for slightly faster convergence—but it is more clear in Probabilistic Blocks World, where the ASNet can only converge within our chosen time limit with the inadmissible policy. While training on fewer problems allowed the network to converge within the time limit, it did not yield as robust a policy, suggesting that the use of a suboptimal teacher is sometimes a necessity.

Is ASNet performing fixed-depth lookahead search? No. This can be seen by comparing SSiPP and ASNet. SSiPP solves fixed-depth sub-problems (a generalization of lookahead for SSPs) and is unable to scale up as well as ASNets when using an equivalent depth parametrisation. Triangle Tire World is particularly interesting because SSiPP can outperform other baselines by quickly finding dead ends and avoiding them. However, unlike an ASNet, SSiPP is unable to generalize the solution of one sub-problem to the next and needs to solve all of them from scratch.

6 Related work

Generalised policies are a topic of interest in planning (Zimmerman and Kambhampati 2003; Jiménez et al. 2012; Hu and De Giacomo 2011). The earliest work in this area expressed policies as decision lists (Khardon 1999), but these were insufficiently expressive to directly capture recursive properties, and thus required user-defined *support predicates*. Later planners partially lifted this restriction by expressing learnt rules with *concept language* or *taxonomic syntax*, which can capture such properties directly (Martin and Geffner 2000; Yoon, Fern, and Givan 2002; 2004). Other work employed features from domain-independent heuristics to capture recursive properties (de la Rosa et al. 2011; Yoon, Fern, and Givan 2006), just as we do with LM-cut landmarks. Srivastava et al. (2011) have also proposed a substantially different generalised planning strategy that provides strong guarantees on plan termination and goal attainment, albeit only for a restricted class of deterministic problems. Unlike the decision lists (Yoon, Fern, and Givan 2002; 2004) and relational decision trees (de la Rosa et al. 2011) employed in past work, our model's input features are fixed before training, so we do not fall prey to the *rule utility problem* (Zimmerman and Kambhampati 2003). Further, our model can be trained to minimise any differentiable loss, and could be modified to use policy gradient reinforcement learning without changing the model. While our approach cannot give the same theoretical guarantees as Srivastava et al., we are able to handle a more general class of problems with less domain-specific information.

Neural networks have been used to learn policies for probabilistic planning problems. The Factored Policy Gradient (FPG) planner trains a multi-layer perceptron with reinforcement learning to solve a factored MDP (Buffet and Aberdeen 2009), but it cannot generalise across problems and must thus be trained anew on each evaluation problem. Concurrent with this work, Groshev et al. (2017) propose generalising "reactive" policies and heuristics by applying a CNN to a 2D visual representation of the problem, and demonstrate an effective learnt heuristic for Sokoban. However, their approach requires the user to define an appropriate visual encoding of states, whereas ASNets are able to work directly from a PPDDL description.

The integration of planning and neural networks has also been investigated in the context of deep reinforcement learning. For instance, Value Iteration Networks (Tamar et al. 2016; Niu et al. 2017) (VINs) learn to formulate and solve a probabilistic planning problem within a larger deep neural network. A VIN's internal model can allow it to learn more robust policies than would be possible with ordinary feedforward neural networks. In contrast to VINs, ASNets are intended to learn reactive policies for known planning problems, and operate on factored problem representations instead of (exponentially larger) explicit representations like those used by VINs.

In a similar vein, Kansky et al. present a model-based RL technique known as schema networks (Kansky et al. 2017). A schema network can learn a transition model for an environment which has been decomposed into entities, but where those entities' interactions are initially unknown. The entity–relation structure of schema networks is reminiscent of the action–proposition structure of an ASNet; however, the relations between ASNet modules are obtained through grounding, whereas schema networks learn which entities are related from scratch. As with VINs, schema networks tend to yield agents which generalise well across a class of similar environments. However, unlike VINs and ASNets—which both learn policies directly—schema networks only learn a model of an environment, and planning on that model must be performed separately.

Extension of convolutional networks to other graph structures has received significant attention recently, as such networks often have helpful invariances (e.g. invariance to the order in which nodes and edges are given to the network) and fewer parameters to learn than fully connected networks. Applications include reasoning about spatio-temporal relationships between variable numbers of entities (Jain et al. 2016), molecular fingerprinting (Kearnes et al. 2016), visual question answering (Teney, Liu, and Hengel 2017), and reasoning about knowledge graphs (Kipf and Welling 2017). To the best of our knowledge, this paper is the first such technique that successfully solves factored representations of automated planning problems.

7 Conclusion

We have introduced the ASNet, a neural network architecture which is able to learn generalised policies for proba-

bilistic planning problems. In much the same way that CNNs can generalise to images of arbitrary size by performing only repeated local operations, an ASNet can generalise to different problems from the same domain by performing only convolution-like operations on representations of actions or propositions which are *related* to one another. In problems where some propositions are only related by long chains of actions, ASNet's modelling capacity is limited by its depth, but it is possible to avoid this limitation by supplying the network with heuristic input features, thereby allowing the network to solve a range of problems.

While we have only considered supervised learning of generalised policies, the ASNet architecture could in principle be used to learn heuristics or embeddings, or be trained with reinforcement learning. ASNet only requires a model of which actions affect which portion of a state, so it could also be used in other settings beyond SSPs, such as MDPs with Imprecise Probabilities (MDPIPs) (White III and Eldeib 1994) and MDPs with Set-Valued Transitions (MDP-STs) (Trevizan, Cozman, and Barros 2007). We hope that future work will be able to explore these alternatives and use ASNets to further enrich planning with the capabilities of deep learning.

References

Bertsekas, D., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Blum, A. L., and Furst, M. L. 1997. Fast planning through planning graph analysis. *AIJ*.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: improving the convergence of real-time dynamic programming. In *AAAI*.

Buffet, O., and Aberdeen, D. 2009. The factored policy-gradient planner. *AIJ*.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and accurate deep network learning by exponential linear units (ELUs). *ICLR*.

de la Rosa, T.; Jiménez, S.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *JAIR*.

de la Rosa, T.; Celorrio, S. J.; and Borrajo, D. 2008. Learning relational decision trees for guiding heuristic planning. In *ICAPS*.

Groshev, E.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2017. Learning generalized reactive policies using deep neural networks. *arXiv:1708.07280*.

Hansen, E. A., and Zilberstein, S. 2001. LAO: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *ICAPS*.

Hu, Y., and De Giacomo, G. 2011. Generalized planning: Synthesizing plans that work for multiple environments. In *IJCAI*.

Jain, A.; Zamir, A. R.; Savarese, S.; and Saxena, A. 2016. Structural-RNN: Deep learning on spatio-temporal graphs. In *CVPR*.

Jiménez, S.; de la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *Knowl. Eng. Rev.*

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *ICML*.

Kearnes, S.; McCloskey, K.; Berndl, M.; Pande, V.; and Riley, P. 2016. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*.

Khardon, R. 1999. Learning action strategies for planning domains. *AIJ*.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

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

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Little, I., and Thiébaux, S. 2007. Probabilistic planning vs. replanning. In *ICAPS workshops*.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *KRR*.

Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes*. Morgan & Claypool.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. In *NIPS workshops*.

Niu, S.; Chen, S.; Guo, H.; Targonski, C.; Smith, M. C.; and Kovačević, J. 2017. Generalized value iteration networks: Life beyond lattices. *arXiv:1706.02416*.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *AIJ*.

Srivastava, S.; Immerman, N.; Zilberstein, S.; and Zhang, T. 2011. Directed search for generalized plans using classical planners. In *ICAPS*.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *NIPS*.

Teichteil-Königsbuch, F.; Vidal, V.; and Infantes, G. 2011. Extending Classical Planning Heuristics to Probabilistic Planning with Dead-Ends. In *AAAI*.

Teney, D.; Liu, L.; and Hengel, A. v. d. 2017. Graph-structured representations for visual question answering. In *CVPR*.

Trevizan, F., and Veloso, M. 2014. Depth-based Short-sighted Stochastic Shortest Path Problems. *Artificial Intelligence*.

Trevizan, F.; Cozman, F. G.; and Barros, L. N. 2007. Planning under risk and knightian uncertainty. In *IJCAI*.

Vallati, M.; Chrpa, L.; Grześ, M.; McCluskey, T. L.; Roberts, M.; Sanner, S.; et al. 2015. The 2014 International Planning Competition: Progress and trends. *AI Mag.*

White III, C. C., and Eldeib, H. K. 1994. Markov decision processes with imprecise transition probabilities. *Operations Research* 42(4):739–749.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *UAI*.

Yoon, S.; Fern, A.; and Givan, R. 2004. Learning reactive policies for probabilistic planning domains. In *IPC Probabilistic Track*.

Yoon, S. W.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *ICAPS*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Using learned policies in heuristic-search planning. In *IJCAI*.

Younes, H. L., and Littman, M. L. 2004. PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Mag.*