Journal of Artificial Intelligence Research 0 (0000) 0-00 Submitted 0/00; published 0/00

ASNets: Deep Learning for Generalised Planning

Sam Toyer SDT@BERKELEY.EDU
University of California, Berkeley

Sylvie Thiébaux SYLVIE.THIEBAUX@ANU.EDU.AU
Felipe Trevizan FELIPE.TREVIZAN@ANU.EDU.AU
Lexing Xie LEXING.XIE@ANU.EDU.AU
The Australian National University

Abstract

In this paper, we discuss the learning of generalised policies for probabilistic and classical planning problems using Action Schema Networks (ASNets). The ASNet is a neural network architecture that exploits the relational structure of (P)PDDL planning problems to learn a common set of weights that can be applied to any problem in a domain. By mimicking the actions chosen by a traditional, non-learning planner on a handful of small problems in a domain, ASNets are able to learn a generalised reactive policy that can quickly solve much larger instances from the domain. This work extends the ASNet architecture to make it more expressive, while still remaining invariant to a range of symmetries that exist in PPDDL problems. We also present a thorough experimental evaluation of ASNets, including a comparison with heuristic search planners on seven probabilistic and deterministic domains, an extended evaluation on over 18,000 Blocksworld instances, and an ablation study. Finally, we show that sparsity-inducing regularisation can produce ASNets that are compact enough for humans to understand, yielding insights into how the structure of ASNets allows them to generalise across a domain.

1. Introduction

Learning and planning are both important ingredients for constructing intelligent agents. Planning can help an agent choose actions which will achieve its long-term goals by reasoning about future trajectories, and learning can assist the agent in using prior experience to more efficiently achieve new goals. However, the prevalent methods for automated planning in the AI literature make limited use of learning. Some planning methods like RTDP are said to "learn" in the sense that they use an iterative algorithm to come to successively better estimates of the value of a certain state in a problem (Barto, Bradtke, & Singh, 1995), but cannot transfer that learnt knowledge to other problem instances. Other planners can be used in conjunction with autoselectors and autoconfigurators that predict which combination of planner and planner configuration might work best for a given problem instance, based on features of that instance. These sorts of learning-based portfolio planners commonly feature in the International Planning Competition (IPC), as they are able to combine the disjoint strengths of many algorithms (Coles, Coles, Olaya, Jiménez, López, Sanner, & Yoon, 2012; Vallati, Chrpa, Grześ, McCluskey, Roberts, Sanner, et al., 2015). However, while an autoselector and autoconfigurator may be able to transfer knowledge across many instances, they are typically limited to choosing between a handful of planners or changing only a few planner settings. Despite some work over the past two decades and a recent uptick in interest (Martin & Geffner, 2000; Yoon, Fern, & Givan, 2002; de la Rosa, Jiménez, Fuentetaja, & Borrajo, 2011; Bonet & Geffner, 2018; Bonet, Frances, & Geffner, 2019), how best to learn and transfer deeper forms of knowledge across instances—such as partial solutions, or knowledge of dead ends, or heuristics—is still an open problem.

Separate to developments in planning, the broader AI community has recently seen a resurgence of interest in neural networks. This interest has been driven by the success of deep learning in tackling problems ranging from image classification (Krizhevsky, Sutskever, & Hinton, 2012) to video game playing (Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, & Riedmiller, 2013) and machine translation (Wu, Schuster, Chen, Le, Norouzi, Macherey, Krikun, Cao, Gao, Macherey, et al., 2016). LeCun, Bengio, and Hinton (2015) argue that deep learning has met with greater success than other machine learning techniques in these domains due to its ability to automatically extract structure from high-level data, thereby obviating the need for laborious feature engineering. LeCun et al. also stress the importance of having appropriate deep learning "architectures" for processing different modalities of input. For instance, Convolutional Neural Networks (CNNs) are particularly well-suited to processing image data, since they naturally capture notions like translation invariance and hierarchical composition of features, and can be efficiently applied to images of arbitrary size at test time (Long, Shelhamer, & Darrell, 2015). Similarly, the ability of bidirectional Recurrent Neural Networks (RNNs) to (in principle) capture long-range dependencies in sequential data of arbitrary length makes them a natural choice for text processing tasks. However, there is not yet a standard neural network architecture that can do for planning problems what CNNs do for images, or what RNNs do for text. The absence of appropriate architectures is a barrier that must be overcome before we see greater adoption of deep learning in automated planning.

Action Schema Networks (ASNets) are one of the first attempts to bridge the worlds of automated planning and deep learning (Toyer, Trevizan, Thiébaux, & Xie, 2018). ASNets generalise the notion of a "convolution" to match the relational structure of factored planning problems. Where 2D CNNs operate on regular grids of features corresponding to locations of pixels in an image, ASNets instead operate on an abstract graph of features corresponding to actions and propositions from a planning problem. The connections between actions and propositions in this graph are derived from the action schemas for the corresponding planning domain. This scheme makes it possible to share weights between policy networks instantiated for different problems in a domain. Hence, a set of small problems from a given domain can be used to learn a single set of parameters which can be transferred to all other problems in that domain. In other words, an appropriately-learnt set of weights can be used to obtain a generalised policy. Geffner (2018b) observes that this kind of generalisation is not possible with fully connected neural networks, which need to have fixed input and output sizes throughout training and evaluation. In a sense, learning generalised policies with ASNets represents a much tighter integration of machine learning with automated planning than other strategies like autoconfiguration, which only learn to tweak a handful of parameters for a hand-coded planning algorithm. Further, the flexible structure and generalisation capacity of ASNets could make them a suitable tool for other tasks beyond learning generalised policies, such as guiding tree search (Shen, Trevizan, Toyer, Thiébaux, & Xie, 2019) or learning generalised heuristics (Shen, Trevizan, & Thiébaux, 2020).

This paper expands upon the original ASNets paper (Toyer et al., 2018) in several ways. Section 3 extends the original architecture with a more expressive pooling mechanism, as well as *skip connections* between modules of the same type in different layers. In Section 5, we perform a more thorough evaluation of ASNets across seven probabilistic and deterministic tasks. The expanded evaluation includes four new tasks, an extended evaluation on 18,300 Blocksworld instances, and an ablation study identifying which ASNet features are most important for obtaining high coverage on our test domains. In Section 6, we present a method for interpreting ASNet policies, and apply this to a policy for the Triangle Tireworld domain to better understand the mechanism that allows ASNets to generalise. Finally, in Section 7, we connect this work to the large body of relevant literature on deep learning and automated planning.

2. Background

This paper considers the task of solving Stochastic Shortest Path Problems (SSPs) (Bertsekas & Tsitsiklis, 1996). An SSP can be represented by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{C}, \mathcal{G}, s_0)$ consisting of finite sets of states $\mathcal{S}$ and actions $\mathcal{A}$, a transition probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, a cost function $\mathcal{C} : \mathcal{S} \times \mathcal{A} \to (0, \infty)$, a set of goal states $\mathcal{G} \subseteq \mathcal{S}$, and an initial state $s_0 \in \mathcal{S}$. An agent following a policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$ in an SSP will start in state s_0, then repeatedly choose an action $a \sim \pi(a|s)$ and execute it to reach a new state $s' \sim \mathcal{T}(s'|s, a)$ while incurring cost $\mathcal{C}(s, a)$ along the way. An optimal solution to an SSP is a policy $\pi^*(a|s)$ which reaches a goal state $s_g \in \mathcal{G}$ with probability 1 while minimising total expected cost. In order to consider problems with unavoidable dead ends, we relax the requirement that policies should reach the goal with probability 1. Instead, policies are permitted to enter dead-end states, but at the cost of incurring a large (but finite) dead end penalty (Mausam & Kolobov, 2012). Note that the classical (deterministic) planning setting can be viewed as a special case of this general SSP setting in which the transition probability distribution $\mathcal{T}(s'|s, a)$ is deterministic.

Large SSPs are typically specified using a *factored* representation $(\mathcal{P}, \mathcal{A}, s_0, s_\star, \mathcal{C})$. $\mathcal{P}$ is a finite set of propositions (binary variables). Each state $s \subseteq \mathcal{P}$ corresponds to the set of propositions that are true, with the remaining propositions taken to be false. The full state space is thus of size $|\mathcal{S}| = 2^{|\mathcal{P}|}$. $s_0 \subseteq \mathcal{P}$ represents the initial state. $s_\star \subseteq \mathcal{P}$ is a subset of propositions that defines the set of goal states $\mathcal{G} = \{s \in \mathcal{S} | s_\star \subseteq s\}$ in which these propositions are all true and the remaining propositions can be either true or false. Each action $a \in \mathcal{A}$ consists of a precondition pre_a and a distribution Pr_a over a set of deterministic effects eff_a. The precondition $\mathrm{pre}_a \subseteq \mathcal{P}$ represents the conditions that must be satisfied before applying a—a can only be applied in a state s such that $\mathrm{pre}_a \subseteq s$. Thus, the set of actions *applicable* in a state s is $\mathcal{A}(s) = \{a \in \mathcal{A} | \mathrm{pre}_a \subseteq s\}$. On applying $a \in \mathcal{A}(s)$, a deterministic effect $e \in \mathrm{eff}_a$ is sampled from Pr_a. Each deterministic effect e consists of a set of add-effects $\mathrm{add}(e) \subseteq P$ and a set of delete-effects $\mathrm{del}(e) \subseteq P$; applying e in state s yields a new state $s' = \mathrm{apply}(e, s) = (s \setminus \mathrm{del}(e)) \cup \mathrm{add}(e)$ in which the truth values of some propositions have changed. $\mathcal{A}$ thus gives rise to a transition probability distribution $\mathcal{T}(s'|s, a) = \sum_{e \in \mathrm{eff}_a | s' = \mathrm{apply}(e,s)} \mathrm{Pr}_a(e)$, where $\mathcal{T}(s'|s, a) = 0$ when $a \notin \mathcal{A}(s)$. Finally, the semantics of the cost function $\mathcal{C}(s, a) \in (0, \infty)$ and initial state s_0 are unchanged from their definitions above.

Formally, we view a *generalised* policy (of the sort learnt by an ASNet) as a solution to the family of all factored SSPs that can be instantiated from a certain *lifted SSP*. A lifted SSP is a tuple $(\mathbb{P}, \mathbb{A}, \mathcal{C})$ consisting of a set of predicates $\mathbb{P}$, a set of action schemas $\mathbb{A}$, and a cost function $\mathcal{C}$. A predicate can be viewed as a function that produces a concrete proposition from a tuple of *objects*, which are names that represent entities in an environment. Repeating this *grounding* process for each predicate and each applicable tuple of objects in an object set $\mathcal{O}$ can thus yield a set of propositions $\mathcal{P}$. For instance, say we are given predicates $\mathbb{P} = \{\mathrm{at}(?robot, ?place)\}$ and objects $\mathcal{O} = \{shakey, hall, kitchen\}$. After grounding, we would end up with a set of propositions $\mathcal{P} = \{\mathrm{at}(shakey, kitchen), \mathrm{at}(shakey, hall), \ldots\}$ which we could use to represent the possible locations of a robot named *shakey*. An action schema can likewise be interpreted as a function that maps a tuple of objects to a concrete action. As an example, we could take a schema $\mathrm{drive}(?robot, ?from, ?to)$ and a tuple of objects $(shakey, kitchen, hall)$ to instantiate a concrete action $\mathrm{drive}(shakey, kitchen, hall)$ which moves *shakey* from the *kitchen* to the *hall*. In this way, a complete factored SSP can be instantiated from a lifted SSP $(\mathbb{P}, \mathbb{A}, \mathcal{C})$, a set of objects $\mathcal{O}$, an initial state s_0, and a partial state $s_\star$ specifying the goal. As we will see in later sections, factored SSPs that have been instantiated from a shared set of predicates and action schemas typically have a similar structure that can be exploited to learn compact generalised policies.

Lifted SSPs and factored SSPs are often specified using *domain* and *problem* definitions written in Probabilistic Planning Domain Definition Language (PPDDL) (Younes & Littman, 2004). A PPDDL domain defines a lifted SSP, while a PPDDL problem can be combined with the action schemas and predicates from a domain to specify a factored SSP. Figure 1 shows an example of such a problem and its corresponding domain.

In addition to the constructs that we mentioned when introducing factored SSPs, PPDDL also supports more complex language features. Some of those features are supported by the ASNet architecture without any kind of additional compilation or reduction. Such features include arbitrarily nested conditional and probabilistic effects; nested precondition formulae featuring disjunction, negation, etc.; and stochastic initial state distributions. As will become clear in Section 3 and Section 4, the precise semantics of the preconditions and effects are only relevant for generating training data. In contrast, when constructing the network, all that matters is where each proposition appears in the precondition or effect of each action (if at all). We use this information to structure the network in such a way that it can generalise to different problems from the same domain, but the semantics of preconditions, effects, propositions, and so on are otherwise irrelevant to the network architecture. For clarity of exposition we will therefore ignore these additional language features, and instead pretend that all PPDDL problems and domains are given in the STRIPS-like form introduced earlier.

Although ASNets support most PPDDL features, there are four PPDDL constructs that ASNets do not yet support: numeric variables, rewards, quantifiers, and arbitrary goal formulae. The lack of support for numeric variables is simply an implementation omission: neural networks are capable of taking scalar inputs, so in principle these could be handled in the same way that propositions are handled currently. However, we deemed a full treatment of numeric problems to be out of scope for this paper. Likewise, ASNets do not yet support PPDDL rewards, but adding support would be straightforward given support for numeric variables and an appropriate teacher planner for generating training data. In contrast, the remaining two constructs are not supported due to actual structural limitations of ASNets:

- **Universal and existential quantifiers:** Generalisation across problems with ASNets requires a specific invariant to hold: if two actions are instantiated from the same action schema, then there should be a one-to-one correspondence between the propositions appearing in the preconditions/effects of the first action and the preconditions/effects of the second action. This should be true even if the two actions are instantiated for different problems from the same domain, and is essential to the mechanism by which ASNets generalise control knowledge across problems. The use of universal or existential quantifiers can create pairs of ground actions which are instantiated from the same action schema, but which do not even have the same number of propositions in their respective preconditions or in their respective effects. Thus, quantifiers are not supported. One way to lift this architectural limitation would be to augment the *action modules* described in Section 3.2 with something akin to the pooling mechanism used for *proposition modules* in Section 3.3. We did not require this capability for our evaluation domains, and so did not investigate it further.

- **Arbitrary goal formulae:** Each problem associated with a given PPDDL domain could have a different formula describing its goal. These formulae might have different structures in different problems: one problem could have a goal expression consisting of a single literal, while another might have a goal expression with deeply nested conjunctions, disjunctions, quantifiers, and so on. To train a generalised policy, there needs to be some regular way of representing goal expressions from different problems. One could imagine addressing this issue by compiling the goal formula for any given problem into a new or existing action. Unfortunately, that would violate ASNets' requirement that all actions for all problems in a domain be instantiated from exactly the same set of action schemas, and so a different solution is required. In Section 3.4, we instead suggest using a vector that indicates, for each proposition in the problem, whether the proposition must be made true in the goal state. Propositions that do not need to be made true are assumed to be irrelevant to the goal. This representation is only suitable for goals which are conjunctions of positive literals. Lifting this restriction would likely require some kind of goal-processing network that generalises to different goal expressions in the same way that ASNets generalise to different planning problems. We leave this challenging problem to future work.

  The same limitations do *not* apply to action preconditions. The "structure" of an action's precondition is determined by the corresponding action schema in the domain, and so does not change across different problems from the same domain.

Finally, a note on grounding: the internal structure of an ASNet for a particular task is dependent on the number of actions and propositions in the grounded problem, which is in turn dependent on the choice of grounding algorithm. The grounding algorithm does *not* affect the number and shape of network parameters, which is problem-independent (as described in Section 3). However, it does affect the number of "neurons" in an ASNet and their connectivity: a naive grounding algorithm could produce a much larger network than a grounding algorithm with sensible optimisations. Our experiments in Section 5 use the

```
(define (domain unreliable-robot-domain)
  ;; ...
  (:action drive
    :parameters (?r - robot ?from ?to - place)
    :precondition (and (at ?r ?from) (path ?from ?to))
    :effect (probabilistic
      9/10 (and (at ?r ?to) (not (at ?r ?from))))))
(define (problem unreliable-robot)
  (:domain unreliable-robot-domain)
  (:objects kitchen hall office - place shakey - robot)
  (:init (at shakey kitchen) (path kitchen hall) (path hall kitchen)
    (path hall office) (path office hall))
  (:goal (at office)))
```

Figure 1: Part of the PPDDL description of a simple problem that we will use to illustrate the structure of ASNets. This is a toy navigation domain where a robot, *shakey*, is tasked with moving from place to place in a building using movement actions of the form drive($shakey, ?from, ?to$). When invoked, a drive action moves the robot from its initial position to its destination successfully with 90% probability, and does nothing the remaining 10% of the time. PPDDL actions combine the add and delete lists into a single `:effect` declaration; here the add list includes at($?r, ?to$), while `(not ...)` indicates that at($?r, ?from$) belongs on the delete list. In the specific problem depicted above, the robot must move from the *kitchen*—as specified in the initial state declaration, `(:init ...)`—to the *office*—which satisfies the goal formula, `(:goal ...)`.

grounding code from MDPSim (Younes, Littman, Weissman, & Asmuth, 2005), which was introduced for use in IPC-4. MDPSim's grounding code supports typed parameters for action schemas, and also includes some basic optimisations to avoid instantiating propositions that can never be made true, or actions that can never be enabled. The running example in Section 3 assumes the use of a grounding algorithm with similar optimisations.

3. Action Schema Networks

In this section, we will describe and extend Action Schema Networks (ASNets), which were introduced in past work by Toyer et al. (2018). The approximate structure of an ASNet is illustrated in Figure 2. An ASNet transforms a feature representation of the current state s into a policy $\pi^\theta(a|s)$ via an alternating sequence of *action layers* and *proposition layers*. Each action layer consists of a single *action module* (Section 3.2) per action. An action module takes a vector of features from proposition modules in the previous layer and outputs a new vector of features which the network can use to capture some relevant fact about the problem. Similarly, each proposition layer consists of a *proposition module* (Section 3.3) for each proposition. Each such module takes a vector of input features from action modules in the previous layer, and produces a new vector of features. Proposition

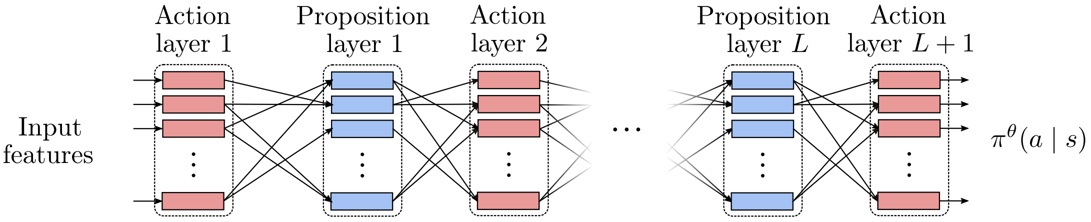


Figure 2: High-level overview of an ASNet. Each coloured rectangle represents an action module (red) or a proposition module (blue); these modules apply learnt transformations to input feature vectors in order to produce more expressive output feature vectors. Information flows from the input (left) to the output (right) along the black lines connecting modules in successive layers. For the sake of visual clarity, skip connections (described in the main text) are not depicted. Modules are grouped into L proposition layers and $L+1$ action layers. Throughout, we refer to such a network as an "L-layer ASNet".

modules in one layer are connected to action modules in the next layer according to a certain notion of *relatedness* of actions and propositions (Section 3.1). This connectivity scheme enables modules to share weights in such a way that the size and shape of learnt weights is the same for *all* ASNets from a given domain, even if the ASNets correspond to problems of different sizes.[1] As a result, a policy represented by an ASNet can be applied to any problem from a given domain.

3.1 Relatedness

The structure of an ASNet is determined by the *relatedness* of actions and propositions in the corresponding PPDDL problem. To define relatedness, we first need to define the notion of a *lifted proposition*: in the context of a specific PPDDL action schema, a lifted proposition is a specific combination of action parameters applied to a predicate. In the $\mathrm{drive}(?r, ?\mathit{from}, ?\mathit{to})$ action schema from the `unreliable-robot` problem in Figure 1, we say that $\mathrm{at}(?r, ?\mathit{from})$ is the first unique lifted proposition, $\mathrm{path}(?\mathit{from}, ?\mathit{to})$ is the second unique lifted proposition, and $\mathrm{at}(?r, ?\mathit{to})$ is the third unique lifted proposition. This notion allows us to define relatedness: a ground action a and ground proposition p are related at position k—as denoted by the predicate $R(a, p, k)$—if p corresponds to the kth unique lifted proposition in the action schema from which a was instantiated. As a result, an action and proposition are related at some position when the proposition appears in one of the action's preconditions or effects. We use this notion to connect action and proposition modules in adjacent layers, as described in Section 3.2 and Section 3.3. Note that this definition of "relatedness at position k" is distinct from the notion of relatedness used by Toyer et al. (2018), which did not draw a distinction between propositions in different positions that

[1] A note on terminology: throughout this paper, we use "an ASNet" to refer to a network instantiated for a specific problem instance ζ from a domain $\mathcal{D}$. A specific ASNet is only capable of selecting actions for the corresponding instance ζ, but its *weights* will be transferable to an ASNet for any other problem ζ' in the same domain $\mathcal{D}$.

were instantiated from the same predicate. We will see in Section 3.3 that the concept of positions leads to a slightly more expressive form of pooling at the inputs to the proposition modules.

To see how our notion of relatedness applies to actual ground actions, consider the drive(*shakey*, *kitchen*, *hall*) action produced by grounding the `unreliable-robot` problem. This action can be executed when at(*shakey*, *kitchen*) $\wedge$ path(*kitchen*, *hall*), in which case it has the effect of making at(*shakey*, *hall*) true and at(*shakey*, *kitchen*) false with probability 90%, or doing nothing otherwise. Hence, drive(*shakey*, *kitchen*, *hall*) is related to at(*shakey*, *kitchen*) at position $k = 1$, path(*kitchen*, *hall*) at position $k = 2$, and at(*shakey*, *hall*) at position $k = 3$. Conversely, the set of actions related to at(*shakey*, *kitchen*) will include drive(*shakey*, *kitchen*, *hall*) at position 1 and drive(*shakey*, *hall*, *kitchen*) at position 3. No other drive actions take *shakey* to or from the *kitchen*. It's worth reiterating that these position numbers reflect the order in which propositions first appear in the action schema of Figure 1. Although at(*shakey*, *kitchen*) appears twice in the action definition (once in the preconditions and once in the effects), we say that it only occurs at one unique position because we do not double-count propositions to the same lifted proposition in the action schema. Hypothetically, if we had a drive(*shakey*, *kitchen*, *kitchen*) action, then it *would* be related to at(*shakey*, *kitchen*) at two positions ($k = 1$ and $k = 3$), because two of the ground proposition's appearances in the ground action correspond to lifted propositions with different parameters in the action schema.

The architecture of an ASNet only depends on aspects of a PPDDL domain and problem that affect the corresponding relatedness graph. Observe that in the `unreliable-robot` problem, the relatedness of actions and propositions is only a very coarse encoding of the semantics of those actions and propositions. For instance, the fact that a drive action can only change the propositions that appear in its (probabilistic) effect 90% of the time did not change the relatedness graph. Likewise, whether a proposition appears negated or unnegated in the precondition of an action does not affect is relatedness to that action. This is why it is straightforward for the ASNets architecture to "support" so many of the PPDDL features discussed in Section 2: most of them do not affect relatedness, and are consequently irrelevant to the high-level structure of the network. The missing semantics are of course important to choosing actions at execution time, but the logic for making these decisions is captured by the *weights* of an ASNet during training, rather than being directly encoded in the architecture of the network.

3.2 Action Layers

Consider an ASNet with $L + 1$ action layers, numbered $l = 1, \ldots, L + 1$. In this section, we will examine the structure of the $L - 1$ intermediate layers $2, \ldots, L$; the structure of first (input) layer and final $L + 1$ (output) layer is slightly different, and will be deferred to Section 3.4. The lth action layer is composed of an action module for each action $a \in \mathcal{A}$. Each such module takes as input some vector $u_a^l \in \mathbb{R}^{d_a^l}$ and produces as output another vector $\phi_a^l \in \mathbb{R}^{d_h}$. We refer to the output size d_h as the *hidden dimension* of the network. To construct the input u_a^l to the action module, we first enumerate all related propositions $p_1, \ldots, p_M$, then concatenate the corresponding hidden representations $\psi_{p_1}^{l-1}, \ldots, \psi_{p_M}^{l-1}$ from the preceding hidden layer of the network. Each of these M inputs will themselves be

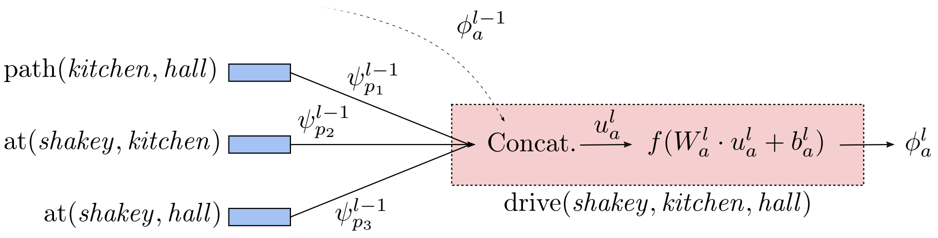


Figure 3: Schematic of an action module for the `unreliable-robot` problem (Figure 1), including skip connection at top. The propositions related to drive(*shakey*, *kitchen*, *hall*) are path(*kitchen*, *hall*), at(*shakey*, *kitchen*) and at(*shakey*, *hall*). Thus, the corresponding proposition module representations are concatenated together and then joined with the previous action representation ϕ_a^{l-1} to produce an input vector u_a^l. u_a^l is then passed through a learnt affine transform $W_a^l u_a^l + b_a^l$ and fixed nonlinearity $f(\cdot)$ to produce an output representation ϕ_a^l.

d_h-dimensional, for a total input size of $u_a^l = M \cdot d_h$. In later layers, we also include a *skip connection* that feeds the representation ϕ_a^{l-1} for action a from the previous action layer into the action module for the current layer. Toyer et al. (2018) did not include skip connections; we have introduced them to make it easier for the network to propagate information across many layers. The input vector u_a^l for the l-th layer action module for a is thus the concatenation of $M + 1$ earlier output vectors:

$$u_a^l = \begin{bmatrix} \psi_{p_1}^{l-1} \\ \vdots \\ \psi_{p_M}^{l-1} \\ \phi_a^{l-1} \end{bmatrix} . \tag{1}$$

An output is computed from the input via $\phi_a^l = f\left(W_a^l u_a^l + b_a^l\right)$, where $f(\cdot)$ is some fixed nonlinearity and $W_a^l \in \mathbb{R}^{d_h \cdot d_a^l}, b_a^l \in \mathbb{R}^{d_h}$ are learnt parameters. An example of such an action module for the `unreliable-robot` problem is shown in Figure 3.

Crucially, the action modules in a given layer are constructed in such a way as to enable weight sharing, which (as we will see) ultimately allows ASNets to apply the same set of learnt weights to any problem in a PPDDL domain. Consider an action $a \in \mathcal{A}$ and its corresponding action schema $\text{op}(a) \in \mathbb{A}$. We can enumerate the propositions related to a by first listing the predicates in the precondition and effect of $\text{op}(a)$, ignoring "duplicate" predicates that appear twice with the same parameters. Next, we ground those predicates by binding their arguments to the same objects used to instantiate a from $\text{op}(a)$, thereby yielding all propositions $p_1, \ldots, p_K$ related to a through positions $k = 1, 2, \ldots, K$. Notice that if we repeat this procedure for some $b \in \mathcal{A}$ with the same schema (so that $\text{op}(b) = \text{op}(a)$), then we will obtain another equal-length list of propositions $q_1, \ldots, q_K$ related to b.[2] Although $q_1, \ldots, q_K$ may be distinct from $p_1, \ldots, p_K$, the propositions still have a semantic

[2] This assumes there are no quantifiers in the domain description, as noted in Section 2.

correspondence: if we always enumerate the predicates in $\mathrm{op}(a)$ in a consistent order, then it will always be the case that q_j and p_j are instantiated from the same predicate, and that they perform a similar “role” in $\mathrm{op}(a)$. Further, the equal length of the proposition lists means that $d_a^l = d_b^l$. Hence, we can tie the weights for the modules for a and b in layer l so that $W_a^l = W_b^l$ and $b_a^l = b_b^l$, and do likewise for all other modules that share the same action schema.

Our weight sharing scheme forces the modules to learn a generic transformation which can be applied by a module for *any* action instantiated from a given schema. This allows us to generalise across problems: because the number and structure of weights depends only on the action schema, we can re-use the same set of learnt weights for any problem in a domain. Our weight sharing scheme is reminiscent of the way that convolutional neural networks learn filters which can be applied to an image at any location. The filter sharing employed by convolutional neural networks improves data efficiency and introduces useful invariances (e.g. translation invariance) (LeCun, Bengio, et al., 1995), and we expect similar benefits from weight sharing in ASNets.

3.3 Proposition Layers

Proposition layers operate in an analogous manner to action layers. An L-layer ASNet contains L proposition layers numbered $l = 1, \ldots, L$. For each proposition $p \in \mathcal{P}$, the lth proposition layer contains a corresponding proposition module which turns some input $v_p^l \in \mathbb{R}^{d_p^l}$ into a new hidden representation $\psi_p^l \in \mathbb{R}^{d_h}$, where d_p^l is the input dimension of the hidden module. Again, the input v_p^l and output ψ_p^l are related via a transformation $\psi_p^l = f\left(W_p^l v_p^l + b_p^l\right)$, for some fixed nonlinearity $f(\cdot)$ and learnt weights $W_p^l \in \mathbb{R}^{d_h \cdot d_p^l}, b_p^l \in \mathbb{R}^{d_h}$. The main difference between action and proposition modules lies in the way that the input v_p^l is constructed, which we consider in detail below.

The need for a different mechanism for computing inputs to proposition modules arises from the fact that two propositions instantiated from the same predicate may be related to a *different* number of actions. As an example, consider the at(*shakey*, *hall*) proposition in the `unreliable-robot` problem (Figure 1). *shakey* can travel to or from the *hall* via the *kitchen* or *office*. Hence, the four actions related to at(*shakey*, *hall*) will be drive(*shakey*, *hall*, *kitchen*) and drive(*shakey*, *hall*, *office*) at position 1, as well as drive(*shakey*, *kitchen*, *hall*) and drive(*shakey*, *office*, *hall*) at position 2. On the other hand, there is only one path leading to and from the *kitchen*, which goes straight from the *hall*, and so there will only be two actions related to the proposition at(*shakey*, *kitchen*). It will not suffice to construct the input for an at module by simply concatenating the representations for all related actions. If we did so then the inputs for different at modules would be of different sizes, and we could not share weights between them.

Instead of constructing the input v_p^l to the l-th layer module for proposition p using concatenation, we choose to pool over related actions. First, we enumerate all action schemas $A_1, \ldots, A_S$ which refer to the predicate $\mathrm{pred}(p)$ in a precondition or effect through positions $k_1, \ldots, k_S$. Clearly, any action related to p must be instantiated from one of these schemas. Further, an action schema may appear more than once in this list if it is related to p through

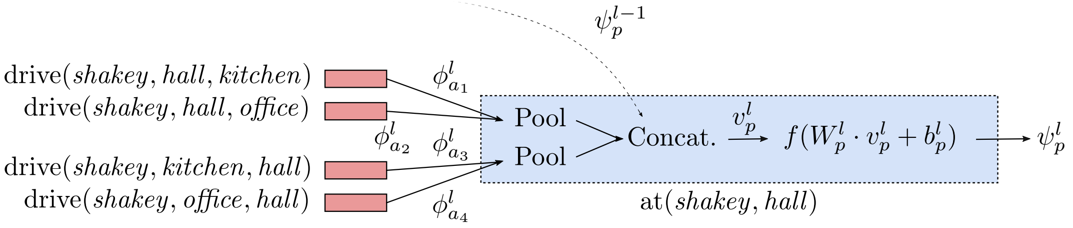

ψ_p^{l-1}
drive(*shakey*, *hall*, *kitchen*)
drive(*shakey*, *hall*, *office*)
drive(*shakey*, *kitchen*, *hall*)
drive(*shakey*, *office*, *hall*)
$\phi_{a_1}^l$
$\phi_{a_2}^l$
$\phi_{a_3}^l$
$\phi_{a_4}^l$
Pool
Pool
Concat.
v_p^l
$f(W_p^l \cdot v_p^l + b_p^l)$
ψ_p^l
at(*shakey*, *hall*)

Figure 4: A proposition module for the `unreliable-robot` problem (Figure 1), again including a skip connection Here there are four drive actions related to at(*shakey*, *kitchen*), including two in position $k = 1$ (top two), and two in position $k = 2$ (bottom two). Related actions in the same position are pooled together using two separate pooling operations. The pooled representations are then concatenated together with the output ψ_p^{l-1} from the corresponding module in the previous proposition layer (dashed line) to produce an input v_p^l. Finally, the input is passed through the learnt transformation $\psi_p^l = f(W_p^l v_p^l + b_p^l)$.

more than one position. From the action schema list, we construct the input v_p^l using

$$v_p^l = \begin{bmatrix} \text{pool}(\{\phi_a^l \mid \text{op}(a) = A_1 \wedge R(a, p, k_1)\}) \\ \vdots \\ \text{pool}(\{\phi_a^l \mid \text{op}(a) = A_S \wedge R(a, p, k_S)\}) \end{bmatrix} , \qquad (2)$$

where $\text{pool}(\{x_1, \ldots, x_R\}) \in \mathbb{R}^{d_h}$ is a function that takes an arbitrary-size set of vectors $x_1, \ldots, x_R \in \mathbb{R}^{d_h}$ and aggregates their elements into a single vector. In later proposition layers, we also introduce skip connections between successive proposition layers, so that v_p^l will also include the previous-layer output representation ψ_p^{l-1} for proposition p. An example proposition module is illustrated in Figure 4.

There are many possible implementations of the pooling function $\text{pool}(\cdot)$. One could perform pooling by averaging corresponding elements (mean pooling), taking an elementwise maximum (max pooling), and so on. In principle, this pooling process could destroy information that would be relevant to expressing a generalised policy. In practice, we have found that a simple max pooling strategy is sufficient to solve a range of interesting problems.

In the original ASNets paper (Toyer et al., 2018), the proposition modules included only a single pooling operation for each related action schema, rather than a separate pooling operation for every related action schema and position. Thus, inputs corresponding to actions related to a proposition through different positions could not be distinguished by the relevant proposition module. To see how this might cause problems, imagine an `unreliable-robot` problem with only two locations: *office* and *hall*. In such a problem, the set of drive actions related to at(*shakey*, *office*) would be the same as the set of actions related to at(*shakey*, *hall*), so both at(*shakey*, $\cdot$) proposition modules would share the same representation! Intuitively, the old pooling scheme could not easily tell whether *shakey* was leaving a room or entering it. The pooling scheme here is more expressive, as it enables a proposition module to easily distinguish between inputs for related actions in which the proposition plays a different "role", such as appearing in a precondition for one action, and an effect for a different action.

After pooling, the input dimension of a proposition module, $d_p^l = S \cdot d_h$, is the same for all proposition modules instantiated from predicate $\operatorname{pred}(p)$, so we can again use weight sharing. Specifically, we define W_p^l to equal W_q^l in the lth proposition layer whenever $\operatorname{pred}(p) = \operatorname{pred}(q)$. As with weight sharing in action modules, this allows us to generalise over different problems drawn from the same domain. In particular, the complete set of weights for any problem in a given domain will be

$$\theta = \{W_a^l, b_a^l \mid 1 \leq l \leq L+1, a \in \mathbb{A}\} \cup \{W_p^l, b_p^l \mid 1 \leq l \leq L, p \in \mathbb{P}\} , \tag{3}$$

where we have abused notation slightly by using a and p to refer to action schemas and predicates instead of actions and propositions. It is thus possible to learn a generalised policy by acquiring a fixed set of weights θ using some small training tasks, and then transferring them to much larger problems.

3.4 Input & Output Action Layers

The first action layer of an ASNet is the input layer of the network, and thus has a different input scheme to later layers. The input to a first-layer module for a given action a is composed of proposition truth values, binary goal indicators, and a binary indicator to show whether the action is applicable in the current state. To make this concrete, consider the propositions $p_1, \ldots, p_M$ which are related to a through each position $k_1, \ldots, k_M$. We define the truth value vector $v \in \{0,1\}^M$ to have $v_j = 1$ if p_j is true in the current state and $v_j = 0$ otherwise, and define the goal information vector $g \in \{0,1\}^M$ to have $g_j = 1$ if a proposition appears unnegated in the goal and $g_j = 0$ otherwise.[3] Further, define $m = 1$ if a is applicable in the current state, and $m = 0$ otherwise. The input to the first-layer action module corresponding to a is

$$u_a^1 = \begin{bmatrix} v \\ g \\ m \end{bmatrix} . \tag{4}$$

In Section 3.5 we further extend this input representation to use information derived from heuristics evaluated at the current state, which our experimental evaluation shows is critical to allowing ASNets to solve problems in some domains.

The final action layer of an ASNet is also slightly different to the preceding ones. At the final layer, we would like an ASNet to give us a probability $\pi^\theta(a_i|s)$ that a given action a_i is the correct action to take in a given state s. Hence, we stipulate that the module for a given action a_i in the final $(L+1)$th action layer of an ASNet should produce a single logit $\phi_{a_i}^{L+1} \in \mathbb{R}$. The probability that a_i should be selected in s is thus proportional to $\exp \phi_{a_i}^{L+1}$. To normalise these probabilities, and to ensure that only applicable actions can be selected, we then pass the logits through a masked softmax activation. Let $m_1, \ldots, m_N \in \{0,1\}$ be binary indicators of whether each action a_i is applicable (1) or not (0), and let

[3] As noted earlier in Section 2, this representation limits us to goals which are conjunctions of positive literals.

$\phi_{a_1}^{L+1}, \ldots, \phi_{a_N}^{L+1} \in \mathbb{R}$ be the unscaled log probabilities produced by the action modules in the final action layer. The scaled output of the ASNet is then

$$\pi^\theta(a_i|s) = \frac{m_i \exp\left(\phi_{a_i}^{L+1}\right)}{\sum_{j=1}^N m_j \exp\left(\phi_{a_j}^{L+1}\right)} \ . \tag{5}$$

Masking out disabled actions ensures that the ASNet is only trained to distinguish between actions relevant to a given state. The non-temporal variant of the earlier FPG planner (Buffet & Aberdeen, 2009) uses the same strategy.

3.5 Heuristic Inputs and the Receptive Field

The literature on traditional convolutional neural networks sometimes discusses the "receptive field" (or "effective receptive field") of an activation in a certain layer of the network (Long et al., 2015). This refers to the region of the input image that is able to influence the output of that activation. Because convolutional neural networks are only locally connected at each layer, it can take many layers to propagate information about the input image to different regions of the network itself, and so the receptive field of an activation in one of the earlier layers of a network will often only be a small sub-window of the input image. The activations in an ASNet suffer from a similar limitation. As a result, the maximum length of chains of actions and propositions that the network can reason about is limited by its depth, since each action or proposition layer only propagates information one step along a given chain of related actions and propositions.

To make the receptive field limitation more concrete, consider a restricted class of problems from the `unreliable-robot` domain of Figure 1. In this restricted class of problems, the robot starts at some location m, and can move along one of two unidirectional chains: a left chain $m \to l_1 \to l_2 \to \cdots \to l_K$ and a right chain $m \to r_1 \to r_2 \to \cdots \to r_K$. If the goal is always either moving to r_K or moving to l_K, then the main challenge faced by the agent will be deciding whether to move to r_1 or l_1 from the initial location m. After it has reached r_1 or l_1, it will have no choice but to follow the corresponding chain to the end. The ASNet will be given information about which chain contains the goal via the g vector passed to the drive(*shakey*, l_{K-1}, l_K) and drive(*shakey*, r_{K-1}, r_K) action modules in the first layer. Those modules correspond to actions at the end of either chain, which are initially disabled. Instead, it is the action module for drive(*shakey*, m, l_1) in the last layer which governs whether or not the agent chooses to move from m to l_1 in the initial state. If we ignore skip connections, then this module is connected to the module for drive(*shakey*, l_{K-1}, l_K) in the input layer via a related action and proposition chain of the form

$$\begin{aligned} &R(\text{drive}(\textit{shakey}, m, l_1), \text{at}(\textit{shakey}, l_1), 3) \\ &\quad \to R(\text{drive}(\textit{shakey}, l_1, l_2), \text{at}(\textit{shakey}, l_1), 1) \\ &\quad \to R(\text{drive}(\textit{shakey}, l_1, l_2), \text{at}(\textit{shakey}, l_2), 3) \\ &\quad \to \cdots \\ &\quad \to R(\text{drive}(\textit{shakey}, l_{K-1}, l_K), \text{at}(\textit{shakey}, l_{K-1}), 1) \ , \end{aligned}$$

and likewise for the right chain. Unfortunately, the length of this relatedness chain depends on the size K of the problem. If a network has fewer than K proposition layers and $K + 1$

action layers then it will be impossible to communicate information about the goal back from drive(*shakey*, r_{K-1}, r_K) to drive(*shakey*, m, l_1) and drive(*shakey*, m, r_1). It is thus impossible for any fixed-depth ASNet to obtain a generalised policy for this entire subclass of arbitrary-depth problems, as we show experimentally in Appendix C.

To overcome the receptive field limitation, we supply each action module in the first layer of an ASNet with two kinds of "heuristic inputs". First, we include features derived from the landmarks identified by the LM-cut heuristic in each state (Helmert & Domshlak, 2009). Second, we add features counting how many times each action has been taken over the current trajectory.

We will begin by describing the LM-cut features. LM-cut derives a lower bound on the cost-to-go for a problem by identifying a set of disjunctive action landmarks for a delete-relaxed, and possibly determinised, version of that problem.[4] Each disjunctive action landmark is a set of actions where at least one action must appear along any path to the goal. By supplying information about the landmarks recovered by LM-cut directly to the network, we can improve its ability to reason about which actions will have helpful long-term consequences. Specifically, for each action a, we create a new indicator vector $c \in \{0,1\}^3$ to use as an auxiliary input to the corresponding first-layer action module. We have $c_1 = 1$ iff a appears as the only action in at least one landmark; $c_2 = 1$ iff a appears in a landmark containing two or more actions; and $c_3 = 1$ iff a does not appear in any landmark. These values are concatenated to the input u_a^1 for the first-layer action module for action a.

In addition to features derived from LM-cut landmarks, we also include a count c_a of the number of times each action a has been executed over the course of the current trajectory. c_a is concatenated to the first-layer action module input u_a^1. We found that this information was useful in domains where the ASNet was unable to distinguish some states from each other even with the help of heuristic information, and would sometimes end up in loops where it would repeatedly switch between two adjacent states.

We note that LM-cut landmarks and action counts are not the only form of heuristic information that could serve to lift the receptive field limitation of ASNets. For example, one could instead feed ASNets information about helpful actions computed by the FF planner (Hoffmann, 2001), as done in past work (de la Rosa et al., 2011). In the probabilistic setting, it may be more appropriate to supply the ASNet with operator counts produced by a probability-aware heuristic like h^{roc} or h^{pom} (Trevizan, Thiébaux, & Haslum, 2017). In some domains it is likely possible to remove the need for heuristic information entirely by augmenting ASNets with some combination of recurrent modules (Hochreiter & Schmidhuber, 1997), attention (Vinyals, Fortunato, & Jaitly, 2015), and memory (Weston, Chopra, & Bordes, 2014). Of course, the downside of a more powerful architecture is that it would weaken the (very strong) inductive bias inherent in ASNets, as well as increase the total computational cost and parameter count of the network. All three of these consequences would in turn lead to increased training times and a higher chance of overfitting to the training set. We leave it to future work to experiment with alternative heuristics and determine the optimal tradeoff between model expressiveness and inductive bias in a planning context.

[4] The *delete relaxation* of a planning problem is one in which all "negative" effects (those which make a proposition false) are removed from each action. A *determinisation* of an SSP is a deterministic planning problem in which each action with stochastic effects is mapped to one or more (inequivalent) actions with only deterministic effects. These ideas are explained further by Mausam and Kolobov (2012).

The receptive field limitation of ASNets is reminiscent of the behaviour of short-sighted probabilistic planners (Trevizan & Veloso, 2012) or receding-horizon Model-Predictive Control (MPC) strategies (Camacho & Alba, 2007), which both choose actions by planning over short lookahead windows. However, the horizon for short-sighted planners is defined in terms of the states reachable in some fixed number of steps, whereas the receptive field of an ASNet is defined by the relatedness of state variables and actions. In Section 5 we compare ASNets against a short-sighted planner on several probabilistic planning benchmarks, and demonstrate that ASNets can easily solve problems that the short-sighted planner cannot solve in a reasonable period of time.

4. Training and Exploiting Generalised Policies

This section explains our mechanism for training and exploiting ASNet-based policies. We note that ASNets could easily be trained in other ways with a variety of different trade-offs. The focus here is on *simple* training and exploitation methods to directly evaluate the quality of ASNets as a class of models, as opposed to evaluating entire learning-based planning systems in which ASNets only play a small part. Readers may refer to the related work in Section 7 for a thorough survey of those other mechanisms for learning ML-based policies or control knowledge (Section 7.1.2) and exploiting such knowledge (Section 7.1.3).

4.1 Training via Imitation Learning

The training procedure for obtaining an ASNet-based generalised policy is depicted in Algorithm 1. The high-level aim of the training procedure is to optimise a set of ASNet weights θ so that the corresponding ASNets mimic the actions selected by a heuristic search planner—which we call the *teacher planner*—on a collection P_{train} of training problems from a given domain. These problems are assumed to be small enough to quickly solve via heuristic search, while also containing structural elements representative of those found in larger problems. Learning with the training set P_{train} proceeds over a series of *epochs*, as depicted in ASNet-Train (Algorithm 1). Throughout the epochs, the algorithm maintains a list $\mathcal{M}$ of encountered states (initially empty), and a current estimate θ of the ASNet weights (initialised randomly). Each epoch is in turn divided into an *exploration phase* and a *training phase*, corresponding to the two outer loops in Train-Epoch. We will now describe each of those phases separately.

In the exploration phase, Train-Epoch uses the current ASNet parameters θ to sample T_{explore} trajectories on each problem $\zeta \in P_{\text{train}}$. The code for sampling a trajectory is shown in Run-Policy: the ASNet starts in state s_0 and must produce an action a_t distributed according to $\pi^\theta(a|s)$ until a goal state or other terminal state is reached, or the trajectory length limit is exceeded (typically based on the dead end penalty D). After obtaining states $s_0, \ldots, s_N$ from a policy rollout and adding them to the state memory $\mathcal{M}$, we also extend the state memory with a series of rollouts under the teacher planner's policy, including a separate rollout starting from each of $s_1, \ldots, s_N$. Including ASNet rollout states in $\mathcal{M}$ ensures that we continue to optimise θ to yield good action choices in the states that our ASNets visit more often. On the other hand, including states from teacher policy rollouts ensures that $\mathcal{M}$ always contains some goal trajectories, and so θ can be optimised to perform well on states close to the goal even before the ASNet has trained for long enough to reach those

Algorithm 1 Learning ASNet weights θ from a set of training problems P_{train}.

```
procedure ASNet-Train
    $\mathcal{M} \leftarrow$ empty list; $\theta \leftarrow$ Random-Initial-Weights(); $n \leftarrow 0$
    repeat
        Train-Epoch($\theta, \mathcal{M}$)
        $n \leftarrow n + 1$
    until $n > T_{\text{max-epochs}}$ or early stopping condition satisfied
procedure Train-Epoch($\theta$, $\mathcal{M}$)
    for $i = 1, \ldots, T_{\text{explore}}$ do
        for all $\zeta \in P_{\text{train}}$ do
            $s_0, \ldots, s_N \leftarrow$ Run-Policy($s_0(\zeta), \pi^\theta$)
            $\mathcal{M}$.extend($\{s_0, \ldots, s_N\}$)
            for $j = 0, \ldots, N$ do
                $s'_j \ldots, s'_M \leftarrow$ Teacher-Rollout($s_j$)
                $\mathcal{M}$.extend($\{s'_j \ldots, s'_M\}$)
    for $i = 1, \ldots, T_{\text{train}}$ do
        $\mathcal{B} \leftarrow$ Sample-Minibatch($\mathcal{M}$)
        Update $\theta$ using $\frac{d\mathcal{L}_\theta(\mathcal{B})}{d\theta}$ (Equation (6))
function Run-Policy($s$, $\pi$)
    $t \leftarrow 0$; $s_t \leftarrow s$
    while $s \notin \mathcal{G} \wedge t < T_{\text{trajectory-limit}}$ do
        $a_t \sim \pi(a_t \mid s_t)$
        $s_{t+1} \sim \mathcal{T}(s_{t+1} \mid s_t, a_t)$
        $t \leftarrow t + 1$
    return $s_0, \ldots, s_t$
```

states itself. The use of a mixture of states generated by π^θ and states generated by the teacher is reminiscent of the way that DAgger imitation learning algorithm (Ross, Gordon, & Bagnell, 2011) interpolates between expert and novice policies when collecting a training dataset. Our use of highly non-convex neural networks means that we cannot translate over the no-regret guarantees of DAgger to this setting. However, it's probable that our use of a similar strategy makes it less likely that an ASNet will go "off-distribution" and encounter a state for which it cannot select a good action at test time.

After extending the state memory $\mathcal{M}$, ASNets enters a learning phase in which it updates the weights θ. The learning phase depends on action labels calculated using the teacher planner during the exploration phase. In particular, when we add a state s to state memory in the exploration phase, we also invoke the teacher planner to obtain a Q-value $Q^{\text{teach}}(s, a)$ with respect to the teacher planner's policy for each enabled action a in s. This allows us to label actions as "optimal" or "sub-optimal" with respect to the teacher's value function: action a in state s is given a label $y_{s,a} \in \{0, 1\}$ that is set to one if

$$Q^{\text{teach}}(s, a) = \max_{a'} Q^{\text{teach}}(s, a')$$

and zero otherwise. During the learning phase of each epoch, we repeatedly sample a fixed-size minibatch of states $\mathcal{B}$ from state memory $\mathcal{M}$, then optimise θ to increase the probability of selecting actions with $y_{s,a} = 1$. Specifically, given a minibatch $\mathcal{B}$, the batch objective for an ASNet is to maximise the cross-entropy-based loss

$$\mathcal{L}_\theta(\mathcal{B}) = \sum_{s \in \mathcal{B}} \sum_{a \in A} \Big[(1 - y_{s,a}) \cdot \log(1 - \pi^\theta(a|s)) + y_{s,a} \log \pi^\theta(a|s) \Big] + \frac{1}{2}\lambda \|\theta\|^2 \ . \tag{6}$$

The last term is an ℓ_2 regulariser (with constant coefficient $\lambda > 0$) that ensures $\mathcal{L}_\theta$ is always bounded below as a function of θ; otherwise it is possible to drive $L_\theta(\mathcal{B})$ to $-\infty$ if the data in $\mathcal{B}$ is linearly separable. We update θ at the end of a learning step by feeding the gradient $\partial \mathcal{L}_\theta / \partial \theta$ into any appropriate first-order optimiser. This process of sampling a minibatch and updating θ is repeated T_{train} times in each learning phase. If the ASNet has sufficient expressive power to imitate the teacher, then the parameter updates should ultimately make it follow similar trajectories to the teacher on the training problems.

The training process typically terminates after a fixed number of epochs or maximum amount of time has elapsed. However, for domains that are relatively easy for ASNets to solve (e.g. Triangle Tireworld), we have found that early termination conditions can sometimes decrease the time required for training. Specifically, we terminate early if the success rate of the ASNet on the training problems has been at least p_{solved} for at least T_{stop} consecutive epochs. In experiments, we use the same values of $T_{\text{stop}} = 20$ and $p_{\text{solved}} = 1$ for all domains.

For domains where our chosen training problems varied widely in difficulty, we found that Algorithm 1 would sometimes spend most of the training period running the teacher planner in order to perform a Teacher-Rollout$(\cdot)$ starting from each visited state. To avoid this problem, we made three modifications to Algorithm 1. First, we cached the results of calls to the planner so that it would only have to be invoked once for each encountered state. Second, in the first epoch of training, we skipped rolling out the ASNet policy, and instead simply extended the memory $\mathcal{M}$ with Teacher-Rollout$(s_0(\zeta))$ for the initial state $s_0(\zeta)$ of each problem $\zeta \in P_{\text{train}}$. This meant that in the second epoch of training, the ASNet was already following a moderately effective, low-entropy policy, and thus encountered fewer unique states. Combined with caching, this led to fewer planner calls, and thus limited the impact of very difficult problems in the training set. Third, we put a timeout of 10s on the teacher planner. If the teacher planner did not succeed in finding a plan or policy starting from a given state within 10s, then the state was omitted from the state memory $\mathcal{M}$ and instead recorded elsewhere so that the planner would not be invoked on the same state again. Together, these changes substantially decreased the cost of planning during the training period.

In addition to supervised learning, we also tried training ASNets with Policy Gradient Reinforcement Learning (PG RL) using a similar strategy to the Factored Policy Gradient (FPG) planner (Buffet & Aberdeen, 2009). Reinforcement learning has the advantage of enabling us to directly minimise the cost of trajectories produced by our ASNet policy on the problems in P_{train}.[5] In contrast, optimising the supervised objective in Equation (6)

[5] Recall that in the fSSPUDE framework, the cost of trajectories that fail to reach the goal are set to a high constant D; hence, minimising the cost of trajectories is typically sufficient to obtain a high probability of reaching the goal, too.

may not lead to a good ASNet policy if the teacher planner's implicit policy is outside of the ASNet's hypothesis space. Unfortunately, we found that basic PG RL (in our case: REINFORCE with a state-dependent baseline) was simply too inefficient to train ASNet-based policies in any reasonable amount of time. We leave exploration of more efficient reinforcement learning strategies to future work.

4.2 Exploitation

We exploit our ASNet-based generalised policy directly, by repeatedly picking an action

$$a_t \in \operatorname*{arg\,max}_{a \in A} \pi^\theta(a|s_t)$$

in state s_t (breaking ties arbitrarily), then sampling a successor state s_{t+1} from the transition distribution

$$s_{t+1} \sim \mathcal{T}(s'|s_t, a_t)$$

until a goal is reached. It would be equally easy to sample an action directly from the output distribution; that is, replacing the arg max above with

$$a_t \sim \pi^\theta(a|s_t) \ .$$

A sampling strategy might be preferable to a direct maximisation strategy on problems where ASNets' learnt control knowledge fails to perfectly solve the problem. In problems without many dead ends, a degree of randomness during evaluation is sometimes sufficient to push an ASNet out of regions of state space where sampling the "best" action could lead to loops. We compares these strategies empirically in Section 5.

We note that it is also possible to use ASNets to guide a heuristic search planner, instead of relying on an ASNet-based policy to solve all problems in a domain directly. In the probabilistic planning setting, one particularly promising approach is to incorporate ASNets into Monte Carlo tree search algorithms like UCT, in the style of AlphaGo (Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, et al., 2016). A recent paper has made a preliminary evaluation of various mechanisms for guiding UCT with ASNets, including the use of ASNet-based generalised policies as rollout policies, and the use of ASNets to bias UCB1 successor selection. These strategies can alleviate the negative impact of inadequate ASNet training, and help solve problems that would be too complex for ASNets to solve on their own (Shen et al., 2019). Of course, UCT and other algorithms that can make use of learnt search control knowledge are generally agnostic to the type of learnt model that they are used with. Thus, it is equally possible to plug ASNets into most existing learning-guided combinatorial search algorithms, which we survey in Section 7.1.3. In order to disentangle the effect of model expressiveness from the quality of heuristic search, our evaluation in the next section will eschew these search-based algorithms in favour of the direct execution approach described previously.

5. Experimental Evaluation

In this section, we empirically evaluate the performance of ASNets on a range of probabilistic and deterministic domains, identify which elements of ASNets contribute the most to performance, and present an extended evaluation on deterministic Blocksworld. Code for our experiments is available on GitHub.[6]

5.1 Time-Based Evaluation

In practice, we envisage that ASNet-style generalised policies will be most useful for solving problems that are too large for heuristic search, but where there also exists some simple domain-specific trick that makes the problem easy to solve. For instance, in the venerable Blocksworld domain (Slaney & Thiébaux, 2001), it's known that optimal planning is NP-hard, but that merely finding a "reasonable" plan can be accomplished in linear time with a domain-specific algorithm. For the user of a planning system, the key question is whether the high fixed cost of training ASNets on a set of small problems from a domain is justified by the time saved when one uses ASNets in place of heuristic search on larger problems. We answer this question for seven probabilistic and deterministic domains by comparing the number and size of problems that ASNets can solve in a given amount of time against the number and size of problems that can be solved by a range of competitive baseline planners.

5.1.1 ASNet Hyperparameters

We use the same architecture and hyperparameters for all ASNet experiments, except where explicitly indicated otherwise. We arrived at these hyperparameters with a two-stage tuning process. In the first stage, we applied the Ray Tune automated hyperparameter tuning framework (Liaw, Liang, Nishihara, Moritz, Gonzalez, & Stoica, 2018) and the random forest optimiser from scikit-optimize[7] to find domain-specific hyperparameter settings that maximised coverage on the benchmark problems after two hours of training. In the second stage, we manually interpolated between the automatically-tuned, domain-specific hyperparameters to find a common set of hyperparameters that worked well on all domains. We report those common hyperparameters below.

Training configuration Our networks have two proposition layers and three action layers (i.e. $L = 2$), with $d_h = 16$ output channels for each action or proposition module. Training is divided into a series of epochs, each of which begins by sampling up to 70 trajectories from the training problems and adding them to the replay buffer, followed by $T_{\text{train}} = 700$ batches of network optimisation. More specifically, at the beginning of each epoch, up to $T_{\text{explore}} = \lceil 70/|P_{\text{train}}| \rceil$ trajectories are sampled from each of the $|P_{\text{train}}|$ different problems simultaneously, with trajectory sampling terminating as soon as each problem has had at least one trajectory sampled. Early termination of the sampling process allows for more concentrated sampling from small problems where planning is cheap, while still sampling at least some trajectories from larger problems that require more time to sample each trajectory and perform planning. For probabilistic problems, the default teacher planner used to label collected trajectories is LRTDP with the h-add heuristic on an all-outcomes determinisation. For deterministic problems, the default teacher planner is A* with the h-add heuristic. After data collection, the batches used for training the network each consist of 64 samples drawn equally from across the training problems. Parameter optimisation itself uses the Adam

[6] `https://github.com/qxcv/asnets`

[7] `https://github.com/scikit-optimize/scikit-optimize`

optimiser ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) with a learning rate of 10^{-3}. We apply an ℓ_2 regulariser of $2 \cdot 10^{-4}$ to prevent weights from exploding, and dropout probability of 0.1 for all layers. We arrived at these hyperparameters through a mix of automated hyperparameter search and manual fine-tuning. Training time is capped at two hours, and we set $T_{\text{stop}} = 20$ with $p_{\text{solved}} = 1$. That is, training can terminate early if the network obtains a success rate of 100% on its training rollouts for 20 consecutive epochs.

Testing configuration When testing ASNets on probabilistic problems, we perform 30 rollouts per test problem using both deterministic execution (where the agent takes a most-favoured action $a_t \in \arg\max_a \pi(a|s_t)$, with ties broken using the lexicographic order of action names) and stochastic execution (where the agent samples $a_t \sim \pi(a|s_t)$). The results of stochastic execution are marked with a "PE" (for Probabilistic Execution) in graphs and results tables. When testing ASNets on deterministic domains, we again test both the strategy of picking the action with highest $\pi(a|s_t)$, and the strategy of sampling $a_t \sim \pi(a|s_t)$. Again, the runs that use the latter strategy are marked with a "PE". However, when choosing actions deterministically in deterministic domains, we only perform a single rollout instead of 30 rollouts, since all rollout outcomes are identical. Likewise, when choosing actions stochastically in deterministic problems, we only perform 10 rollouts instead of 30.

In addition to varying the strategy used to pick actions, we also include results for ASNet runs where there are no LM-cut landmarks or past action counts available to the network. This gives us an indication of how essential those features are in different domains. In graphs and tables, runs with neither landmarks nor action counts are marked "no h." (for "no heuristic inputs"). Further, in probabilistic problems, we perform experiments using a teacher planner with the (admissible) LM-cut heuristic on an all-outcomes determinisation, instead of the default h-add heuristic on an all-outcome determinisation. In some domains we found that the h-add heuristic led to slightly suboptimal teacher policies (e.g. this was the case in Exploding Blocksworld and Probabilistic Blocksworld, as explained in Section 5.3.2 and Section 5.3.3). Substituting in LM-cut ensures the ASNet is *always* trained to mimic optimal action choices on the training problems, albeit at the cost of the teacher planner sometimes taking longer to converge. Runs with the LM-cut heuristic are marked "adm." (short for "admissible teacher heuristic").

5.1.2 Baselines

On probabilistic domains, we compare against a total of four different baseline configurations, each composed of one of two different baseline planners and one of two heuristics. The baseline planner is either LRTDP (Bonet & Geffner, 2003) or SSiPP (Trevizan et al., 2017), while the heuristic is either LM-cut (Helmert & Domshlak, 2009) or h-add (Haslum & Geffner, 2000) heuristic, both with the all-outcomes determinisation. For each problem, we evaluated each baseline by sequentially performing 30 runs in which planners were re-initialised from scratch with different random seeds, and applied a three-hour cutoff. Runs that were not evaluated within the cutoff time were marked as failures. Specific planner configurations were as follows:

LRTDP During each evaluation run, LRTDP was executed until its value estimates converged to within a tolerance of $\epsilon = 10^{-4}$, after which we performed a single rollout of the recovered policy. If the value estimates did not converge within 60 seconds of the end of the allotted time, we suspended the planning process and executed a greedy policy with respect to the un-converged state values for a maximum of 300 time steps. We found that this modification fractionally increased cumulative coverage on some domains, relative to counting runs where LRTDP did not converge as failures. We included LRTDP in our comparison as its performance is representative of the kind of heuristic search planners that are currently popular for solving SSPs.

SSiPP SSiPP was repeatedly executed as a re-planner until 60 seconds before the time cut-off, or until it completed 50 successful rollouts (whichever came first). The short-sighted SSPs encountered by SSiPP were constructed with a horizon of 3 steps, and solved with LRTDP. After termination, SSiPP's policy was executed once, regardless of whether or not it had converged. We included SSiPP in our comparison because it makes use of limited lookahead to solve problems. We can thus compare its results to those of the ASNet policies to evaluate the degree to which trained ASNets behave like limited lookahead planners in practice.

On deterministic domains, we compare against three different types of baseline planners in five different configurations, which we describe below. For all planners, we used the corresponding implementation from Fast Downward (Helmert, 2006).

A* (LM-cut, LM-count) Two configurations of standard A* search, one with the admissible LM-cut heuristic, and another with the inadmissible LM-count heuristic.

GBF (LM-cut) Greedy best-first search (i.e. A* with $g = 0$ and no node re-opening), guided by the LM-cut heuristic. This configuration is included to determine how well a planner could do using *only* LM-cut heuristic values.

LAMA-2011 and LAMA-first We compare against the state-of-the-art LAMA-2011 portfolio planner (Richter, Westphal, & Helmert, 2011), as implemented in Fast Downward. We also compare against the first stage of the LAMA-2011 portfolio, which we label "LAMA-first" in experiments.

5.1.3 Hardware

To provide a fair comparison, all ASNet and baseline planner runs were executed on the same hardware. Specifically, each run was restricted to a single core of an Intel Xeon Platinum 8175 processor attached to an Amazon AWS `r5.12xlarge` instance, with 16GB of memory available per run. We opted not to use GPUs or multiple CPU cores because doing so would have allowed ASNets to exploit parallelism that is not available to the sequential baseline planners. In informal tests, we found that the filters in ASNets were too small to effectively exploit GPU resources. This stands in contrast to large image CNNs, which are bottlenecked by large matrix multiplication and convolution operations that can be efficiently parallelised on GPU hardware. However, ASNets could still utilise all available CPU cores during training if not restricted to a single core.

5.2 Evaluation Domains

This section describes the set of training and testing problems for each of our seven evaluation domains. We will describe the Triangle Tireworld domain in additional detail because it appears again in Section 6. We will also provide a detailed description of CosaNostra Pizza, which some readers may not be familiar with. Detailed descriptions of the remaining domains are left to Appendix B.

Triangle Tireworld (Little & Thiébaux, 2007) This is a navigation task in which the agent must move a vehicle from one corner of a triangular road network to another. Whenever the vehicle moves from location a to location b, there is a 50% chance it will arrive in b with a flat tire, in which case it must replace the damaged tire with a spare before moving again. Unfortunately, only some locations have spare tires. In particular, there are spare tires at every location along the outside edges of the triangle, but no tires on the inside edge of the triangle, and only tires at some locations in the interior. The precise pattern of tire locations is illustrated in Figure 5. The policy with the highest probability of reaching the goal is to take the longest path from the start to the goal around the outside edges of the triangle, thereby ensuring that all visited locations have a spare tire. Little and Thiébaux (2007) introduced this domain to serve as a simple example of a "probabilistically interesting" planning task; that is, a task in which planners must intelligently account for risk. Determinised heuristics ignore the risk of negative action outcomes, and thus encourage heuristic search to look for (risky) paths through the middle of the triangle. We train Triangle Tireworld policies on three problems of sizes 1–3, and test on 17 problems of sizes 4–20.

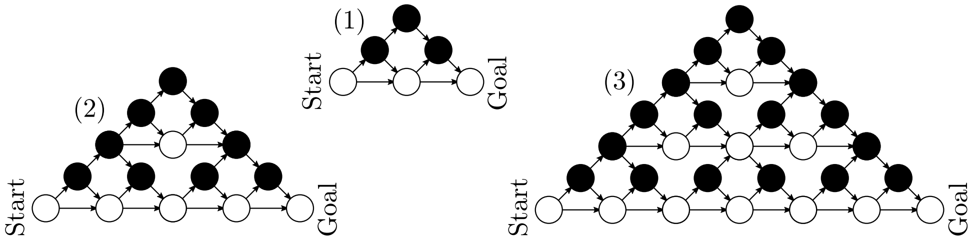


Figure 5: Illustration of the tire placement pattern in the first three Triangle Tireworld problems, adapted from Little and Thiébaux (2007). Black locations, including those around the outside edge, have spare tires; white locations do not. The road structure of larger instances is constructed recursively from that of smaller ones.

CosaNostra Pizza (Toyer et al., 2018) CosaNostra Pizza is a simple probabilistic domain that has been designed to be challenging for current heuristic search planners. The agent's task is to pick up a pizza from a shop, drive a vehicle to a customer's house, deliver the pizza, then drive back to the shop. The shop and the customer's house both lie at opposite ends of a chain of locations. At each location is a toll booth: the agent may either spend one time step paying the operator, or save a time step by driving through without paying. However, if the agent chooses not to pay, then there is a 50% chance that the toll booth operator will drop the boom gate and crush the agent's vehicle the next time they pass through that toll booth. The optimal policy thus pays all toll operators on the path from the shop to the customer, thereby ensuring that the agent can return to the shop safely. Paying operators twice is unnecessary, so an optimal policy will not pay any of them on the path back from the customer to the shop. Like Triangle Tireworld, this domain poses a significant challenge to heuristic search planners that use determinised heuristics, since they ignore the risk of the toll operator crushing the agent's car. This domain also poses a challenge to delete-relaxation heuristics: a delete-relaxation heuristic evaluated in the initial state (at the shop) will believe that upon delivering the pizza, the agent is both at the customer's house *and* at the shop. Thus, the heuristic will not recommend taking actions that could decrease the cost (or risk) of the return path. We train CosaNostra Pizza policies on five problems with 1–5 toll booths, and test on 17 problems with 6–50 toll booths.

Probabilistic Blocksworld A simple probabilistic version of Blocksworld in which blocks may sometimes slip from the gripper onto the table. Our version differs from the IPPC version (Younes & Littman, 2004) in that it lacks actions for moving towers of blocks, and can only move a single block at a time. The modified version produces n-block instances with $O(n^2)$ ground actions, rather than $O(n^3)$ ground actions in the version with tower movement operators. Without this change, we found that our grounding and network construction code was too memory-intensive to perform experiments on larger (30+-block) instances where heuristic search-based planners struggle. We train Probabilistic Blocksworld policies on 25 problems with 5–9 blocks, and test on 30 problems with 15–40 blocks.

Exploding Blocksworld (Younes & Littman, 2004) A challenging probabilistic version of Blocksworld in which blocks can "explode", which leads to problems with both avoidable and unavoidable dead-ends. This domain has been modified to remove a bug in which blocks could be stacked on top of themselves. We train on 24 problems with 5–9 blocks, and test on 30 problems with 11–20 blocks.

Gold Miner (Fern, Khardon, & Tadepalli, 2011) A deterministic domain in which an agent must navigate through a grid to find gold, destroying obstacles along the way. We train on 17 problems with sizes from 4×4 to 6×6 then test on 21 problems of size 7×7 to size 19×19. Importantly, while we use the instance generator from the IPC 2008 Learning Track, we do *not* use the same train or test instances. We found that the original training instances did not provide an adequate curriculum for training: they were either too small to learn useful control knowledge (the bootstrap distribution) or too difficult to solve with heuristic search (the target distribution). Likewise, we found that the test instances did not cover a wide enough range of difficulty to produce an informative comparison with heuristic search. Instead, we generate a new training set in which problems are an appropriate size for our teacher planner, and a new test set that includes substantially larger instances which cannot all be solved within three hours.

Matching Blocksworld (Fern et al., 2011) A more challenging deterministic variant of the traditional Blocksworld domain with *two* grippers, where incorrectly using one gripper instead of the other can lead to a dead end. We train on 23 problems with 5–9 blocks each, then test on 30 problems with 15–60 blocks each. Again, we use the instance generator from the IPC 2008 Learning Track, but generate a new training set and a test set with larger problems.

Blocksworld The standard deterministic Blocksworld domain. We have included both deterministic Blocksworld and Probabilistic Blocksworld in our evaluation to demonstrate that ASNets can obtain good coverage on a domain regardless of whether it has any probabilistic elements. In Section 5.5, we also respond to the "Blocksworld challenge" of Geffner (2018b) by showing that ASNets can generalise extremely well to large Blocksworld instances. We train on 25 problems with 8–10 blocks, and test on 30 problems with either 35 or 50 blocks each.

5.3 Results of Time-Based Evaluation

We present our main results as cumulative coverage graphs in Figure 6 (probabilistic domains) and Figure 7 (deterministic domains). Specifically, for each domain, we plot the cumulative fraction of evaluation rollouts that have reached a goal state, summed over the total number of problems seen so far. The fraction of evaluation rollouts that have reached the goal for a specific problem is always a number between 0 and 1, so the sum of those fractions over all n evaluation problems will be a value between 0 and n. For instance, say that after 5 minutes, a given planner has reached the goal on 30/30 runs on one problem, and on 15/30 runs on another problem, but has not finished planning on any other problems. As a result, its cumulative coverage at 5 minutes would be $30/30 + 15/30 + 0 = 1.5$. For deterministic domains, both the ASNets and the baselines generally only do one run per problem, so each problem contributes either 1 (solved) or 0 (not solved) to the total cumulative coverage. The exceptions are the stochastic rollouts marked "ASNets (PE)" where several trajectories are sampled from the same policy. Those runs are plotted in the same way as for evaluation rollouts on probabilistic domains, where each of the k runs on a single problem contributes $1/k$ to the total cumulative coverage if it is successful, and 0 otherwise. The number of test instances for each domain is plotted with a red, dashed horizontal line. Note that we include the time taken to train ASNets in these plots, so all the ASNet runs have zero cumulative coverage during the training period of up to two hours, after which cumulative coverage increases as the ASNets are evaluated on problems.

In addition to cumulative coverage plots, we have included detailed coverage and solution cost tables in Appendix A. There is one table per domain. Each row of a table corresponds to a different problem in the test set, while each column corresponds to a different planner or ASNet configuration. For probabilistic domains, each cell shows the fraction of planner evaluations that reached the goal (e.g. 29/30) within the allotted total time limit for the problem. Each cell also shows the mean cost of trajectories that reached the goal, along with bounds for a 95% confidence interval (e.g. "25.2 ± 0.4"). Cells with a "-" indicate that the corresponding planner did not produce *any* trajectories that reached the goal within the allotted time. For deterministic domains, we generally only compute one plan (i.e. rollout) per planner, and so we only report the cost of that plan, or a "-" if the planner could not solve the problem within the allotted time. The exceptions are the "ASNet (PE)" runs, where we report the mean cost of stochastic rollouts that did reach the goal, along with the fraction of trajectories that reached the goal in parentheses (e.g "7/10"). Our discussion here will focus on the cumulative coverage plots in Figure 6 and Figure 7, although we will occasionally highlight portions of cost tables in Appendix A.

5.3.1 The Big Question: When is it Worth Training an ASNet?

Across the plots for from Figure 6 and Figure 7, we see a common pattern. Initially, the baseline planners rapidly solve a collection of small instances, while the ASNets remain at

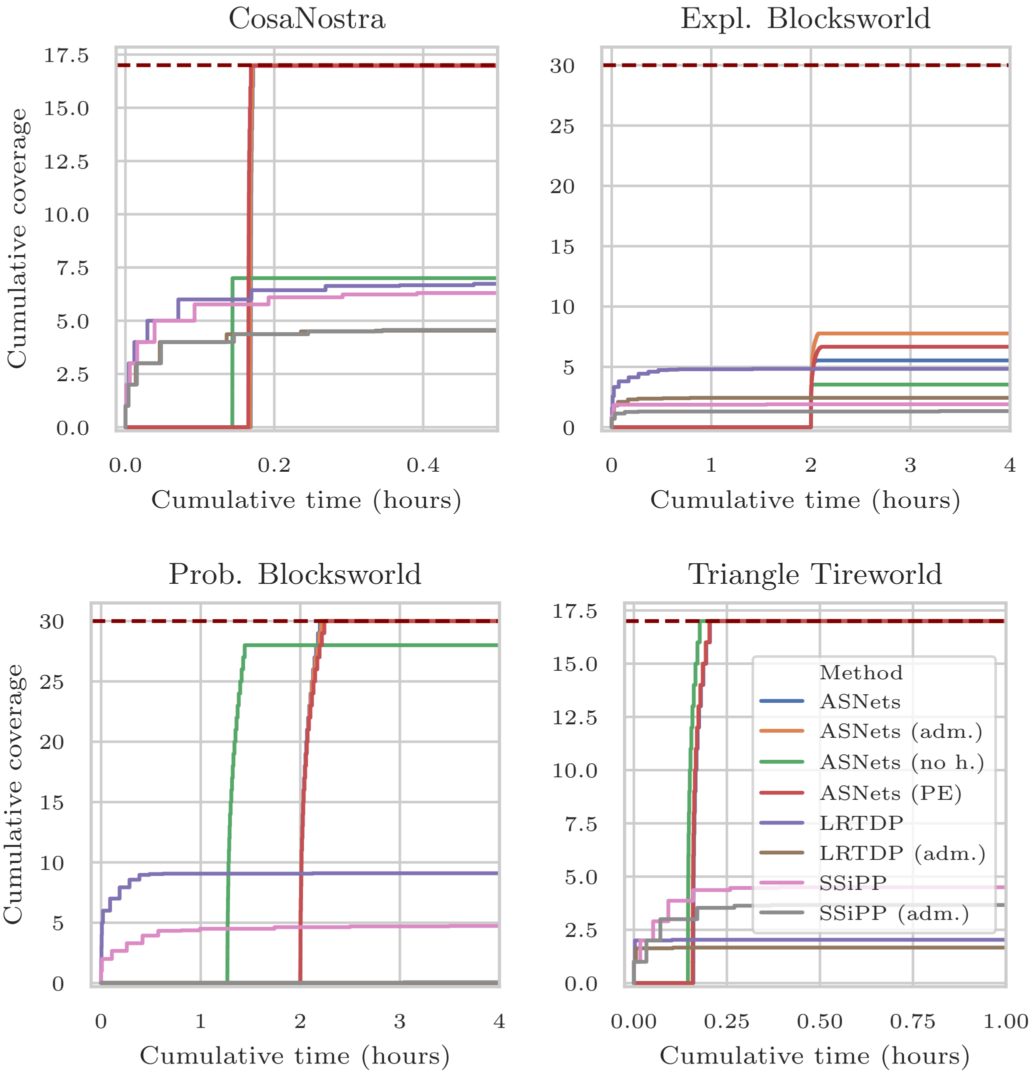


Figure 6: Cumulative success rate of rollouts across a set of test problems for each probabilistic domain. All ASNet configurations except the configuration without heuristic input ("no h.") asymptotically achieve the same (perfect) coverage in CosaNostra, Triangle Tireworld, and Probabilistic Blocksworld; lines for missing configurations are simply occluded by the other configurations. Refer to main text for interpretation of plotted quantities.

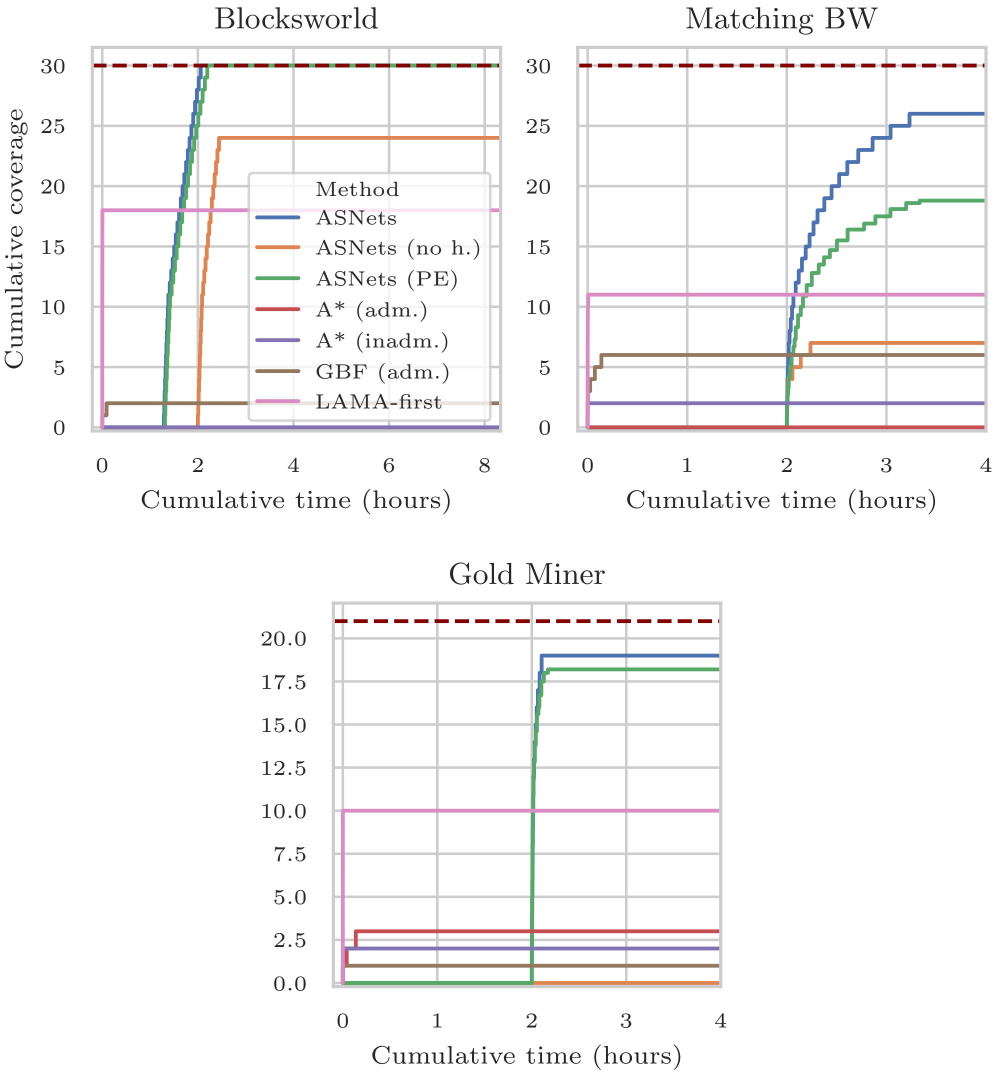


Figure 7: Cumulative number of problems solved over time for deterministic test problems. Refer to main text for description of plotted quantities. LAMA-2011 is omitted: it achieved the same coverage as LAMA-first on our evaluation problems, but took much more time to execute on each problem because it continued to evaluate improved planners after finishing its first planning stage. As a result, the LAMA-2011 curve for cumulative coverage over time was dominated by the corresponding curve for LAMA-first. LAMA-2011 results are instead presented in tabular form in Appendix A.

zero coverage because they are training. However, once the ASNets are trained, they quickly surpass the baselines in cumulative coverage because they can be rapidly evaluated on test problems. Further, the performance of the ASNet runs consistently plateaus at a higher level than the baselines. From the cost tables in Appendix A, it can be seen that in most domains, this difference in coverage arises from the fact that ASNets can solve some very large problems that the baseline planners cannot solve within the three-hour cutoff. We will discuss detailed results for each domain in the next section, but the upshot is that it's worthwhile to train an ASNet in most of these domains if you need to solve a single very large problem, or a collection of moderately large problems.

5.3.2 Results Organised by Domain

CosaNostra Pizza Results for CosaNostra Pizza are presented in Figure 6 and Table 3 (Appendix A). All four ASNets learn the same (optimal) policy of paying the toll booth operators on the way to the customer's home, then driving through the toll booths without paying on the way back. The baseline planners can solve some test instances, but cannot solve instances with more than 14 toll booths, since all of them are guided by determinisation-based heuristics that ignore the possibility of the toll booth operator damaging the vehicle on the way back from the customer. This domain represents a favourable case for ASNets: not only was it constructed to be particularly difficult for planners with determinising heuristics, but it also has a simple trick (pay the operator on the way to the customer) that makes all instances trivial.

Exploding Blocksworld Results for Exploding Blocksworld are presented in Figure 6 and Table 4 (Appendix A). Because Exploding Blocksworld problems generally contain unavoidable dead ends, we do not know the theoretical maximum cumulative coverage for our test set. The thick, dotted red line at the top of the plot in Figure 6 simply shows the number of test instances, which is a loose upper bound on the maximum cumulative coverage. Nevertheless, we can see that the ASNet configured to use an optimal teacher (LRTDP with the admissible LM-cut heuristic) obtains substantially higher cumulative coverage than the baselines. Table 4 shows how this difference in cumulative coverage arises. The baselines manage to find reliable solutions for a handful of test problems, and the high proportion of successful rollouts on those problems account for most of their total cumulative coverage. In contrast, the ASNet policies are generally less reliable on problems where the baselines do well, but still manage to produce a few goal-reaching trajectories on difficult test problems where the baselines fail entirely. This suggests that ASNets may not have learnt to fully exploit domain-specific tricks (like intentionally detonating explosive blocks in order to prevent them from damaging other blocks), but have still learnt a policy that is good enough to partly solve a wide range of problems. Given enough planning time, the baseline planners can obviously "discover" all the tricks required for a reliable policy on one specific instance, but their inability to transfer knowledge between problems means that they must discover all of those tricks anew on each test instance.

Probabilistic Blocksworld Results for Probabilistic Blocksworld are presented in Figure 6 and Table 5 (Appendix A). Here three of the ASNet configurations yield similar policies with perfect coverage on the test set. The largest test instances have 40 blocks, but the baselines struggle to scale beyond 25 blocks. On the training instances we found that LRTDP with LM-cut only visited 20% fewer states on average than LRTDP with the zero heuristic, and actually took more wall time because of the overhead of heuristic evaluation. It is therefore unsurprising to see that it does not solve any of the test instances. Table 5 shows that while SSiPP sometimes produces goal-reaching trajectories on larger problems, it never does so within the three-hour limit on more than 1/30 rollouts, and its solution costs are generally poor (e.g. 583 actions for one of the 35–block instances, where the ASNets require slightly over 120 on average). The ASNet configuration with no heuristic information does surprisingly well: it only fails on two test instances, where it eventually gets stuck repeatedly picking up and putting down the same block. We posit that this may be because towers with misplaced blocks usually have pairs of blocks near the top that are not meant to be on top of one another in the goal.[8] Thus, the network can unstack those towers even though the receptive field limitation prevents it from "seeing" all of the blocks in the tower.

Triangle Tireworld Results for Triangle Tireworld are presented in Figure 6 and Table 6 (Appendix A). As with CosaNostra Pizza, all four ASNet configurations learn the same optimal policy of navigating around the outside edge of the triangle to the goal, changing tires when necessary. In contrast, the baselines struggle to solve instances beyond size 13 because their determinising heuristics neglect to account for the possibility of losing a tire at a location with no spare. On the training instances, we found that LRTDP with h-add or LM-cut failed to significantly outperform LRTDP with the zero heuristic in terms of number of states visited. We also found that LRTDP with LM-cut took slightly more wall time than blind search because of the overhead of heuristic evaluation. Only SSiPP solves the instances beyond size 5, and that is likely because its fixed-depth lookahead strategy is particularly well-suited to Triangle Tireworld. Like CosaNostra, this domain is particularly favourable to ASNets, since it is deliberately constructed to be difficult for planners that ignore stochastic transitions, but can easily be solved with a simple domain-specific strategy. Specifically, a policy can follow the edge of the triangle by repeatedly moving from one location to an adjacent location that has a spare tire, and which is also adjacent to a third location with a spare tire.

Deterministic Blocksworld Results for Deterministic Blocksworld are presented in Figure 7 and Table 7 (Appendix A). Again, we see a similar story to Probabilistic Blocksworld: the three ASNet configurations each manage to solve all or most of the test problems, while the baselines struggle with the larger (50-block) problems. A* does not solve any of the test problems, all of which are relatively large (35+ blocks), while GBF solves only two problems. LAMA-2011 (in Table 7) and LAMA-first solve all of the 35-block problems and some of the 50-block problems, but tend to produce much longer plans than the ASNets. For example, there are five 50-block problems where LAMA baselines both require 200-286 actions to reach the goal, but the ASNets need only 134-164 actions.

Matching Blocksworld Results for Matching Blocksworld are presented in Figure 7 and Table 8 (Appendix A). Matching Blocksworld is much more challenging for the baseline planners: while some of them can quickly solve the small instances (particularly those with

[8] This is not an exotic property of the instance generator, but a consequence of the fact that few blocks in a random initial state will happen to sit on the same block they are meant to sit on in a different, random goal state.

15–25 blocks), they do not solve many of the larger instances. In contrast, the best ASNet configuration manages to solve most instances, including one of the three instances with 60 blocks. However, it is still somewhat surprising that ASNets do not achieve higher coverage. When tuning hyperparameters, we found that performance on Matching Blocksworld was quite sensitive to the choice of ℓ_2 regularisation coefficient and dropout, and that with the best combination of regularisation (around 10^{-4}) and dropout (around 0.25) we could generally obtain higher test coverage. The same hyperparameters did not transfer well to Exploding and Probabilistic Blocksworld, so we did not use them for the final evaluation. Nevertheless, this suggests that ASNets' limited generalisation on Matching Blocksworld is a consequence of our training strategy and hyperparameters, rather than a fundamental representational limitation.

Gold Miner Results for Gold Miner are presented in Figure 7 and Table 9 (Appendix A). As with Matching Blocksworld, we find that ASNets manage to solve almost all test problems when equipped with heuristic input features, while the baselines fail to solve the larger instances featuring 13x13 to 19x19 grids. The solutions produced by LAMA-2011 and LAMA-first for the larger instances also tend to be quite inefficient; in the case of LAMA-2011, this may be because only the first planner in the portfolio (LAMA-first) manages to finish within the three allotted hours. We did not find an obvious pattern in the executed trajectories on the instances where ASNets failed to reach the goal. Further, as with Matching Blocksworld, we found that alternative hyperparameter settings led to policies that would solve all test instances. Again, this suggests that the limited generalisation of ASNets on this domain is due to a flaw of our training strategy, as opposed to some fundamental representational limitation.

5.3.3 Other Discussion Questions

Are ASNets performing fixed-depth lookahead in state space? No. As noted in Section 3.5, ASNets' receptive field limitation is superficially similar to the limitations of short-sighted probabilistic planning or model-predictive control, both of which choose actions by solving a series of short-horizon sub-problems. However, the receptive field of an ASNet is a consequence of the relatedness of actions and propositions, rather than of limited lookahead in state space. Our comparison with SSiPP in the probabilistic domains experimentally demonstrates the difference. In all four domains, SSiPP's strategy of repeatedly solving fixed-depth sub-problems is able to solve some instances. However, its coverage always plateaus well before that of ASNets, as we might expect given the substantial differences between the two strategies.

Do slightly sub-optimal teacher planners lead to worse ASNet policies? In two domains (Exploding Blocksworld and Probabilistic Blocksworld), our results show that slight suboptimality of teaching data leads to slight suboptimality of ASNet policies. In our experiments on probabilistic domains, we performed ASNet runs with both an optimal teacher (LRTDP with the LM-cut heuristic) and a potentially-suboptimal teacher (LRTDP with h-add). On the training problems for Triangle Tireworld and CosaNostra Pizza, we found that LRTDP with h-add always returned optimal solutions after a similar wall-time duration to LRTDP with LM-cut. Our results confirm that there is indeed no gap between ASNet policies trained by LRTDP with h-add and ASNet policies trained by LRTDP with

LM-cut, either in terms of either coverage or mean solution cost. On the training problems for Exploding Blocksworld and Probabilistic Blocksworld, we found that LRTDP with h-add returned solutions that were slightly (1% to 2%) more costly than those for LM-cut. Consequently, we see that the cost of trained ASNet policies becomes fractionally higher in these domains when using a h-add teacher planner rather than an LM-cut teacher planner. It's worth noting that we did obtain better coverage on Exploding Blocksworld when using an optimal planner instead of a sub-optimal planner. However, when training with different ASNet hyperparameters, we found that this relationship sometimes reversed, and the ASNet with a sub-optimal teacher would occasionally do better than the one with an optimal teacher (and both would do better than the baselines). For this reason, we suspect that the better coverage of the ASNet trained with a sub-optimal teacher is due to an interaction between choice of planner and other hyperparameters, rather than some intrinsic advantage of training with optimal planners. For deterministic domains, we simply used a sub-optimal teacher for all runs, as we did not find that an optimal planner lead to increased coverage.

Are heuristic input features always necessary? Not always, although they are important in most of the domains that we evaluated on. For simple domains like Triangle Tireworld, it is straightforward to train correct generalised policies without any heuristic features at all, as we discuss in Section 6. Heuristic features were helpful for more complex domains, however, as shown by the poor performance of ASNets without heuristic inputs on blocksworld-type domains, on Gold Miner and on CosaNostra. In all of these domains, it's easy to see why that's the case: for example, in blocksworld-type problems, the blocks at the top and bottom of a long tower will not be in the receptive field for any one enabled action, and so heuristic inputs are necessary to determine whether the tower contains any misplaced blocks. In CosaNostra, it's not always possible to tell which direction leads to the customer versus to the shop without the use of heuristic inputs, while in Gold Miner the agent cannot "see" the location of the gold without heuristic inputs.

Are LM-cut heuristic values sufficient to trivially solve our test tasks? No. The most obvious evidence of this is the limited performance of the GBF planner equipped with LM-cut in the deterministic domains of Figure 7. Although it can solve some test problems easily, it cannot solve all of them. Likewise, the A*, LRTDP, and SSiPP baselines equipped with LM-cut never managed to exceed the final coverage of the ASNets. This shows that LM-cut heuristic values alone are not sufficient to solve these problems, and also suggests that the LM-cut-based input features given to the ASNet are likely not sufficient to solve those problems alone either.

How well do ASNets perform on the remaining IPC 2008 Learning Track domains? There are four domains from the IPC 2008 Learning Track that we have not presented results for: n-Puzzle, Thoughtful (solitaire), Parking, and Sokoban. In all four cases, we could not obtain an ASNet policy that both achieved high coverage on the test problems and managed to scale better than classical baseline planners.

In Thoughtful and Parking, ASNets were stymied by the large number of ground actions and propositions in some instances from the original test set. Both domains include action schemas with relatively high arity: Parking is a blocksworld-like problem with "direct" movement actions that take three blocks as parameters, while Thoughtful is a complex card game in which some action schemas have up to seven parameters. As a result, the number of actions and propositions increases rapidly as a function of the number of objects in a problem. For example, some Parking instances from the original IPC 2008 test set include over 250,000 ground actions. As a result, we found that it was too slow to construct and perform forward propagation on an ASNet for the larger problems from either domain. This limitation is likely to arise in any problem with a very large number of actions and propositions, and is a consequence of the fact that we need to evaluate one or more action and proposition modules for every ground action and ground proposition in the problem. This is a significant limitation of the ASNet architecture, and we leave investigation of alternative architectures that scale more gracefully with action count to future work.

We also failed to find a reliable generalised policy for n-Puzzle. In principle, it may be possible to find an efficient generalised policy with ASNets, since tractable suboptimal policies do exist (Ratner & Warmuth, 1986). While ASNets had little trouble learning to solve training instances from the "bootstrap" training set from IPC 2008, the resulting policies did not generalise to larger problems. This may simply be a consequence of not using large enough training instances, or sufficiently many instances; we leave further experiments for future work.

In Sokoban, we simply could not train a reliable ASNet policy under the evaluation conditions described in Section 5.1. This is not terribly surprising: Sokoban is PSPACE-complete (Culberson, 1997), and so it is highly improbable that any polynomial time generalised policy exists. Groshev, Goldstein, Tamar, Srivastava, and Abbeel (2018) show that it is possible to train an ordinary convolutional neural network to solve instances of a limited class of Sokoban problems on up to 18 × 18 grids. However, their training set consists of 45,000 instances, including 9,000 different obstacle configurations and five combinations of initial state and goal per obstacle configuration. Solving all instances in such a large training set would be very expensive, and thus erode the advantage of training a generalised policy instead of repeatedly invoking a heuristic search planner. We found it difficult to train with more than 10–20 instances while remaining under our self-imposed two-hour training limit. In future work, it would be interesting to evaluate ASNets on this domain with much more training time and a much larger number of training instances.

5.4 Additional Ablations

Table 1 shows the coverage achieved by ASNet configurations which have had various features disabled or modified. Each column corresponds to one domain, and each row to one ASNet configuration. Each cell shows the total cumulative coverage for one configuration on a given domain, along with the number of problems in the domain. If we were to plot the results for these ablations in the style of Figure 6 and Figure 7, then the total cumulative coverage would be the value that the cumulative coverage curve plateaus to in the limit. The default configuration is the one marked "ASNets" in the previous subsection: a network with two (proposition) layers, 16 hidden units, and a teacher guided by the h-add heuristic (LRTDP for probabilistic domains, $A^\star$ for deterministic domains). The other lines in Table 1 correspond, respectively, to the following modifications over the default:

- Replace the new pooling mechanism from Section 3.3 with the previous pooling mechanism from Toyer et al. (2018).
- Remove skip connections from the network.

Configuration	CN	ExBW	PBW	TTW	BW	GM	MBW
Default	**17.0/17**	5.5/30	**30.0/30**	**17.0/17**	**30.0/30**	19.0/21	26.0/30
Old-style pooling	**17.0/17**	4.5/30	**30.0/30**	**17.0/17**	**30.0/30**	**21.0/21**	28.0/30
No skip conn.	**17.0/17**	2.9/30	**30.0/30**	**17.0/17**	**30.0/30**	18.0/21	**30.0/30**
One layer	**17.0/17**	1.1/30	17.5/30	**17.0/17**	0.0/30	**21.0/21**	5.0/30
Three layers	**17.0/17**	**7.1/30**	**30.0/30**	**17.0/17**	**30.0/30**	19.0/21	**30.0/30**
No history	**17.0/17**	4.5/30	**30.0/30**	**17.0/17**	**30.0/30**	5.0/21	26.0/30
No LM-cut	**17.0/17**	5.2/30	**30.0/30**	**17.0/17**	**30.0/30**	13.0/21	25.0/30
No LM-cut/hist.	7.0/17	3.5/30	28.0/30	**17.0/17**	24.0/30	0.0/21	7.0/30

Table 1: Additional ablations for ASNets on our seven test domains. Domains are CosaNostra Pizza (CN), Exploding Blocksworld (ExBW), Probabilistic Blocksworld (PBW), Triangle Tireworld (TTW), deterministic Blocksworld (BW), Gold Miner (GM), Matching Blocksworld (MBW). Boldface indicates that corresponding planner achieved (equal) best coverage for the domain in the corresponding column.

- Go from two proposition layers ($L = 2$) down to one ($L = 1$).
- Go from two proposition layers up to three ($L = 3$).
- Remove "history" input features that count the number of times each action has been executed so far (but not LM-cut features).
- Remove LM-cut input features (but not history features).
- Remove both LM-cut and history features.

All networks were trained using the same procedure and on the same hardware as the original time-based evaluation.

We can draw a number of inferences from these ablations. First, we see that the model is relatively insensitive to the choice of pooling mechanism and use of skip connections. The exceptions are Exploding Blocksworld (where skip connections and new-style pooling both increase coverage), Gold Miner (where new-style pooling reduces coverage), Matching Blocksworld (where the inclusion of skip connections reduces coverage). The fact that deeper ASNets do better on Exploding Blocksworld suggests that the domain benefits from increased network capacity, and both skip connections and improved pooling may improve performance by increasing network capacity. The reason why the new pooling mechanism decreases coverage on Gold Miner is less clear, as is the reason why skip connections decrease performance on Matching Blocksworld. The results for the one-layer network indicate that depth is unimportant for the simplest problems (CosaNostra, Triangle Tireworld), but matters a great deal for more complex blocksworld-type problems, where the deepest (three-layer) network achieves equal or better coverage than the default configuration. Interestingly, the results for Gold Miner *improve* with fewer layers. We speculate that some form of overfitting or additional optimisation difficulties associated with larger networks might be behind this phenomenon. It's also worth noting that the three-layer network achieves equal or better

Blocks	18	25	35	50	
Towers	1–18 (6 sizes)	Rand.	Rand.	1–40 (9 sizes)	Rand.
Instances	9,000	100	100	9,000	100

Table 2: Distribution of block and tower counts used for evaluating the trained ASNets. The "Rand." instances were sampled uniformly from the space of all instances with the corresponding number of blocks, without fixing a certain tower count. Other instances were sampled uniformly from the space of all instances with a specific number of blocks *and* towers.

coverage than the two-layer network across all domains. However, the three-layer network is slower to train and evaluate than the two-layer network as it requires around 50% more time per network evaluation, so we opted to use the two-layer network in the cumulative coverage experiments of the preceding section. Finally, we see that LM-cut landmarks and history features are each responsible for some improvement in coverage on at least one domain. It should be emphasised that because Table 1 evaluates over tens of problem instances rather than thousands, it does not reflect changes in coverage on the far tails of the instance distribution. When performing large-scale experiments on thousands of blocksworld instances in Section 5.5, we observed that landmark features were in fact essential to getting from good generalisation (i.e. above 99%) to near-perfect generalisation.

5.5 Extended Experiments on Deterministic Blocksworld

Recently, Geffner (2018b) has argued that a key limitation of learning-based techniques in AI is their inability to reliably generalise to arbitrary instances of problems from a given class. Not only are some neural network architectures only capable of processing fixed-size vectors of input, but they also tend to be evaluated in settings where low coverage and small evaluation problems are considered acceptable. In the context of Blocksworld, this means that learning-based planning systems sometimes demonstrate "a coverage of 68% on selected instances with seven blocks", when they should be able to demonstrate "near 100% coverage on arbitrary instances" (Geffner, 2018a). ASNets (and graph convolutional networks more generally) represent one possible way of side-stepping the issue of fixed input size. In this section, we perform extended experiments on deterministic Blocksworld which demonstrate that ASNets can achieve very high coverage ("near 100%") on large (35–50 block) instances, even after training on only a few relatively small instances.

For these experiments we modified the hyperparameters from the previous section, using two-layer networks with 20 channels (instead of 16) in each action and proposition module. We used 50 training problems with 8–10 blocks instead of 25 problems, and trained our network for six hours instead of two hours. We also increased dropout to 0.3 and shifted the total number of ASNet rollouts per epoch from 70 to 150, both of which encouraged the ASNet to explore a wider variety of states at training time. The distribution of block and tower counts in our test set is shown in Table 2. Our test set includes 300 instances sampled uniformly at random with 25, 35, or 50 blocks each. It also includes 9,000 instances of 18 blocks constrained to have 1, 3, 5, 10, 15, or 18 towers, as well as 9,000 instances of 50 blocks constrained to have 1, 4, 7, 10, 15, 20, 25, 30 or 40 towers. These were generated using BWSTATES and BWKSTATES (Slaney & Thiébaux, 2001). Since the hardness of optimal planning in Blocksworld is highly correlated with the number of towers in the initial and goal states, constraining the number of towers allows us to consider problems with varying structure and hardness level.

We found that ASNets could solve all 18,300 test instances after training using the hyperparameters described above. Although this strong empirical result is not proof of the policy's ability to generalise to *all* problems in the domain, it nevertheless underscores the ability of ASNets to solve non-trivial planning tasks with very high reliability when given a modest number of training instances. It's worth noting that when using the same hyperparameters and training set as Section 5.1, ASNets would sometimes fail at a small proportion of instances in our extended test set. For example, due to the receptive field limitation, some trained ASNets would get stuck in loops when the top few blocks in all towers appeared to be sitting immediately on top of the correct block, but blocks further down each tower were not in-position. The additional instances used to train the ASNets described in this section likely made them more robust to rare edge cases of that kind.

6. Understanding ASNet policies

Like other kinds of neural networks, ASNets suffer from a lack of interpretability, and in particular a lack of *transparency*: it is difficult for a human to understand what the network is doing internally. This is because ASNets have far too many parameters and activations for a human to keep track of. For instance, the main Triangle Tireworld policy described in Section 5.1 has 7,634 parameters, and an instantiated ASNet for a large Triangle Tireworld problem will also have tens of thousands of internal activations. Ideally, we would instead like trained ASNets to have the property of *simulability*, in the terminology of Lipton (2016): a person should be able to mentally simulate the expected behaviour of the policy, and anticipate scenarios in which it may or may not work. In this section, we show that simple sparsity regularisation can yield ASNet policies that have very few parameters, and thus can be simulated in one's head. Although we do not consider formal verification, we note that the model simplification techniques discussed here would likely make it easier to compile ASNets down into a format that could be automatically verified to be correct with respect to some formal specification of the domain (perhaps expressed as a MILP or SMT problem). This kind of reasoning is an active area of research in the deep learning community (Katz, Barrett, Dill, Julian, & Kochenderfer, 2017; Tjeng, Xiao, & Tedrake, 2019).

6.1 Sparsity through regularisation

To train sparse ASNets, we will use two ingredients: an appropriate regulariser, and an appropriate training strategy. It is well known in the machine learning and statistics communities that an ℓ_1 regulariser tends to lead to sparse model weights (Tibshirani, 1996). For the purpose of this section, we removed the ℓ_2 and dropout regularisers, then added an ℓ_1 penalty $\gamma\|\theta\|_1$ to the batch objective in Equation (6) with $\gamma = 10^{-2}$. We also trained the ASNet for eight hours instead of two, and decayed the learning rate from 10^{-2} to 10^{-3}

(after 30 epochs) to 10^{-4} (after 40 epochs), rather than keeping the learning rate fixed at 10^{-3}. Together, these changes led to an extremely sparse ASNet for Triangle Tireworld that we will present in the next section. We also repeated the same experiment for CosaNostra Pizza, but we will defer discussion of the resulting policy to Appendix D.

6.2 A sparse policy for Triangle Tireworld

First action layer: simply gives large constant activation to changetire modules.

$$\phi^1_{\text{changetire}(?loc)} = 15.76$$

First proposition layer: vehicle-at modules assign high value to locations where tires can be changed, and low value to others:

$$\psi^1_{\text{vehicle-at}(?loc)} = 0.81 \cdot \text{pool}\left(\phi^1_{\text{changetire}(?loc)}\right) + 0.02$$

Second action layer: move-car modules have high activation when the destination has a tire, and low activation otherwise.

$$\phi^2_{\text{move-car}(?from,?to)} = 0.46 \cdot \psi^1_{\text{vehicle-at}(?to)}$$

Second proposition layer: vehicle-at modules have highest activation when $?loc$ has a tire, and there is also a move action to an adjacent location with a tire.

$$\psi^2_{\text{vehicle-at}(?loc)} = 0.35 \cdot \psi^1_{\text{vehicle-at}(?loc)} + 0.44 \cdot \text{pool}\left(\phi^2_{\text{move-car}(?loc,\cdot)}\right) + 6.45$$

Third action layer: the most preferred actions are those that move to a location that has a tire and is adjacent to another location with a tire. changetire is only chosen when no other actions are available.

$$\phi^3_{\text{move-car}(?from,?to)} = 1.12 \cdot \psi^2_{\text{vehicle-at}(?to)}$$
$$\phi^3_{\text{changetire}(?loc)} = 0$$

Figure 8: An easily-readable depiction of the sparse ASNet trained for the Triangle Tireworld problem, along with comments describing the intuitive meaning of action and proposition layers (demarcated by boxes). The ELU activation functions have been omitted because all activations are positive and the ELU is simply the identify function on $[0, \infty)$.

Figure 8 depicts the "lifted" equations defining a sparse ASNet policy for Triangle Tireworld, trained using the procedure described in the preceding section. This policy has a mere eight nonzero parameters. As a result, all of the modules have either been reduced to a single nonzero output or eliminated (zeroed out) entirely, despite starting with $d_h = 16$ nonzero output channels at initialisation. It is thus easy to verify that this policy is correct by cross-referencing with the PPDDL domain definition (Figure 10 in Appendix B). In the first action layer, we have an action module output $\phi^1_{\text{changetire}(?loc)}$ for each location with a spare tire, which simply ignores the input values and outputs a large positive constant. The procedure for collecting ground actions only includes changetire actions for locations with a spare tire in the initial state,[9] so the $\psi^1_{\text{vehicle-at}(?loc)}$ modules at the next layer will pool

[9] Recall that we use the MDPSim grounding code, which instantiates all actions that could possibly be enabled in some state while attempting not to instantiate those actions that will never be applicable. In the case of Triangle Tireworld, changetire($?loc$) depends on a spare-in($?loc$) proposition which cannot

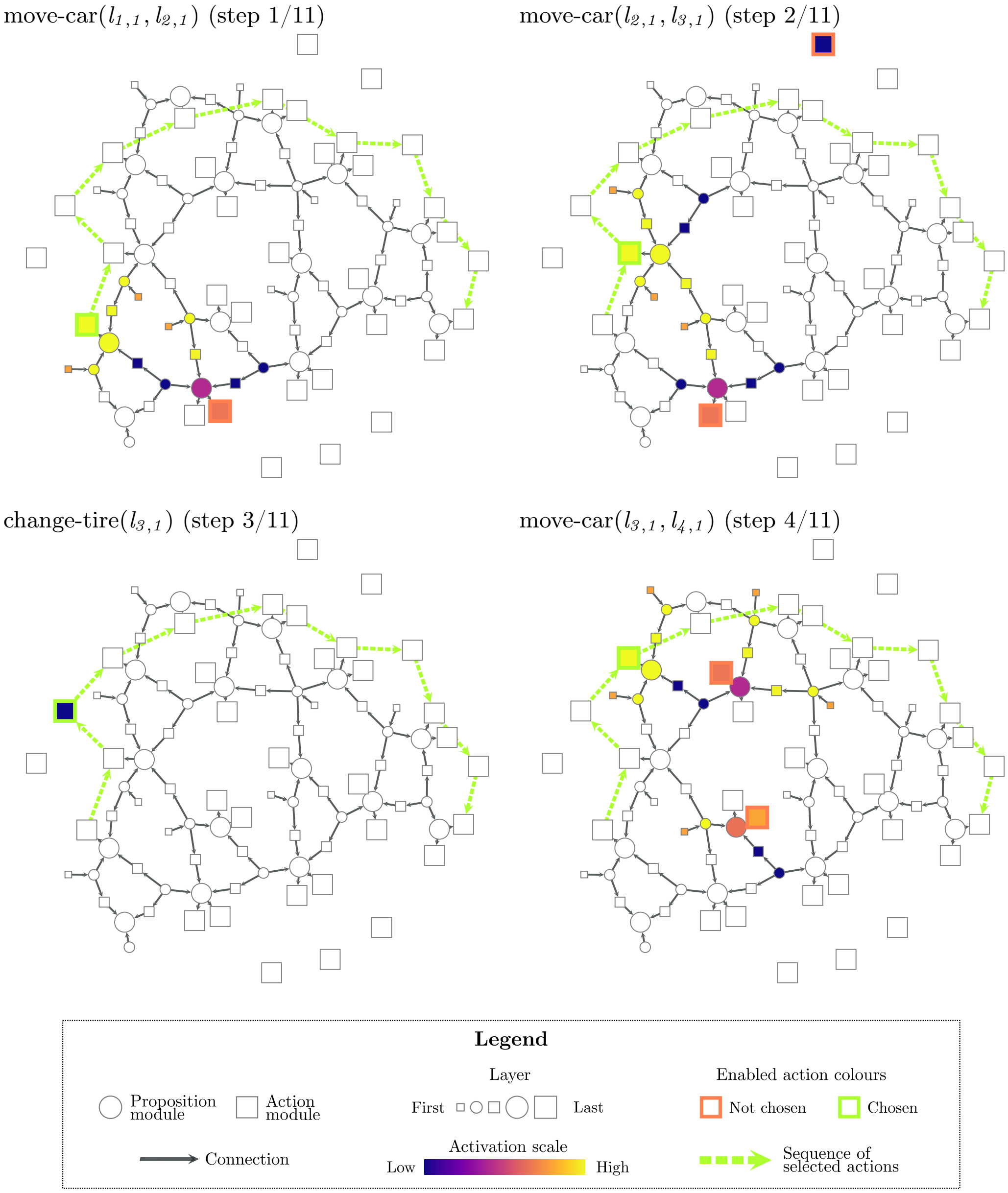

move-car($l_{1,1}, l_{2,1}$) (step 1/11)
move-car($l_{2,1}, l_{3,1}$) (step 2/11)
change-tire($l_{3,1}$) (step 3/11)
move-car($l_{3,1}, l_{4,1}$) (step 4/11)
Legend
Proposition module
Action module
Layer
First
Last
Enabled action colours
Not chosen
Chosen
Connection
Activation scale
Low
High
Sequence of selected actions

Figure 9: Visualisation of the activations of an ASNet for the first four actions in a successful rollout on Triangle Tireworld. The diagram for "step t" represents the action chosen by the ASNet and executed at time t.

over zero or one corresponding $\phi^1_{\text{changetire}(?loc)}$ modules to produce an output. This output will be large if there is a tire at ?loc (and thus a corresponding changetire action), and small otherwise. The $\phi^2_{\text{move-car}(?from,?to)}$ action modules in the next layer simply propagate

be made true by any action, so if spare-in(?loc) is not true in the initial state then changetire(?loc) will never be added to the collected list of actions.

the $\psi^1_{\text{vehicle-at}(?to)}$ value for the destination location up to the next layer. In the proposition layer that follows, $\psi^2_{\text{vehicle-at}(?loc)}$ sums the output of a skip connection back to $\psi^1_{\text{vehicle-at}(?loc)}$ (in the previous proposition layer) and a max-pooling operation over modules for move-car locations that start in $?loc$ and end in some other location. As a result, its value will be moderately high if there is a tire at $?loc$ or if some adjacent location that can be reached with one move-car action, and very high if both conditions are true. In the final layer, $\phi^3_{\text{move-car}(?from,?to)}$ attains the highest positive value when the destination $?to$ has a spare tire and also leads to another location with a spare tire. It's easy to see from the diagrams in Figure 5 that following such actions will keep the vehicle on the outside edge of the triangle as it moves towards the goal, as desired. Further, the final-layer module for changetire always outputs 0, so it will only be chosen when changetire is the only action available (i.e. when the car has a flat tire, as one can see from the domain in Figure 10). We can thus conclude that this policy correctly exploits the structure of Triangle Tireworld instances in order to generalise across the entire domain.

We can gain another perspective on these sparse ASNet weights by using them to instantiate an ASNet for a small Triangle Tireworld problem, and then plotting the activations. Figure 9 depicts such a visualisation for a Triangle Tireworld problem of size two. In this visualisation, module outputs are represented by squares (action modules) or circles (proposition modules), where size increases with layer number. We show activation magnitudes for the first four steps of a successful plan execution. Activations for modules that contributed to a decision are filled with an appropriate colour, while modules that could not have influenced the decision (due to network connectivity) are left white. Additionally, we use a series of thick green arrows between final-layer activation modules to depict the sequence of actions chosen over the entire course of the plan. Notice that the node positions computed by the force-directed layout algorithm that generated this figure roughly match the triangular road network for problem (2) in Figure 5. This a good illustration of how the structure of an ASNet reflects the intuitive structure of the corresponding problem.

The activation diagram in Figure 9 also allows us to check some of the claims we made about the lifted policy. For example, in the top left plot, we can see that the chosen action (large square, yellow with green outline) has a high activation because of the high activation $\psi^2_{\text{vehicle-at}(?loc)}$ of the related vehicle-at module in the preceding proposition layer (large yellow circle). Further, that proposition module has a high activation because of its skip connection to a $\psi^1_{\text{vehicle-at}(?loc)}$ module corresponding to a location with a tire (smaller yellow circle), and because of the fact that it pools over a $\phi^2_{\text{move-car}(?loc,?next)}$ module that leads to a location with a tire (smaller yellow square). Likewise, in the bottom left plot of Figure 9, we can see that the changetire action is chosen despite its low activation (dark blue) because it is the only available action in states where the agent has a flat tire. Together, Figure 8 and Figure 9 thus give us confidence that our learnt policy is correct.

7. Related Work

To the best of our knowledge, ASNets were the first approach to generalised probabilistic and deterministic planning with neural networks. Nevertheless, there is a great deal of related prior work at the intersection of learning and planning, as well as concurrent and later work that deals with similar themes. In this section, we compare ASNets to prior work in planning (Section 7.1 and Section 7.2) and structured deep learning (Section 7.3).

7.1 Learning Generalised Domain-Specific Control Knowledge

We will begin our survey of prior work by contrasting previous approaches to learning domain-specific control knowledge with our approach. Although there are many forms of domain-specific knowledge that can be learnt (Jiménez, de la Rosa, Fernández, Fernández, & Borrajo, 2012), our focus will be on generalised policies, generalised heuristics, and other forms of knowledge that could conceivably be learnt by ASNets. We will decompose prior methods along three axes: first, we consider possible representations for domain-specific knowledge. Second, we consider methods used to learn domain-specific knowledge. Third, and finally, we consider the methods through which the resulting knowledge is exploited to solve unseen planning problems.

7.1.1 Knowledge Representations

Decision lists are one of the oldest and simplest representations for learnt generalised policies in automatic planning. They can be viewed as sequences of if–then rules which allow different actions to be selected when different combinations of logical conditions are satisfied. For instance, Khardon (1999) represents a generalised policy using a sequence of rules for selecting action schemas based on conjunctions over the predicates of a domain. Unfortunately, conditions based on conjunctions over fixed sets of predicates are not sufficient to solve many planning problems of practical interest. Thus, Khardon also employs hand-coded, domain-specific *support predicates* which allow the decision list to encode more powerful action selection rules. Later extensions to Khardon's algorithm allow it to make use of *concept language* (Martin & Geffner, 2000) or *taxonomic syntax* (Yoon et al., 2002). Both concept languages and taxonomic syntax obviate the need for support predicates by allowing more complex logical conditions (e.g. those involving recursion) to be employed in constructing a decision list. de la Rosa et al. (2011) present another extension of Khardon's approach which also increases its expressive power. First, they replace the single decision list with a pair of decision trees:[10] one for selecting an action schema, and one for binding objects to the action schema to obtain a ground action. Second, they introduce new features based on the helpful actions produced by the FF planner. Such heuristic features serve a similar purpose to support predicates, but do not have to be manually coded for each problem; in a sense, use of these heuristic features can therefore be viewed as an alternative to the use of concept language or taxonomic syntax. Gretton and Thiébaux (2004) show that the rich concepts necessary to represent lifted policies can also be obtained by repeatedly applying logical regression to reward formulae in a first-order description of a domain, then performing inductive logic programming with those concepts. It's worth noting that all of these methods can be combined in ensembles to produce more-accurate composite models (Dietterich, 2000). Indeed, past work shows that ensembles can substantially improve the accuracy of the aforementioned generalised planning systems (Yoon et al., 2002; de la Rosa & Fuentetaja, 2017).

[10] Decision lists and decision trees have equivalent power in this context (Blockeel & de Raedt, 1998), although the two representations do lend themselves to different training strategies.

Our approach fundamentally differs from all the aforementioned techniques in that it uses a class of continuously-parameterised function approximators based on graph convolutional neural networks, rather than using discretely-parameterised decision lists (or trees). This distinction is particularly relevant for learning, as explained in Section 7.1.2. Further, unlike Yoon et al. (2002) and de la Rosa and Fuentetaja (2017), we find that our models are sufficiently accurate to solve complex problems (e.g. Blocksworld) without resorting to ensembles. However, our use of heuristic input features is similar to the way that de la Rosa et al. (2011) use helpful actions from FF, in that both methods use inputs derived in part from domain-independent heuristics to increase the range of policies that can be expressed without resorting to hand-coded input features.

Concurrent with the original ASNets paper, Groshev et al. (2018) also proposed a novel representation for generalised policies and heuristics based on neural networks. They propose using a hand-coded, domain-specific translator to convert states of a problem into a form that is amenable to processing by neural networks. For instance, states from Sokoban can be processed by first converting them to 2D images depicting the current and goal positions of all boxes, the layout of the warehouse walls, and the position of the agent. The image can then be passed to a 2D CNN which produces an appropriate action or heuristic value. A travelling salesman problem can likewise be solved by expressing it as a graph of locations to be visited and then processing the graph with a graph convolutional neural network. This approach is similar to ours insofar as it uses neural networks with convolution-like operations. However, it differs from our method in that it requires a manually-engineered input representation for each domain, whereas our method can easily be applied to any planning problem expressed as (P)PDDL.

Sievers, Katz, Sohrabi, Samulowitz, and Ferber (2019) have also used traditional (image-based) convolutional neural networks for planning-related tasks. Instead of learning a generalised policy like the present work or like Groshev et al. (2018), they learn to perform planner selection in classical planning domains. Further, their process for converting planning instances into images is domain-independent. First they convert an instance into either a problem description graph, which captures the structure of variables and effects in a grounded planning task, or an abstract structure graph, which instead captures the structure of an instance at the level of un-grounded PDDL. Next, they render the adjacency matrix of that graph as a binary image. Finally, they dilate the image and resize it to fixed dimensions so that it can be fed to a convolutional neural network. This work differs from ours in that it performs planner selection instead of action selection, operates on a different graph representation, and uses an image-based convolutional neural network to process the graph, rather than a graph convolutional network.

Closer to our work are the more recent ToRPIDo (Bajpai, Garg, et al., 2018) and TraPSNet (Garg, Bajpai, & Mausam, 2019) neural network architectures, both of which allow transfer of learnt knowledge between different RDDL problems. ToRPIDo uses a collection of neural network components, including a graph convolutional network, to represent a policy for a given planning problem. After training on one problem, some components can be directly transferred to other instances, so long as those instances are of the same size as the original training problem(s). This enables faster learning of policies for new problems, since only some components need to be re-learned from scratch. TraPSNet is an improved architecture in which all network components can be transferred between problems of different size, under certain assumptions about domain structure (e.g. that all action templates and fluents are unary). TraPSNet uses a Graph ATtention network (GAT) and global pooling mechanism to learn an embedding vector for each object in a given RDDL problem. By assuming that all action templates are unary, TraPSNet can also learn a sub-network that takes a single object embedding and indicates how desirable each of the corresponding actions are. The GAT can be transferred between different numbers of objects, and the action selection sub-network can be applied separately to each object, and so TraPSNet can be transferred between problems of different sizes. This is analogous to ASNets' ability to transfer policies across problems of arbitrary size, but applied to a subset of RDDL rather than (P)PDDL.

Although ToRPIDo and TraPSNet serve a similar need to ASNets, we do not evaluate against them in Section 5. This is because the three network architectures are closely coupled with either RDDL (TorPIDo and TraPSNet) or PPDDL (ASNets). Automated translations of problems from one language to the other generally produce a separate target-language domain for each input-language problem, which precludes direct comparison of generalisation ability. Nevertheless, a comparison between the key components of the different network architectures (graph convolutions for ToRPIDo and ASNets, graph attention for TraPSNet) on problems expressed in the same language would make for interesting future work.

The STRIPS Hypergraph Network (STRIPS-HGN) (Shen et al., 2020) is another recent approach which uses graph networks to generalise learnt knowledge across tasks. The hypergraph underlying the STRIPS-HGN architecture is derived from a delete relaxation of the problem. The vertices in the hypergraph each correspond to a particular propositions, while the hyperedges each correspond to actions. Specifically, for each action a, there is a hyperedge linking the set of vertices corresponding to pre_a to the set of vertices corresponding to eff_a. The input to the network is an initial set of vertex and hyperedge features derived from the current and goal states; the output is a single heuristic estimate for the state. Unlike ASNets, STRIPS-HGN architectures do not depend on the specifics of any particular domain, and so a single set of learnt weights can be applied to *any* state of *any* STRIPS problem. Consequently, Shen et al. show that it can be used to learn either domain-specific or domain-independent heuristics. Further, STRIPS-HGN uses a form of weight tying across different layers that allows the same "layer" (i.e. a single set of weights) to be applied multiple times to the same input, in much the same fashion as a recurrent layer in an RNN. This can in principle overcome the receptive field limitation of ASNets by allowing the same layer to be applied as many times as is necessary to propagate information across the graph. Shen et al. do not apply the architecture to probabilistic problems, and it is not clear whether it is capable of learning effective generalised *policies*, rather than just heuristics.

7.1.2 Knowledge Acquisition

In addition to an appropriate representation for domain-specific knowledge like generalised policies, we also need an appropriate learning algorithm to acquire that knowledge. Most existing techniques for learning generalised policies incorporate a strategy for obtaining experience—typically in the form of pairs of states and "correct" actions for some small problems—and a strategy for learning a policy from that experience. We will now consider these two aspects of knowledge acquisition (obtaining experience and learning from it) in greater detail.

Acquiring experience Much like our work, most existing approaches to learning generalised policies employ a non-learning "teacher" planner which tells the learning-based planner which actions to take in the states observed at training time. However, prior work differs in how the set of training states is generated. One simple approach is to collect training states by running the teacher planner on randomly-generated problem instances and then labelling and storing all generated states along goal trajectories produced by the teacher (Martin & Geffner, 2000; Yoon et al., 2002). However, the resulting training set would be static, and could not adapt to the observed weaknesses of the planner during training; indeed, Martin and Geffner (2000) show that it is helpful to extend the dataset with states visited during training which were misclassified by the learnt policy. The training set can be expanded further by including *all* optimal goal trajectories for the given training problems, rather than just a subset. This can be achieved, for instance, with a branch-and-bound algorithm (de la Rosa et al., 2011). We found it most effective to use a training set composed of all states visited by the agent during rollouts, plus rollouts under the teacher planner's policy starting in each of those states.

Rather than relying solely on high-quality plans from a teacher, some prior methods use self-supervision or reinforcement learning to learn on problems that may be too large for other planners to handle. For instance, the Factored Policy Gradient (FPG) planner learns a policy for a single problem via policy gradient reinforcement learning, which gradually tweaks the parameters of a learnt policy so that its probability of success increases over time (Buffet & Aberdeen, 2009). We attempted to use the same technique, but found that the random exploration employed by policy gradient methods was too inefficient to solve our benchmark problems. Groshev et al. (2018) present an alternative *leapfrogging* approach that combines self-supervision with supervision from a teacher planner. It begins by using a teacher policy to acquire experience on small problems. Later, it uses its partially-learnt control knowledge to guide a search algorithm on larger problems, and then feeds the actions recommended by that self-guided search algorithm back into its own training set. The same method is likely applicable to our model, although we leave it to future work to investigate this and other techniques for interleaving learning and planning.

Learning a policy from experience Different knowledge representations lend themselves to different learning algorithms for distilling experience into control knowledge. For decision list representations, a standard approach is Rivest's algorithm (Rivest, 1987), which iteratively builds a decision list by adding rules with perfect precision or perfect recall until all samples are classified correctly. This process requires a search over the space of possible conditions at each iteration, so its efficiency is dependent on either restricting the size of this space or having a good search algorithm at hand to find useful conditions. For instance, Yoon et al. (2002) restrict the size of their taxonomic syntax expressions, and employ heuristic-guided beam search to find useful expressions at each iteration. The different knowledge representation of de la Rosa et al. (2011) allows them to instead use standard methods for learning decision trees (Blockeel & de Raedt, 1998; Quinlan, 1986). Nevertheless, all these methods must perform a search through a space of discrete models, and consequently suffer from all of the issues that discrete search entails (high branching factor, difficulty of guiding the search, etc.).

In contrast to the above approaches, we use a continuously-parameterised knowledge representation which can be trained via Stochastic Gradient Descent (SGD). Training via SGD offers a different set of tradeoffs to search in discrete spaces, and arguably provides greater flexibility by allowing us to optimise *any* differentiable loss. For instance, although this paper only examines the performance of our model in a classification setting (with a cross-entropy loss), it could just as easily be used to regress Q-values (e.g with an ℓ_2 loss). Likewise, we could train ASNets with policy gradient reinforcement learning; this could allow ASNets to be used to select planning strategies in the recent framework of Gomoluch, Alrajeh, and Russo (2019), for example. In contrast, past approaches using discrete representations would need substantially different optimisation algorithms in order to extend them to value-learning or reinforcement-learning settings.

7.1.3 Knowledge Exploitation

When learnt control knowledge is expressed in the form of a generalised policy, the most obvious way to exploit it is to simply execute it directly. However, direct execution is not always possible for other forms of learnt control knowledge, such as generalised heuristics, and is not always the best way to make use of a generalised policy. For instance, de la Rosa et al. (2011) note that learnt generalised policies can sometimes include defective rules, and propose depth-first and breadth-first search algorithms to ameliorate this problem: the search is guided by the learnt policy, but is also able to back-track if it reaches a dead end. Beam search (Xu, Fern, & Yoon, 2007) and limited discrepancy search (Yoon, Fern, & Givan, 2006), and ordinary A* search have all been used in a similar manner. For probabilistic problems, it is potentially more appropriate to use sampling-based strategies to either expand the search space of a heuristic search algorithm (Yoon, Fern, & Givan, 2007) or to estimate the Q-values of actions using policy rollouts (Fern, Yoon, & Givan, 2004). Along the latter lines, it is also possible to apply Monte Carlo Tree Search (MCTS) in conjunction with learnt policies, as is done by AlphaGo (Silver et al., 2016). As mentioned in Section 4, concurrent work has shown that using ASNets in conjunction with UCT at test time can help prevent mistakes that might occur due to incomplete training of the ASNet (Shen et al., 2019). In this paper, we are primarily concerned with examining what sort of generalised policies ASNets can represent directly, so we have not experimented further with search-based methods for exploiting generalised policies.

7.2 Other Related Planning Work

Not all work at the intersection of learning and planning fits into Section 7.1's taxonomy for generalised knowledge acquisition. For instance, the previously-mentioned FPG planner uses reinforcement learning to train a neural-net-based policy, but the resulting policy is only able to solve a *single* problem (Buffet & Aberdeen, 2009). Likewise, Issakkimuthu, Fern, and Tadepalli (2018) investigate a restricted class of neural networks for representing single-instance policies, including networks that use a limited form of weight-sharing. Ferber, Helmert, and Hoffman (2020) similarly investigate the properties that make fully-connected neural networks well-suited to learning problem-specific heuristics that generalise only over the initial state of the problem. The primary difference between these papers and this work is our focus on generalising across different problems.

There are also several existing approaches to generalised planning that do not approach the problem from a machine learning perspective. Srivastava, Immerman, Zilberstein, and Zhang (2011) consider how to acquire generalised plans for domains expressed with a restricted form of classical planning. Rather than learning from traces produced by a teacher planner on a small, fixed set of training instances, Srivastava *et al.* instead assume access to an algorithm that can automatically generate states which are not assigned an action by the current (partially-complete) generalised plan. In this way, they can sometimes acquire generalised plans that are *guaranteed* to provide an action for all encountered states, and to terminate eventually. Continuing on the same theme, Hu and De Giacomo (2011) consider the complexity of generalised planning for finite and infinite environments, while Bonet and Geffner (2018) propose a practical generalised planning algorithm that can accommodate changes in the number of objects and actions in the problems of a domain. Francès, Corrêa, Geissmann, and Pommerening (2019) propose an algorithm that can recover generalised heuristics from linear combinations numeric features derived from simple concept language expressions. In some domains, they are able to manually prove that the recovered heuristics are *descending and dead-end avoiding* across the entire domain, and can consequently guide a greedy planner to a goal state in polynomial time. In contrast to these approaches, our approach does not provide any theoretical guarantees about generalisation of a learnt model to unseen instances. However, our method also imposes fewer limitations on the sorts of problems that can (in principle) be solved. For instance, Srivastava et al. (2011) only consider "generalisation to *n*", where problems are identical but for the number of instances of a certain kind of object. It remains to be seen whether there is a compromise approach which can offer reasonable guarantees about generalisation while still remaining applicable to a wide range of problems.

There have also been a number of related techniques that use deep learning to acquire models of an environment, and then obtain policies through reinforcement learning or planning. For instance, Value Iteration Networks (VINs) are a kind of convolutional neural network that can learn to formulate an MDP from an observation of an environment, solve that MDP, and use the result to choose an action (Tamar, Wu, Thomas, Levine, & Abbeel, 2016). Generalised VINs extend this approach to MDPs with more general transition dynamics by employing graph convolutional neural networks instead of ordinary convolutional neural networks (Niu, Chen, Guo, Targonski, Smith, & Kovačević, 2017). In a similar vein, *schema networks* learn a STRIPS-like environment model using a specially-structured neural network, then choose actions by planning on that learnt model (Kansky, Silver, Mély, Eldawy, Lázaro-Gredilla, Lou, Dorfman, Sidor, Phoenix, & George, 2017). Say, Wu, Zhou, and Sanner (2017) show that it's possible to learn transition models for mixed discrete–continuous planning problems using deep learning, and then plan on the learnt model with a traditional MILP solver, as opposed to reinforcement learning. LatPlan (Asai & Fukunaga, 2018) likewise shows that discrete autoencoders (specifically, Gumbel-Softmax VAEs) can learn how to convert image-based observations of an environment into a PDDL problem description, which can then be solved using an ordinary classical planner. Finally, Zambaldi, Raposo, Santoro, Bapst, Li, Babuschkin, Tuyls, Reichert, Lillicrap, Lockhart, Shanahan, Langston, Pascanu, Botvinick, Vinyals, and Battaglia (2019) show that adding a form of self-attention (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, & Polosukhin, 2017) to convolutional network policies can make it easier for reinforcement learning to solve image-based tasks requiring relational reasoning. These approaches bear surface similarities to our approach, such as the use of convolutions, or the use of an internal representation based on the action-proposition structure of a problem. However, they differ in that they all aim to learn models for unknown environments which can then be planned on, as opposed to directly learning a policy for a known environment.

7.3 Structured Deep Learning

ASNets can be interpreted as an extension of convolutional neural networks to handle a different kind of underlying graph structure. These sorts of generalisations have been widely studied in recent years, motivated in part by the desire to apply deep learning to domains where data cannot be expressed as points sampled on a regular n-dimensional spatial grid (Bronstein, Bruna, LeCun, Szlam, & Vandergheynst, 2017). For instance, graph convolutional neural networks have previously been used to model molecules (Duvenaud, Maclaurin, Iparraguirre, Bombarell, Hirzel, Aspuru-Guzik, & Adams, 2015; Kearnes, McCloskey, Berndl, Pande, & Riley, 2016), where a graph structure is induced by the bonds between atoms, and spatial-temporal action sequences (Jain, Zamir, Savarese, & Saxena, 2016), where interactions between objects in an environment are modelled as a graph. These techniques have also been used to reason about relational databases (Uwents & Blockeel, 2005), where the structure of the network is determined by the database schema, and to first-order logic (Sourek, Aschenbrenner, Zelezny, Schockaert, & Kuzelka, 2018), where the structure of the network is derived from the structure of a series of first order logic statements. So far as we are aware, ASNets are the first application of this technique to automated planning.

8. Conclusion and future work

State-of-the-art classical and probabilistic planners currently have limited ability to transfer control knowledge between similar problems or domains. The most common form of transfer is to learn an autoselector or autoconfigurator that chooses an appropriate planner for new problems based on past planner performance, but such methods only have a limited ability to influence the search for plans and policies. This paper discussed Action Schema Networks (ASNets), a neural network representation for generalised policies. An ASNet applies a learnt convolution-like operator to a graph of actions and propositions in order to select appropriate actions for a problem. Because the number of parameters required by this convolution-like operator is independent of any one problem, the resulting policy can be applied to any instance from a given (P)PDDL domain. Our experiments across seven probabilistic and deterministic domains show that training ASNets on some small instances from a domain and then executing the resulting policy on a few large instances can be much faster than applying a heuristic search planner directly to those large instances. In each domain, there is a simple “trick” that the ASNet can learn from small problems which makes planning in larger problems straightforward. In contrast, the baseline planners cannot transfer any of their experience between problems, and must instead re-discover such tricks anew on each instance.

In this paper, we have also challenged the view that neural-network-based policies must necessarily be unreliable and uninterpretable. In response to Geffner (2018a), we showed that ASNets could learn a highly reliable policy on the classical Blocksworld benchmark. Our policy correctly solved 18,300 test instances with 18–50 blocks after training on just 50 smaller instances with 8–10 blocks. We have also shown that it is possible to train a very sparse ASNet policy for Triangle Tireworld which can easily be interpreted by a human. There will still be problems for which ASNets cannot learn a reliable policy, or where the policy cannot be made sparse enough for easy interpretation, but our results show that deep learning still can yield interpretable, reliable policies in some settings.

While ASNets have demonstrated impressive performance on some domains, there still remain many directions for future improvement. One direction is the exploration of search methods that can better exploit the learnt control knowledge in an ASNet, including helping to solve problems that ASNets cannot solve on their own, or for which learnt control knowledge is flawed. Shen et al. (2019) have already taken some steps towards this goal by using ASNets to guide Monte Carlo tree search. Another possible direction is to experiment with different methods of training ASNets. Our current training mechanism simply has ASNets imitate a "teacher" planner on a small set of problems, even though it may not be possible to generalise that teacher's strategy to larger problems. Reinforcement learning is more appealing in this regard, but further research will be required to make RL-based training reliable and efficient enough to be practical on the benchmarks used in classical and probabilistic planning. It would also be interesting to investigate alternative architectures that lift the structural limitations of ASNets. The fixed receptive field of an ASNet is one such limitation, as is its lack of support for quantified action preconditions and arbitrary goal formulae. The computational overhead of ASNets in problems with many ground actions or propositions is another important limitation. Indeed, there may be alternative generalised policy architectures that do not require a ground representation at all! Finally, we note that ASNets could easily be extended to apply to more areas beyond probabilistic and classical planning, including planning problems with numeric state, and problems with concurrent, durative actions. More generally, we hope that ASNets can serve as both a starting point and inspiration for researchers looking to bring the advances of deep learning to the planning community.

Acknowledgments

We would like to acknowledge Jörg Hoffmann for suggesting the alternative pooling mechanism in Section 3, and Hector Geffner for insightful discussion about network structure and generalisation. We would also like to thank our anonymous reviewers for helping to strengthen the paper, particularly in the background and evaluation sections.

Sylvie Thiébaux and Felipe Trevizan are partially funded by Australian Research Council Discovery Project grant DP180103446 "On-line Planning for Constrained Autonomous Agents in an Uncertain World."

Appendix A. Coverage and Cost Results on Test Domains

Problem	ASNet				LRTDP		SSiPP	
	-	PE	Adm.	No h.	h-add	LM-cut	h-add	LM-cut
cosanostra-n6	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)	30/30 (22.0 ± 0)
cosanostra-n7	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)	30/30 (25.0 ± 0)
cosanostra-n8	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)	30/30 (28.0 ± 0)
cosanostra-n9	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)	30/30 (31.0 ± 0)
cosanostra-n10	30/30 (34.0 ± 0)	30/30 (34.0 ± 0)	30/30 (34.0 ± 0)	30/30 (34.0 ± 0)	30/30 (34.0 ± 0)	11/30 (34.0 ± 0)	30/30 (34.0 ± 0)	11/30 (34.0 ± 0)
cosanostra-n11	30/30 (37.0 ± 0)	30/30 (37.0 ± 0)	30/30 (37.0 ± 0)	30/30 (37.0 ± 0)	30/30 (37.0 ± 0)	4/30 (37.0 ± 0)	23/30 (37.0 ± 0)	4/30 (37.0 ± 0)
cosanostra-n12	30/30 (40.0 ± 0)	30/30 (40.0 ± 0)	30/30 (40.0 ± 0)	30/30 (40.0 ± 0)	13/30 (40.0 ± 0)	1/30 (40.0)	10/30 (40.0 ± 0)	2/30 (40.0 ± 0)
cosanostra-n13	30/30 (43.0 ± 0)	30/30 (43.0 ± 0)	30/30 (43.0 ± 0)	-	6/30 (43.0 ± 0)	-	4/30 (43.0 ± 0)	-
cosanostra-n14	30/30 (46.0 ± 0)	30/30 (46.0 ± 0)	30/30 (46.0 ± 0)	-	2/30 (46.0 ± 0)	-	2/30 (46.0 ± 0)	-
cosanostra-n15	30/30 (49.0 ± 0)	30/30 (49.0 ± 0)	30/30 (49.0 ± 0)	-	1/30 (49.0)	-	-	-
cosanostra-n20	30/30 (64.0 ± 0)	30/30 (64.0 ± 0)	30/30 (64.0 ± 0)	-	-	-	-	-
cosanostra-n25	30/30 (79.0 ± 0)	30/30 (79.0 ± 0)	30/30 (79.0 ± 0)	-	-	-	-	-
cosanostra-n30	30/30 (94.0 ± 0)	29/30 (94.0 ± 0)	30/30 (94.0 ± 0)	-	-	-	-	-
cosanostra-n35	30/30 (109.0 ± 0)	30/30 (109.0 ± 0)	30/30 (109.0 ± 0)	-	-	-	-	-
cosanostra-n40	30/30 (124.0 ± 0)	30/30 (124.0 ± 0)	30/30 (124.0 ± 0)	-	-	-	-	-
cosanostra-n45	30/30 (139.0 ± 0)	30/30 (139.0 ± 0)	30/30 (139.0 ± 0)	-	-	-	-	-
cosanostra-n50	30/30 (154.0 ± 0)	30/30 (154.0 ± 0)	30/30 (154.0 ± 0)	-	-	-	-	-

Table 3: Results for ASNets and probabilistic baseline planners on CosaNostra Pizza. For each planner, we report the fraction of 30 rollouts that reached the goal, along with the mean trajectory cost (and 95% confidence interval bounds) for successful rollouts. The number in each problem name denotes the number of toll booths between the pizza shop and the customer.

Problem	ASNet				LRTDP		SSiPP	
	-	PE	Adm.	No h.	h-add	LM-cut	h-add	LM-cut
ex-bw-n11-s0	15/30 (28.0 ± 0)	9/30 (36.7 ± 9.8)	15/30 (28.0 ± 0)	17/30 (28.9 ± 0.5)	21/30 (44.4 ± 9.4)	28/30 (45.0 ± 7.3)	1/30 (32.0)	4/30 (66.0 ± 55.2)
ex-bw-n11-s1	-	3/30 (88.0 ± 86.2)	-	-	4/30 (99.5 ± 29.7)	-	-	1/30 (64.0)
ex-bw-n11-s2	16/30 (26.0 ± 0)	17/30 (30.2 ± 1.6)	16/30 (26.0 ± 0)	-	23/30 (28.4 ± 1.1)	2/30 (24.0 ± 0)	1/30 (28.0)	1/30 (24.0)
ex-bw-n12-s0	29/30 (26.0 ± 0)	30/30 (26.3 ± 0.3)	29/30 (26.0 ± 0)	29/30 (26.0 ± 0)	29/30 (28.2 ± 1.5)	9/30 (26.0 ± 0)	28/30 (27.7 ± 0.9)	13/30 (26.9 ± 1.2)
ex-bw-n12-s1	-	4/30 (65.0 ± 32.9)	3/30 (56.0 ± 0)	-	1/30 (52.0)	6/30 (67.0 ± 33.0)	-	-
ex-bw-n12-s2	-	5/30 (76.4 ± 32.1)	5/30 (42.4 ± 13.8)	-	1/30 (42.0)	-	-	-
ex-bw-n13-s0	26/30 (25.6 ± 1.1)	24/30 (27.4 ± 1.5)	29/30 (25.2 ± 0.4)	27/30 (25.3 ± 0.4)	27/30 (27.9 ± 0.8)	26/30 (24.0 ± 0)	27/30 (28.1 ± 1.1)	21/30 (24.0 ± 0)
ex-bw-n13-s1	-	2/30 (56.0 ± 203.3)	7/30 (60.3 ± 23.1)	-	9/30 (79.3 ± 22.6)	-	-	-
ex-bw-n13-s2	11/30 (58.7 ± 10.2)	2/30 (57.0 ± 63.5)	11/30 (49.1 ± 12.9)	-	14/30 (63.1 ± 15.9)	1/30 (82.0)	-	-
ex-bw-n14-s0	-	5/30 (64.8 ± 17.3)	-	-	-	-	-	-
ex-bw-n14-s1	-	6/30 (62.0 ± 20.7)	15/30 (34.0 ± 0)	-	-	-	-	-
ex-bw-n14-s2	12/30 (38.0 ± 0)	12/30 (40.7 ± 1.9)	14/30 (36.0 ± 0)	12/30 (38.0 ± 0)	10/30 (55.4 ± 10.1)	1/30 (46.0)	1/30 (80.0)	1/30 (68.0)
ex-bw-n15-s0	6/30 (41.3 ± 1.1)	9/30 (45.8 ± 3.0)	7/30 (48.3 ± 4.9)	-	-	-	-	-
ex-bw-n15-s1	6/30 (69.3 ± 18.7)	2/30 (61.0 ± 38.1)	3/30 (56.0 ± 9.9)	-	-	-	-	-
ex-bw-n15-s2	18/30 (56.7 ± 5.8)	14/30 (54.1 ± 16.6)	13/30 (51.2 ± 3.5)	14/30 (36.0 ± 0)	5/30 (73.6 ± 28.7)	-	1/30 (66.0)	1/30 (96.0)
ex-bw-n16-s0	-	13/30 (69.8 ± 17.7)	14/30 (53.0 ± 12.2)	-	-	-	-	-
ex-bw-n16-s1	4/30 (61.0 ± 20.2)	4/30 (69.0 ± 19.7)	6/30 (46.0 ± 0)	-	-	-	1/30 (46.0)	-
ex-bw-n16-s2	1/30 (78.0)	3/30 (70.7 ± 30.4)	3/30 (62.0 ± 8.6)	-	-	-	-	-
ex-bw-n17-s0	1/30 (76.0)	5/30 (62.4 ± 15.4)	3/30 (52.7 ± 20.1)	3/30 (80.0 ± 117.5)	1/30 (42.0)	-	-	-
ex-bw-n17-s1	7/30 (67.4 ± 30.3)	2/30 (66.0 ± 76.2)	4/30 (101.0 ± 84.9)	-	-	-	-	-
ex-bw-n17-s2	1/30 (64.0)	2/30 (93.0 ± 165.2)	-	-	-	-	-	-
ex-bw-n18-s0	-	2/30 (76.0 ± 25.4)	4/30 (70.5 ± 8.0)	-	-	-	-	-
ex-bw-n18-s1	2/30 (53.0 ± 12.7)	4/30 (71.5 ± 22.0)	2/30 (72.0 ± 25.4)	-	-	-	-	-
ex-bw-n18-s2	-	1/30 (104.0)	-	-	-	-	-	-
ex-bw-n19-s0	-	8/30 (99.0 ± 13.3)	2/30 (83.0 ± 368.5)	-	-	-	-	-
ex-bw-n19-s1	-	1/30 (96.0)	-	-	-	-	-	-
ex-bw-n19-s2	10/30 (57.8 ± 1.3)	8/30 (63.2 ± 4.6)	17/30 (52.0 ± 0)	-	-	-	-	-
ex-bw-n20-s0	1/30 (70.0)	2/30 (78.0 ± 254.1)	6/30 (79.7 ± 27.1)	4/30 (81.5 ± 18.8)	-	-	-	-
ex-bw-n20-s1	-	-	5/30 (76.0 ± 12.0)	-	-	-	-	-
ex-bw-n20-s2	-	1/30 (86.0)	-	-	-	-	-	-

Table 4: Experiment results for Exploding Blocksworld. The first number in each problem name denotes the number of blocks.

Problem	ASNet				LRTDP		SSiPP	
	-	PE	Adm.	No h.	h-add	LM-cut	h-add	LM-cut
prob-bw-b15-es1	30/30 (49.4 ± 2.2)	30/30 (53.8 ± 2.7)	30/30 (48.9 ± 1.8)	-	30/30 (52.9 ± 2.5)	-	20/30 (110.4 ± 23.0)	1/30 (241.0)
prob-bw-b15-es2	30/30 (42.2 ± 2.1)	30/30 (48.2 ± 2.8)	30/30 (42.8 ± 2.0)	30/30 (44.2 ± 2.1)	30/30 (48.3 ± 2.3)	-	30/30 (114.1 ± 21.5)	-
prob-bw-b15-es3	30/30 (66.0 ± 2.2)	30/30 (67.3 ± 2.2)	30/30 (63.8 ± 2.2)	30/30 (66.0 ± 2.2)	30/30 (74.6 ± 2.7)	-	30/30 (70.7 ± 3.0)	-
prob-bw-b15-es4	30/30 (48.6 ± 2.0)	30/30 (49.2 ± 2.0)	30/30 (48.4 ± 2.2)	30/30 (48.6 ± 2.0)	30/30 (58.3 ± 4.2)	-	19/30 (107.9 ± 20.7)	1/30 (168.0)
prob-bw-b15-es5	30/30 (52.3 ± 2.5)	30/30 (54.6 ± 2.5)	30/30 (46.1 ± 2.2)	30/30 (54.8 ± 2.4)	30/30 (55.1 ± 1.5)	-	19/30 (140.8 ± 29.4)	-
prob-bw-b20-es1	30/30 (73.5 ± 2.4)	30/30 (77.1 ± 3.0)	30/30 (73.5 ± 2.3)	30/30 (72.3 ± 2.4)	12/30 (81.1 ± 6.9)	-	1/30 (91.0)	-
prob-bw-b20-es2	30/30 (68.5 ± 2.7)	30/30 (76.1 ± 2.5)	30/30 (69.2 ± 2.3)	30/30 (68.4 ± 2.6)	30/30 (77.0 ± 2.9)	-	1/30 (79.0)	1/30 (89.0)
prob-bw-b20-es3	30/30 (68.7 ± 2.0)	30/30 (71.0 ± 3.2)	30/30 (68.7 ± 2.5)	30/30 (69.3 ± 2.5)	30/30 (78.2 ± 2.6)	-	4/30 (74.2 ± 8.5)	-
prob-bw-b20-es4	30/30 (76.3 ± 2.5)	30/30 (80.3 ± 2.7)	30/30 (73.1 ± 2.7)	30/30 (76.2 ± 2.4)	28/30 (87.6 ± 3.9)	-	1/30 (73.0)	-
prob-bw-b20-es5	30/30 (84.2 ± 2.4)	30/30 (84.7 ± 2.4)	30/30 (83.0 ± 2.8)	30/30 (84.2 ± 2.4)	19/30 (85.4 ± 3.7)	-	12/30 (85.1 ± 3.2)	-
prob-bw-b25-es1	30/30 (90.4 ± 2.7)	30/30 (101.4 ± 3.4)	30/30 (93.7 ± 2.5)	30/30 (90.0 ± 2.7)	1/30 (79.0)	-	4/30 (135.2 ± 120.6)	-
prob-bw-b25-es2	30/30 (90.1 ± 2.7)	30/30 (92.7 ± 2.3)	30/30 (88.1 ± 2.5)	30/30 (89.3 ± 2.5)	-	-	1/30 (88.0)	-
prob-bw-b25-es3	30/30 (84.5 ± 2.2)	30/30 (88.8 ± 3.1)	30/30 (77.8 ± 2.6)	-	-	-	2/30 (106.5 ± 184.2)	-
prob-bw-b25-es4	30/30 (101.0 ± 3.1)	30/30 (100.9 ± 3.3)	30/30 (91.9 ± 2.9)	30/30 (95.6 ± 2.6)	2/30 (106.0 ± 76.2)	-	1/30 (162.0)	-
prob-bw-b25-es5	30/30 (91.7 ± 2.3)	30/30 (93.7 ± 2.5)	30/30 (89.4 ± 2.6)	30/30 (90.5 ± 2.8)	-	-	-	-
prob-bw-b30-es1	30/30 (106.5 ± 3.4)	30/30 (116.4 ± 3.6)	30/30 (102.7 ± 2.5)	30/30 (109.4 ± 3.0)	-	-	-	-
prob-bw-b30-es2	30/30 (101.4 ± 2.3)	30/30 (108.5 ± 3.1)	30/30 (103.7 ± 3.4)	30/30 (106.2 ± 2.7)	-	-	1/30 (148.0)	-
prob-bw-b30-es3	30/30 (111.4 ± 2.6)	30/30 (121.9 ± 3.4)	30/30 (110.0 ± 3.3)	30/30 (111.4 ± 2.8)	1/30 (135.0)	-	-	-
prob-bw-b30-es4	30/30 (111.5 ± 3.5)	30/30 (115.2 ± 3.8)	30/30 (108.6 ± 2.6)	30/30 (113.5 ± 3.0)	-	-	1/30 (190.0)	-
prob-bw-b30-es5	30/30 (104.6 ± 3.5)	30/30 (110.4 ± 3.2)	30/30 (104.8 ± 3.2)	30/30 (111.3 ± 2.4)	-	-	-	-
prob-bw-b35-es1	30/30 (131.6 ± 3.8)	30/30 (140.3 ± 4.5)	30/30 (132.3 ± 3.4)	30/30 (132.8 ± 3.1)	-	-	-	-
prob-bw-b35-es2	30/30 (137.9 ± 3.8)	30/30 (147.8 ± 3.9)	30/30 (135.1 ± 4.1)	30/30 (143.7 ± 3.2)	-	-	-	-
prob-bw-b35-es3	30/30 (120.2 ± 3.0)	30/30 (131.4 ± 4.2)	30/30 (115.7 ± 3.0)	30/30 (125.1 ± 3.0)	-	-	-	-
prob-bw-b35-es4	30/30 (121.3 ± 2.8)	30/30 (131.5 ± 3.7)	30/30 (123.6 ± 3.4)	30/30 (125.3 ± 3.2)	-	-	-	-
prob-bw-b35-es5	30/30 (133.6 ± 3.5)	30/30 (138.5 ± 3.6)	30/30 (134.4 ± 3.1)	30/30 (130.3 ± 3.3)	-	-	-	-
prob-bw-b40-es1	30/30 (127.0 ± 3.5)	30/30 (140.9 ± 5.1)	30/30 (123.6 ± 2.9)	30/30 (133.8 ± 3.0)	-	-	-	-
prob-bw-b40-es2	30/30 (147.1 ± 3.8)	30/30 (153.4 ± 3.1)	30/30 (144.4 ± 3.3)	30/30 (149.9 ± 3.7)	-	-	-	-
prob-bw-b40-es3	30/30 (146.8 ± 4.1)	30/30 (160.0 ± 4.1)	30/30 (142.1 ± 3.8)	30/30 (154.0 ± 4.2)	-	-	-	-
prob-bw-b40-es4	30/30 (152.2 ± 3.4)	30/30 (155.2 ± 4.1)	30/30 (144.8 ± 3.3)	30/30 (151.8 ± 4.3)	-	-	-	-
prob-bw-b40-es5	30/30 (144.9 ± 3.5)	30/30 (149.5 ± 3.9)	30/30 (143.1 ± 3.0)	30/30 (145.8 ± 4.2)	-	-	-	-

Table 5: Results on Probabilistic Blocksworld. The first number in each problem name denotes the number of blocks.

Problem	ASNet				LRTDP		SSiPP	
	-	PE	Adm.	No h.	h-add	LM-cut	h-add	LM-cut
triangle-tire-4	30/30 (23.4 ± 0.7)	30/30 (23.4 ± 0.7)	30/30 (23.4 ± 0.7)	30/30 (23.4 ± 0.7)	30/30 (23.7 ± 0.7)	30/30 (23.8 ± 0.6)	30/30 (24.0 ± 0.7)	30/30 (23.2 ± 0.7)
triangle-tire-5	30/30 (28.9 ± 0.8)	30/30 (28.9 ± 0.8)	30/30 (28.9 ± 0.8)	30/30 (28.9 ± 0.8)	30/30 (30.7 ± 0.8)	19/30 (29.1 ± 1.1)	29/30 (30.8 ± 0.9)	30/30 (30.2 ± 0.9)
triangle-tire-6	30/30 (34.9 ± 0.9)	30/30 (34.9 ± 0.9)	30/30 (34.9 ± 0.9)	30/30 (34.9 ± 0.9)	1/30 (34.0)	1/30 (34.0)	30/30 (38.0 ± 0.8)	30/30 (35.7 ± 0.8)
triangle-tire-7	30/30 (40.8 ± 0.9)	30/30 (40.8 ± 0.9)	30/30 (40.8 ± 0.9)	30/30 (40.8 ± 0.9)	1/30 (42.0)	1/30 (45.0)	27/30 (45.5 ± 1.0)	16/30 (41.7 ± 1.3)
triangle-tire-8	30/30 (46.8 ± 1.1)	30/30 (46.8 ± 1.1)	30/30 (46.8 ± 1.1)	30/30 (46.8 ± 1.1)	1/30 (53.0)	1/30 (50.0)	15/30 (51.7 ± 1.2)	3/30 (48.3 ± 7.6)
triangle-tire-9	30/30 (52.9 ± 1.3)	30/30 (52.9 ± 1.3)	30/30 (52.9 ± 1.3)	30/30 (52.9 ± 1.3)	1/30 (59.0)	-	3/30 (60.7 ± 3.8)	1/30 (55.0)
triangle-tire-10	30/30 (59.0 ± 1.1)	30/30 (59.0 ± 1.1)	30/30 (59.0 ± 1.1)	30/30 (59.0 ± 1.1)	1/30 (69.0)	-	1/30 (67.0)	-
triangle-tire-11	30/30 (64.8 ± 1.1)	30/30 (64.8 ± 1.1)	30/30 (64.8 ± 1.1)	30/30 (64.8 ± 1.1)	1/30 (75.0)	-	1/30 (77.0)	-
triangle-tire-12	30/30 (71.1 ± 1.2)	30/30 (71.1 ± 1.2)	30/30 (71.1 ± 1.2)	30/30 (71.1 ± 1.2)	-	-	1/30 (79.0)	-
triangle-tire-13	30/30 (76.9 ± 1.2)	30/30 (76.9 ± 1.2)	30/30 (76.9 ± 1.2)	30/30 (76.9 ± 1.2)	-	-	-	-
triangle-tire-14	30/30 (82.8 ± 1.3)	30/30 (82.8 ± 1.3)	30/30 (82.8 ± 1.3)	30/30 (82.8 ± 1.3)	-	-	-	-
triangle-tire-15	30/30 (88.7 ± 1.4)	30/30 (88.7 ± 1.4)	30/30 (88.7 ± 1.4)	30/30 (88.7 ± 1.4)	-	-	-	-
triangle-tire-16	30/30 (94.8 ± 1.3)	30/30 (94.8 ± 1.3)	30/30 (94.8 ± 1.3)	30/30 (94.8 ± 1.3)	-	-	-	-
triangle-tire-17	30/30 (100.8 ± 1.2)	30/30 (100.8 ± 1.2)	30/30 (100.8 ± 1.2)	30/30 (100.8 ± 1.2)	-	-	-	-
triangle-tire-18	30/30 (106.5 ± 1.4)	30/30 (106.5 ± 1.4)	30/30 (106.5 ± 1.4)	30/30 (106.5 ± 1.4)	-	-	-	-
triangle-tire-19	30/30 (112.5 ± 1.6)	30/30 (112.5 ± 1.6)	30/30 (112.5 ± 1.6)	30/30 (112.5 ± 1.6)	-	-	-	-
triangle-tire-20	30/30 (118.4 ± 1.5)	30/30 (118.4 ± 1.5)	30/30 (118.4 ± 1.5)	30/30 (118.4 ± 1.5)	-	-	-	-

Table 6: Results for both ASNets and probabilistic baseline planners on Triangle Tireworld. Problems are numbered using the same convention as Little and Thiébaux (2007).

Problem	ASNet			A*		GBF	LAMA	
	-	PE	No h.	LM-cut	LM-count	LM-cut	-2011	-first
blocks-n35-s1	110	119 (10/10)	108	-	-	-	138	138
blocks-n35-s2	108	120.2 (10/10)	112	-	-	-	108	162
blocks-n35-s3	98	111.6 (10/10)	100	-	-	-	108	170
blocks-n35-s4	110	125.2 (10/10)	120	-	-	-	134	228
blocks-n35-s5	114	133.6 (10/10)	108	-	-	-	146	166
blocks-n35-s6	102	111 (10/10)	102	-	-	-	110	110
blocks-n35-s7	96	97 (10/10)	96	-	-	262	102	118
blocks-n35-s8	108	114.8 (10/10)	106	-	-	290	114	160
blocks-n35-s9	106	125.8 (10/10)	114	-	-	-	114	116
blocks-n35-s10	106	113.6 (10/10)	104	-	-	-	116	116
blocks-n50-s1	158	181 (10/10)	-	-	-	-	-	-
blocks-n50-s2	166	199.8 (10/10)	-	-	-	-	-	-
blocks-n50-s3	164	198 (10/10)	-	-	-	-	216	240
blocks-n50-s4	160	185 (10/10)	162	-	-	-	-	-
blocks-n50-s5	158	182.6 (10/10)	170	-	-	-	240	278
blocks-n50-s6	134	155.6 (10/10)	-	-	-	-	202	202
blocks-n50-s7	162	182.6 (10/10)	160	-	-	-	-	-
blocks-n50-s8	154	182.2 (10/10)	158	-	-	-	-	-
blocks-n50-s9	184	237 (10/10)	188	-	-	-	-	-
blocks-n50-s10	146	162.4 (10/10)	154	-	-	-	-	-
blocks-n50-s11	158	197.4 (10/10)	172	-	-	-	-	-
blocks-n50-s12	150	161 (10/10)	146	-	-	-	200	236
blocks-n50-s13	136	155.4 (10/10)	134	-	-	-	208	286
blocks-n50-s14	158	179 (10/10)	166	-	-	-	-	-
blocks-n50-s15	170	219.8 (10/10)	190	-	-	-	-	-
blocks-n50-s16	138	156.2 (10/10)	142	-	-	-	154	154
blocks-n50-s17	168	207.8 (10/10)	-	-	-	-	172	172
blocks-n50-s18	160	201.4 (10/10)	-	-	-	-	-	-
blocks-n50-s19	148	197.4 (10/10)	152	-	-	-	-	-
blocks-n50-s20	152	169.8 (10/10)	156	-	-	-	154	154

Table 7: Results for both ASNet and the baseline planners on our evaluation set of (deterministic) Blocksworld problems. Each cell shows the length of the plan returned by the corresponding planner. For ASNets in stochastic execution mode ("PE"), 10 rollouts are performed, and we report both the mean cost of successful plans (first number) and the fraction of runs that reach the goal (second number). The first number in each problem name indicates the number of blocks, and the second identifies the seed used to generate the instance.

Appendix B. Domain descriptions

This appendix contains complete verbal descriptions of each of the domains that we use for evaluation. We have also included full (P)PDDL for those domains which do not appear in previous work or are otherwise helpful for understanding our experiments.

B.1 Probabilistic domains

Probabilistic Blocksworld A simple probabilistic adaptation of the venerable Blocksworld domain. Each problem in this domain includes n blocks that can be stacked on top of one

Problem	ASNet			A*		GBF	LAMA	
	-	PE	No h.	LM-cut	LM-count	LM-cut	-2011	-first
mbw-b15-t1-s717	52	59.2 (8/10)	54	-	-	104	52	142
mbw-b15-t3-s484	38	50.7 (6/10)	-	-	38	94	38	78
mbw-b15-t3-s961	42	57 (10/10)	46	-	38	66	38	72
mbw-b20-t1-s1244	68	87.7 (6/10)	-	-	-	292	82	114
mbw-b20-t4-s1749	60	70.9 (9/10)	-	-	-	136	68	104
mbw-b20-t4-s1830	56	70 (6/10)	-	-	-	102	68	140
mbw-b25-t1-s1727	98	135.4 (7/10)	-	-	-	-	116	214
mbw-b25-t2-s1558	90	106.9 (9/10)	-	-	-	-	106	212
mbw-b25-t5-s888	66	101 (9/10)	64	-	-	-	-	-
mbw-b30-t1-s1641	172	150.8 (10/10)	-	-	-	-	-	-
mbw-b30-t5-s1551	258	122.8 (10/10)	-	-	-	-	-	-
mbw-b30-t9-s1824	74	99.5 (10/10)	-	-	-	-	86	126
mbw-b35-t1-s463	162	186.7 (9/10)	-	-	-	-	-	-
mbw-b35-t5-s766	138	165.4 (7/10)	-	-	-	-	140	156
mbw-b35-t8-s688	100	123.6 (9/10)	-	-	-	-	-	-
mbw-b40-t1-s1766	178	208.6 (10/10)	158	-	-	-	-	-
mbw-b40-t10-s305	110	133.2 (6/10)	-	-	-	-	-	-
mbw-b40-t6-s1705	108	162.3 (6/10)	-	-	-	-	-	-
mbw-b45-t1-s931	204	269.2 (5/10)	-	-	-	-	-	-
mbw-b45-t2-s782	182	230 (5/10)	-	-	-	-	-	-
mbw-b45-t6-s1396	156	220.2 (8/10)	-	-	-	-	-	-
mbw-b50-t1-s1530	-	280 (2/10)	210	-	-	-	-	-
mbw-b50-t4-s179	270	243.7 (6/10)	-	-	-	-	-	-
mbw-b50-t7-s963	156	244.3 (6/10)	-	-	-	-	180	208
mbw-b55-t1-s992	-	-	-	-	-	-	-	-
mbw-b55-t13-s653	146	219 (9/10)	152	-	-	-	-	-
mbw-b55-t7-s1529	206	-	-	-	-	-	-	-
mbw-b60-t1-s842	-	-	-	-	-	-	-	-
mbw-b60-t7-s124	184	-	206	-	-	-	-	-
mbw-b60-t7-s261	-	-	-	-	-	-	-	-

Table 8: Results for both ASNets and baseline planners on Matching Blocksworld. Although the domain is from the IPC 2008 learning track, the instances were custom-generated. Each instance name shows the number of blocks (e.g `b40`), the number of towers in the initial state and goal (e.g `t3`), and the random seed used to generate the problem (e.g `s201`).

Problem	ASNet			A*		GBF	LAMA	
	-	PE	No h.	LM-cut	LM-count	LM-cut	-2011	-first
gm-7x7-s70	43	110.4 (10/10)	-	-	-	98	45	98
gm-7x7-s71	29	29.5 (10/10)	-	28	28	-	28	100
gm-7x7-s72	29	28.7 (10/10)	-	28	28	-	28	155
gm-8x8-s80	42	45.8 (10/10)	-	-	-	-	248	248
gm-8x8-s81	36	36.1 (10/10)	-	-	-	-	-	-
gm-8x8-s82	40	40.2 (10/10)	-	-	-	-	40	51
gm-9x9-s90	42	45.7 (10/10)	-	-	-	-	42	101
gm-9x9-s91	39	38.3 (10/10)	-	-	-	-	207	207
gm-9x9-s92	36	36.6 (8/10)	-	36	-	-	36	118
gm-10x10-s100	43	43.2 (10/10)	-	-	-	-	259	259
gm-10x10-s101	-	74.2 (4/10)	-	-	-	-	-	-
gm-10x10-s102	38	38.6 (10/10)	-	-	-	-	235	235
gm-13x13-s130	77	83 (2/10)	-	-	-	-	-	-
gm-13x13-s131	-	104.4 (8/10)	-	-	-	-	-	-
gm-13x13-s132	64	65.1 (10/10)	-	-	-	-	-	-
gm-16x16-s160	71	70.8 (10/10)	-	-	-	-	-	-
gm-16x16-s161	83	85.6 (7/10)	-	-	-	-	-	-
gm-16x16-s162	75	76.3 (10/10)	-	-	-	-	-	-
gm-19x19-s190	89	90.2 (8/10)	-	-	-	-	-	-
gm-19x19-s191	80	80.4 (10/10)	-	-	-	-	-	-
gm-19x19-s192	103	111 (5/10)	-	-	-	-	-	-

Table 9: Results for both ASNet and the baseline planners on our suite of Gold Miner problems. Each problem was produced by supplying a unique random seed to the same generator used in the IPC 2008 Learning Track. The size of each grid (i.e. width × height) is shown in each problem name.

another to form towers. The agent is equipped with a gripper that can lift up blocks and deposit them either on the table, or on top of some other block. The aim is to move from the initial configuration of block towers to some specific goal configuration. In the probabilistic setting, this task is complicated by the possibility of gripper failure: each time the gripper goes to pick up a block, or place a block on top of another block, there is a 25% chance that the block held by the gripper will instead fall onto the table. Because there are no dead ends, this domain can be solved by taking the actions recommended by a classical planner for a determinisation of the problem, and re-planning whenever a gripper action fails (Little & Thiébaux, 2007). This domain is based on domains that have appeared in past rounds of the International Probabilistic Planning Competition (Younes & Littman, 2004). Our version differs from some past versions in that it lacks actions for moving *towers* of blocks; instead, only a single block at a time can be moved, just as in deterministic Blocksworld. All instances for this task (as well as Exploding and Deterministic Blocksworld) were generated by the algorithm from Slaney and Thiébaux (2001).

Exploding Blocksworld (Younes & Littman, 2004) A more challenging variant of deterministic Blocksworld which includes both avoidable and unavoidable dead ends. Unlike Probabilistic Blocksworld, there is no chance of the gripper dropping a block onto the table unless commanded to do so. Instead, the challenge comes from dealing with two additional attributes given to each block: namely, whether the block has been *destroyed*, and whether the block has been *detonated* (both of which are initially false). Once a block has been destroyed, it cannot be picked up by the gripper any more and is thus stuck in its position; once a block has been detonated, it *can* still be moved and picked up by the gripper, but it cannot detonate again. Whenever a block b_1 is placed on top of a block b_2, there is a 10% chance that b_1 will detonate and destroy b_2, thereby preventing b_2 from being moved again. When placing a block b_1 on the table, there is a 40% chance that b_1 will detonate and destroy *the table*, in which case no further blocks can be placed directly onto the table. Because being unable to place additional blocks on the table can prevent the goal from being reached, optimal policies for exploding Blocksworld often involve intentionally detonating blocks by placing them on top of other blocks that are already in their goal positions. This renders the detonated block inert and allows it to be placed on the table at no risk. Our version of this domain first appeared in IPPC 2008 (Bryce & Buffet, 2008), although we have modified it to remove a bug which allowed blocks to be stacked on top of themselves.

Triangle Tireworld (Little & Thiébaux, 2007) This domain was described in the main text of the article. A PPDDL description of the domain is given in Figure 10.

CosaNostra Pizza (Toyer et al., 2018) This domain was described in the main text of the article. A PPDDL description of the domain is given in Figure 11.

B.2 Deterministic domains

Deterministic Blocksworld Our deterministic Blocksworld variant is identical to the Probabilistic Blocksworld domain described in Section 5.2, but with a 0% probability that the gripper will drop a block.

Gold Miner (Fern et al., 2011) Each problem in this domain consists of an $n \times n$ grid in which some locations are filled with soft rock and some locations are filled with hard rock,

```
(define (domain triangle-tire)
  (:requirements :typing :strips :equality :probabilistic-effects)
  (:types location)
  (:predicates (vehicle-at ?loc - location) (spare-in ?loc - location)
               (road ?from - location ?to - location) (not-flattire))
  (:action move-car
    :parameters (?from - location ?to - location)
    :precondition (and (vehicle-at ?from) (road ?from ?to) (not-flattire))
    :effect (and (vehicle-at ?to) (not (vehicle-at ?from))
                 (probabilistic 0.5 (not (not-flattire)))))
  (:action changetire
    :parameters (?loc - location)
    :precondition (and (spare-in ?loc) (vehicle-at ?loc))
    :effect (and (not (spare-in ?loc)) (not-flattire))))
```

Figure 10: PPDDL domain for Triangle Tireworld.

and one location contains some gold (which the agent wishes to retrieve). There are also laser cannons and bombs in some cells that the agent can pick up in order to destroy the rock. A laser cannon can be used repeatedly and is able to destroy both hard and soft rock in a cell adjacent to the agent, but will also destroy any gold in that adjacent cell. A bomb can only be used once, and can only destroy soft rock in an adjacent cell, but will never destroy gold. The instances of this domain are generated in such a way that there is a simple generalised policy that solves all instance of this domain. Specifically, the agent can get a laser cannon, blast connected paths between the gold location and a bomb location, then pick up the bomb and use it to remove the last piece of rock needed to get to the goal. This domain appeared for the first time in the learning track of the 2008 International Planning Competition (Fern et al., 2011).

Matching Blocksworld (Fern et al., 2011) Another variant of the Blocksworld domain in which there are two grippers with opposite polarities. Each block is also assigned a polarity matching one of the grippers. Either gripper can be used to move either type of block. However, if a gripper of one polarity is used to move a block of a different polarity, then the block will be "damaged" so that no other block can be placed on top of it, although the damaged block itself can still be picked up and moved. A generalised policy must be able to solve arbitrary Blocksworld instances *and* avoid picking up blocks with mismatched grippers. Like Gold Miner, this domain first appeared in the learning track of the 2008 International Planning Competition.

Appendix C. Receptive Field Experiments

We experimentally verified the receptive field limitation mentioned in Section 3.5 by training ASNets of different depths on a collection of `unreliable-robot` problems with two location chains of varying length K. Specifically, we used problems with $K = 1, \ldots, 5$ locations to train an ASNet, and then tested the ASNet on those same problems. Our training strategy was almost the same as in Section 5.1. The main difference is that for the deeper, harder-

```
(define (domain cosanostra)
  (:requirements :typing :strips :probabilistic-effects :conditional-effects
                 :negative-preconditions)
  (:types location - object
          toll-booth open-intersection - location)
  (:predicates (have-pizza) (tires-intact) (deliverator-at ?l - location)
               (pizza-at ?l - location) (open ?booth - toll-booth)
               (operator-angry ?booth - toll-booth) (road ?from ?to - location))
  (:action load-pizza
           :parameters (?loc - location)
           :precondition (and (deliverator-at ?loc) (pizza-at ?loc))
           :effect (and (not (pizza-at ?loc)) (have-pizza)))
  (:action unload-pizza
           :parameters (?loc - location)
           :precondition (and (deliverator-at ?loc) (have-pizza))
           :effect (and (pizza-at ?loc) (not (have-pizza))))
  (:action pay-operator
           :parameters (?booth - toll-booth)
           :precondition (and (deliverator-at ?booth))
           :effect (and (open ?booth)))
  (:action leave-toll-booth
           :parameters (?from - toll-booth ?to - location)
           :precondition (and (deliverator-at ?from) (tires-intact)
                              (road ?from ?to))
           :effect (and
                    (when (and (operator-angry ?from))
                          (and (probabilistic
                                0.5 (and (not (tires-intact)))
                                0.5 (and (not (deliverator-at ?from))
                                         (deliverator-at ?to)))))
                    (when (and (not (operator-angry ?from)))
                          (and (not (deliverator-at ?from))
                               (deliverator-at ?to)))
                    (when (and (not (open ?from)))
                          (and (operator-angry ?from)))))
  (:action leave-open-intersection
           :parameters (?from - open-intersection ?to - location)
           :precondition (and (deliverator-at ?from) (tires-intact)
                              (road ?from ?to))
           :effect (and (not (deliverator-at ?from)) (deliverator-at ?to))))
```

Figure 11: PPDDL domain for CosaNostra Pizza.

to-train policies (e.g four proposition layers/five action layers), we disabled regularisers to make the network converge more quickly. We also disabled skip connections, although they should not affect the results for this particular domain. Table 10 shows the results of our experiment. As expected, an ASNet with $L \geq K$ proposition layers will succeed at a length-K problem, but not at problems with $L < K$.

Proposition layers	Chain length K				
	1	2	3	4	5
1	30/30	14/30	14/30	14/30	14/30
2	30/30	30/30	14/30	14/30	14/30
3	30/30	30/30	30/30	14/30	14/30
4	30/30	30/30	30/30	30/30	14/30

Table 10: Coverage (expressed as a fraction of 30 runs) for ASNets of different depths on the receptive-field-limited class of `unreliable-robot` problems.

Appendix D. A Sparse Policy for CosaNostra Pizza

First action layer.

$$\begin{aligned}
\phi^1_{\text{leave-toll-booth}(?\mathit{from},?\mathit{to})} = \phi^1_{\text{ltb}(?f,?t)} &= \sigma(-1.31 \cdot \text{action-count}(\text{leave-toll-booth}(?\mathit{from}, ?\mathit{to})) \\
&\quad + 0.28 \cdot \text{is-goal}(\text{deliverator-at}(?\mathit{to})) + 3.22) \\
\phi^1_{\text{pay-operator}(?\mathit{loc})} = \phi^1_{\text{po}(?l)} &= \sigma(-1.10 \cdot \text{action-count}(\text{pay-operator}(?\mathit{loc})) \\
&\quad - 0.61 \cdot \text{open}(?\mathit{loc}) + 1.29)
\end{aligned}$$

First proposition layer.

$$\begin{aligned}
\psi^1_{\text{deliverator-at}(?\mathit{loc})} = \psi^1_{\text{da}(?l)} &= \sigma(0.40 \cdot \text{pool}(\phi^1_{\text{leave-toll-booth}(?\mathit{loc},\cdot)}) \\
&\quad - 1.46 \cdot \text{pool}(\phi^1_{\text{leave-toll-booth}(\cdot,?\mathit{loc})} \\
&\quad - 1.48 \cdot \text{pool}(\phi^1_{\text{pay-operator}(?\mathit{loc})}) + 4.98)
\end{aligned}$$

Second action layer: omitted, all weights zero.

Second proposition layer: merely “scales up” the first proposition layer using a skip connection.

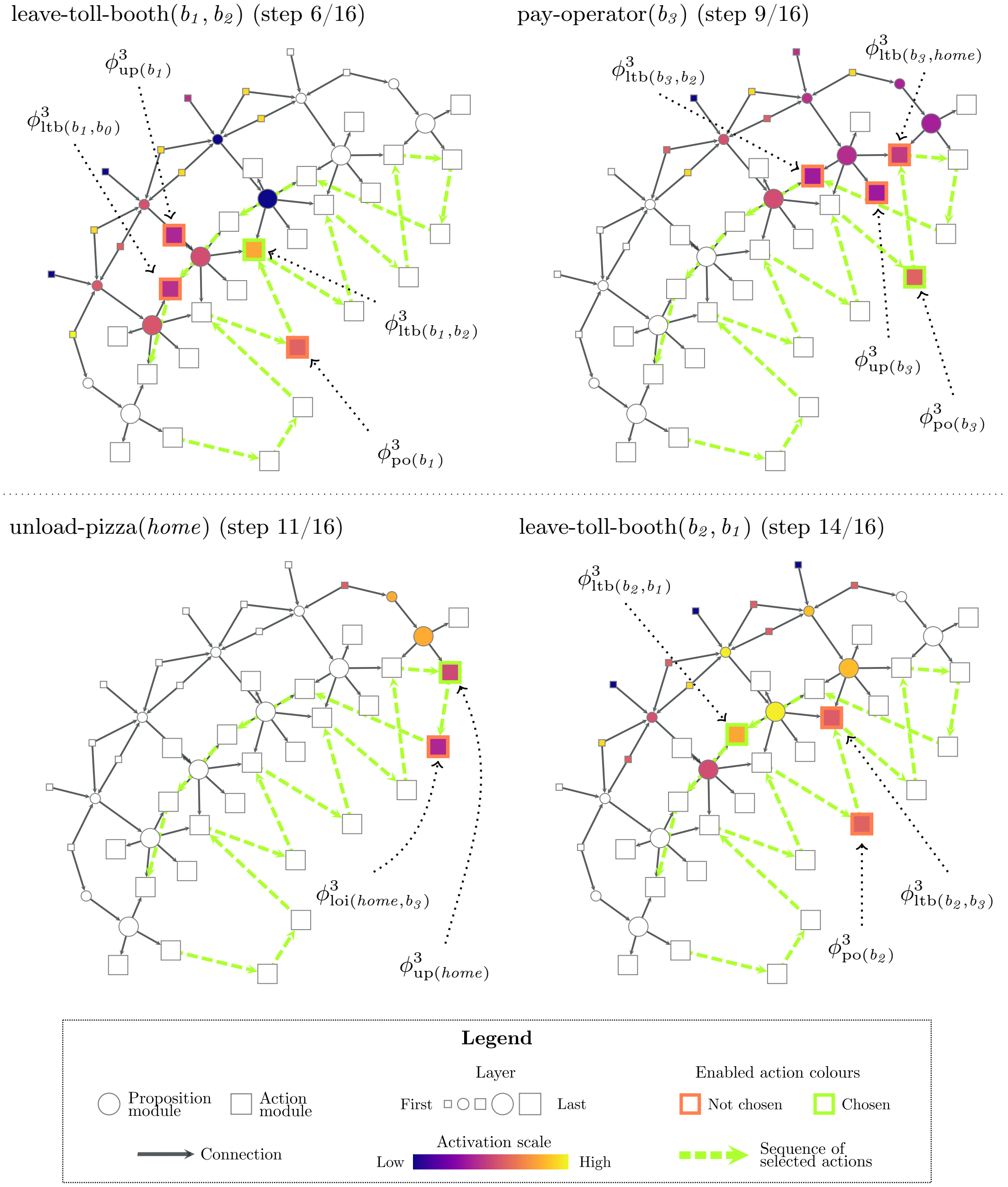
$$\psi^2_{\text{deliverator-at}(?\mathit{loc})} = \psi^2_{\text{da}(?l)} = \sigma(3.60 \cdot \psi^1_{\text{deliverator-at}(?\mathit{loc})})$$

Third action layer.

$$\begin{aligned}
\phi^3_{\text{leave-open-intersection}(?\mathit{from},?\mathit{to})} = \phi^3_{\text{loi}(?f,?t)} &= 0 \\
\phi^3_{\text{leave-toll-booth}(?\mathit{from},?\mathit{to})} = \phi^3_{\text{ltb}(?f,?t)} &= 1.25 \cdot \psi^2_{\text{deliverator-at}(?\mathit{from})} - 1.07 \cdot \psi^2_{\text{deliverator-at}(?\mathit{to})} \\
\phi^3_{\text{load-pizza}(?\mathit{loc})} = \phi^3_{\text{lp}(?l)} &= -0.65 \cdot \psi^2_{\text{deliverator-at}(?\mathit{loc})} + 4.71 \\
\phi^3_{\text{pay-operator}(?\mathit{loc})} = \phi^3_{\text{po}(?l)} &= 4.99 \\
\phi^3_{\text{unload-pizza}(?\mathit{loc})} = \phi^3_{\text{up}(?l)} &= 0.66 \cdot \psi^2_{\text{deliverator-at}(?\mathit{loc})} - 4.74
\end{aligned}$$

Figure 12: An easily-readable representation of a sparse ASNet trained for the CosaNostra Pizza domain. Weights have been rounded to two decimal places, and $\sigma(\cdot)$ has been used to denote the ELU activation function. The middle column shows abbreviations used to refer to activations when discussing their semantics in the main text (e.g. $\psi^1_{\text{da}(?l)}$).

Recall that in Section 6, we examined a sparse ASNet policy for Triangle Tireworld that was sufficiently compact to be written out as a small series of human-readable equations. We repeated the same experiment for the CosaNostra Pizza domain introduced in Section 5.2.



leave-toll-booth(b_1, b_2) (step 6/16)

pay-operator(b_3) (step 9/16)

unload-pizza(*home*) (step 11/16)

leave-toll-booth(b_2, b_1) (step 14/16)

$\phi^3_{\text{up}(b_1)}$
$\phi^3_{\text{ltb}(b_1,b_0)}$
$\phi^3_{\text{ltb}(b_1,b_2)}$
$\phi^3_{\text{po}(b_1)}$
$\phi^3_{\text{ltb}(b_3,b_2)}$
$\phi^3_{\text{ltb}(b_3,home)}$
$\phi^3_{\text{up}(b_3)}$
$\phi^3_{\text{po}(b_3)}$
$\phi^3_{\text{loi}(home,b_3)}$
$\phi^3_{\text{up}(home)}$
$\phi^3_{\text{ltb}(b_2,b_1)}$
$\phi^3_{\text{ltb}(b_2,b_3)}$
$\phi^3_{\text{po}(b_2)}$
Legend
Proposition module
Action module
Layer
First
Last
Enabled action colours
Not chosen
Chosen
Connection
Activation scale
Low
High
Sequence of selected actions

Figure 13: Activations for the sparse CosaNostra Pizza policy given in Figure 12. On the top row, the illustrated behaviours include moving in the correct direction when travelling from the shop to the customer, and paying tolls. The bottom row depicts unloading the pizza at the customer's home, and moving back from the customer to the shop. Each diagram includes labels for the final-layer modules which correspond to enabled actions.

CosaNostra Pizza is a logistics problem in which a vehicle must pick up a pizza from its starting location (a pizza shop), travel though a chain of toll booths $b_0, b_1, \ldots, b_{K-1}$ leading from a shop to a customer, deliver a pizza to the customer, then return to the pizza shop. This task is complicated by the behaviour of the toll booth operators: if the agent chooses not to pay at one of the toll booths on the way to the customer, then the toll collector may crush their car on the way back with 50% probability. The optimal policy is to pay all operators on the way to the customer, but not on the way back. This policy reaches the goal with probability 1.

Our sparse network for CosaNostra Pizza is illustrated in Figure 12 (equations) and Figure 13 (activations). We obtained this network by training ten networks with an ℓ_1 regularisation coefficient of 0.02 for one hour each, and then using the sparsest network that could successfully solve the test problems. CosaNostra is a more complex domain than Triangle Tireworld, and a successful policy must solve several distinct sub-tasks: it must be able to pick up the pizza, alternate between paying toll booth operators and moving between booths, drop off the pizza, and travel back to the shop. To keep our treatment concise, we will use Figure 12 to show that the ASNet behaves successfully at only one of these sub-tasks: choosing the correct direction to travel when moving the vehicle from the shop to the customer. We invite the reader to try a similar approach to verify other desirable properties of the network (for instance: why does it not pick up pizza again immediately after dropping it off?).

Consider the actions chosen by an ASNet while it is moving the vehicle from the shop to the customer. Specifically, say that it is at a booth b_k in a chain $b_0, b_1, \ldots, b_{K-1}$, where $0 < k < K-1$ (i.e. it is not at the first or last booth). After paying the toll at booth b_k, the ASNet faces four choices: it can pay the toll booth operator again; it can unload the pizza; or it can take one of two movement actions, one of which leads to the customer and one of which leads back to the shop. We will focus on the last two options: when does the ASNet choose the action that moves forward to booth b_{k+1} over the action that moves back to booth b_{k-1}? Formally, leave-toll-booth(b_k, b_{k+1}) will be chosen over leave-toll-booth(b_k, $b_{k\text{-}1}$) when the corresponding final-layer activations satisfy $\phi^3_{\mathrm{ltb}(b_k, b_{k+1})} > \phi^3_{\mathrm{ltb}(b_k, b_{k\text{-}1})}$ (recall our shorthand of using, e.g., *ltb* in place of *leave-toll-booth*). We can simplify this condition by substituting in a series of definitions from Figure 12:

$$\phi^3_{\mathrm{ltb}(b_k, b_{k+1})} > \phi^3_{\mathrm{ltb}(b_k, b_{k\text{-}1})}$$

$$\begin{array}{c} 1.25 \cdot \psi^2_{\mathrm{da}(b_k)} \\ -\ 1.07 \cdot \psi^2_{\mathrm{da}(b_{k+1})} \end{array} > \begin{array}{c} 1.25 \cdot \psi^2_{\mathrm{da}(b_k)} \\ -\ 1.07 \cdot \psi^2_{\mathrm{da}(b_{k\text{-}1})} \end{array} \qquad \text{(expand definition)}$$

$$-1.07 \cdot \psi^2_{\mathrm{da}(b_{k+1})} > -1.07 \cdot \psi^2_{\mathrm{da}(b_{k\text{-}1})} \qquad \text{(simplify)}$$

$$\psi^2_{\mathrm{da}(b_{k+1})} < \psi^2_{\mathrm{da}(b_{k\text{-}1})} \qquad \text{(flip sign)}$$

$$\psi^1_{\mathrm{da}(b_{k+1})} < \psi^1_{\mathrm{da}(b_{k\text{-}1})} \qquad \text{(expand, remove } \sigma(\cdot)\text{)}$$

$$\begin{array}{l} 0.40 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{ltb}(b_{k+1},\cdot)}) \\ \quad -\ 1.46 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{ltb}(\cdot,b_{k+1})} \\ \quad -\ 1.48 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{po}(b_{k+1})}) \end{array} < \begin{array}{l} 0.40 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{ltb}(b_{k-1},\cdot)}) \\ \quad -\ 1.46 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{ltb}(\cdot,b_{k-1})} \\ \quad -\ 1.48 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{po}(b_{k-1})}) \end{array} \qquad \text{(expand, remove } \sigma(\cdot))$$

Crucially we have used the fact that the ELU activation function, given by $\sigma(x) = x$ when $x \geq 0$ and $\sigma(x) = \exp(x) - 1$ when $x \leq 0$, is strictly increasing. Consequently, we have $\sigma(a) > \sigma(b)$ iff $a > b$.

To help simplify the last inequality, recall from Figure 12 that the leave-toll-booth modules in the first action layer are defined as

$$\phi^{1}_{\text{leave-toll-booth}(?\mathit{from},?\mathit{to})} = \sigma(-1.31 \cdot \text{action-count}(\text{leave-toll-booth}(?\mathit{from}, ?\mathit{to}))\ .$$

This will take the value $\sigma(-1.31) < 0$ if leave-toll-booth($?\mathit{from}, ?\mathit{to}$) has ever been executed, and $\sigma(0) = 0$ otherwise. Observe that all of the max-pooling operations over $\phi^{1}_{\mathrm{ltb}(?f,?t)}$ modules will be pooling over a module for at least one leave-toll-booth action that has not already been executed. For instance, $\mathrm{pool}(\phi^{1}_{\mathrm{ltb}(\cdot,b_{k-1})})$ pools over $\phi^{1}_{\mathrm{ltb}(b_k,b_{k-1})}$, but leave-toll-booth($b_k, b_{k-1}$) will not have been executed yet (under normal execution of an otherwise-optimal policy). Hence, we can replace those pool operations with zero values, and only consider the pooling operations over modules for pay-operator (po) actions, giving us:

$$\begin{array}{rcl} -1.48 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{po}(b_{k+1})}) & < & -1.48 \cdot \mathrm{pool}(\phi^{1}_{\mathrm{po}(b_{k-1})}) \\ \phi^{1}_{\mathrm{po}(b_{k+1})} & > & \phi^{1}_{\mathrm{po}(b_{k-1})} \\ \begin{array}{l} -\ 1.10 \cdot \text{action-count}(\text{pay-operator}(b_{k+1})) \\ \quad -\ 0.61 \cdot \text{open}(b_{k+1}) \end{array} & > & \begin{array}{l} -\ 1.10 \cdot \text{action-count}(\text{pay-operator}(b_{k-1})) \\ \quad -\ 0.61 \cdot \text{open}(b_{k-1}) \end{array} \end{array}$$

In the second step we flipped the sign and then dropped the pool operator, which we are justified in doing because each toll booth has exactly one related pay-operator action. In the third step we substituted in the definition of $\phi^{1}_{\mathrm{po}(?l)}$, then dropped the activation and bias, since $u \mapsto \sigma(u + c)$ is an increasing function of u for any constant bias c.

In our scenario, the agent is travelling from the shop to the customer, and is between two toll booths b_{k-1} and b_{k+1}. At this point it will have already paid the toll operator at b_{k-1}, and so booth b_{k-1} will be open and the action count for the corresponding action will be one. It has not visited the next booth b_{k+1}, so open(b_{k+1}) will be false and the corresponding action count will be zero. Substituting these values in the last inequality, we obtain

$$-1.10 \cdot 0 - 0.61 \cdot 0 = 0 > -1.10 \cdot 1 - 0.61 \cdot 1 = -2.81\ ,$$

which is of course true. This proves that the agent will correctly choose leave-toll-booth(b_k, b_{k+1}) over leave-toll-booth(b_k, b_{k-1}) when at booth b_k somewhere in a chain $b_1, b_2, \ldots b_{K-1}$ (for $0 < k < K - 1$).

References

Asai, M., & Fukunaga, A. (2018). Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Bajpai, A. N., Garg, S., et al. (2018). Transfer of deep reactive policies for mdp planning. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*.

Bertsekas, D., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Blockeel, H., & de Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*.

Bonet, B., Frances, G., & Geffner, H. (2019). Learning features and abstract actions for computing generalized plans. In *AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 33, pp. 2703–2710.

Bonet, B., & Geffner, H. (2003). Labeled RTDP: Improving the convergence of real-time dynamic programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Bonet, B., & Geffner, H. (2018). Features, projections, and representation change for generalized planning. *arXiv preprint arXiv:1801.10055*.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*.

Bryce, D., & Buffet, O. (2008). 6th International Planning Competition: Uncertainty part. In *Workshop on the International Planning Competition*.

Buffet, O., & Aberdeen, D. (2009). The factored policy-gradient planner. *Artificial Intelligence*.

Camacho, E. F., & Alba, C. B. (2007). *Model predictive control*. Springer Science & Business Media.

Coles, A., Coles, A., Olaya, A. G., Jiménez, S., López, C. L., Sanner, S., & Yoon, S. (2012). A survey of the seventh international planning competition. *AI Magazine*.

Culberson, J. (1997). Sokoban is PSPACE-complete. https://webdocs.cs.ualberta.ca/~joe/Preprints/Sokoban/paper.html.

de la Rosa, T., & Fuentetaja, R. (2017). Bagging strategies for learning planning policies. *Annals of Mathematics and Artificial Intelligence*.

de la Rosa, T., Jiménez, S., Fuentetaja, R., & Borrajo, D. (2011). Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research (JAIR)*.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *Multiple classifier systems*.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Ferber, P., Helmert, M., & Hoffman, J. (2020). Neural network heuristics for classical planning: A study of hyperparameter space. In *ECAI*.

Fern, A., Khardon, R., & Tadepalli, P. (2011). The first learning track of the international planning competition. *Machine Learning*.

Fern, A., Yoon, S., & Givan, R. (2004). Approximate policy iteration with a policy language bias. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Francès, G., Corrêa, A. B., Geissmann, C., & Pommerening, F. (2019). Generalized potential heuristics for classical planning.. International Joint Conferences on Artificial Intelligence (IJCAI).

Garg, S., Bajpai, A., & Mausam (2019). Size-independent neural transfer for rddl planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Geffner, H. (2018a). International Joint Conferences on Artificial Intelligence (IJCAI)-ECAI keynote: Model-free, model-based, and general intelligence.. Recording at `https://www.youtube.com/watch?v=g3lc8BxTPiU&t=1906`.

Geffner, H. (2018b). Model-free, model-based, and general intelligence. *arXiv:1806.02308*.

Gomoluch, P., Alrajeh, D., & Russo, A. (2019). Learning classical planning strategies with policy gradient. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Gretton, C., & Thiébaux, S. (2004). Exploiting first-order regression in inductive policy selection. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Groshev, E., Goldstein, M., Tamar, A., Srivastava, S., & Abbeel, P. (2018). Learning generalized reactive policies using deep neural networks. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Haslum, P., & Geffner, H. (2000). Admissible heuristics for optimal planning. In *International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research (JAIR)*.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: what's the difference anyway?. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

Hoffmann, J. (2001). FF: The Fast-Forward planning system. *AI Magazine*.

Hu, Y., & De Giacomo, G. (2011). Generalized planning: Synthesizing plans that work for multiple environments. In *International Joint Conferences on Artificial Intelligence (IJCAI)*.

Issakkimuthu, M., Fern, A., & Tadepalli, P. (2018). Training deep reactive policies for probabilistic planning problems. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Jain, A., Zamir, A. R., Savarese, S., & Saxena, A. (2016). Structural-RNN: Deep learning on spatio-temporal graphs. In *Computer Vision and Pattern Recognition (CVPR)*.

Jiménez, S., de la Rosa, T., Fernández, S., Fernández, F., & Borrajo, D. (2012). A review of machine learning for automated planning. *The Knowledge Engineering Review*.

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

Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer-Aided Verification (CAV)*.

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

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv:1807.05118*.

Lipton, Z. C. (2016). The mythos of model interpretability. *arXiv:1606.03490*.

Little, I., & Thiébaux, S. (2007). Probabilistic planning vs. replanning. In *International Conference on Automated Planning and Scheduling (ICAPS) workshops*.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR)*.

Martin, M., & Geffner, H. (2000). Learning generalized policies in planning using concept languages. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

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

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

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

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*.

Ratner, D., & Warmuth, M. K. (1986). Finding a shortest solution for the n× n extension of the 15-puzzle is intractable.. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Richter, S., Westphal, M., & Helmert, M. (2011). LAMA 2008 and 2011. In *International Planning Competition*, pp. 117–124.

Rivest, R. L. (1987). Learning decision lists. *Machine learning.*

Ross, S., Gordon, G., & Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS.*

Say, B., Wu, G., Zhou, Y. Q., & Sanner, S. (2017). Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *International Joint Conferences on Artificial Intelligence (IJCAI).*

Shen, W., Trevizan, F., & Thiébaux, S. (2020). Learning domain-independent planning heuristics with hypergraph networks. *International Conference on Automated Planning and Scheduling (ICAPS).*

Shen, W., Trevizan, F. W., Toyer, S., Thiébaux, S., & Xie, L. (2019). Guiding search with generalized policies for probabilistic planning. In *Symposium on Combinatorial Search (SOCS).*

Sievers, S., Katz, M., Sohrabi, S., Samulowitz, H., & Ferber, P. (2019). Deep learning for cost-optimal planning: Task-dependent planner selection. In *AAAI Conference on Artificial Intelligence (AAAI).*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature.*

Slaney, J., & Thiébaux, S. (2001). Blocks world revisited. *Artificial Intelligence.*

Sourek, G., Aschenbrenner, V., Zelezny, F., Schockaert, S., & Kuzelka, O. (2018). Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research (JAIR).*

Srivastava, S., Immerman, N., Zilberstein, S., & Zhang, T. (2011). Directed search for generalized plans using classical planners.. In *International Conference on Automated Planning and Scheduling (ICAPS).*

Tamar, A., Wu, Y., Thomas, G., Levine, S., & Abbeel, P. (2016). Value iteration networks. In *Conference on Neural Information Processing Systems (NeurIPS).*

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Royal Stat. Soc.: Series B (Methodological).*

Tjeng, V., Xiao, K. Y., & Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR).*

Toyer, S., Trevizan, F., Thiébaux, S., & Xie, L. (2018). Action schema networks: Generalised policies with deep learning. In *AAAI Conference on Artificial Intelligence (AAAI).*

Trevizan, F., Thiébaux, S., & Haslum, P. (2017). Occupation measure heuristics for probabilistic planning. In *International Conference on Automated Planning and Scheduling (ICAPS).*

Trevizan, F. W., & Veloso, M. M. (2012). Short-sighted stochastic shortest path problems. In *International Conference on Automated Planning and Scheduling (ICAPS).*

Uwents, W., & Blockeel, H. (2005). Classifying relational data with neural networks. In *International Conference on Inductive Logic Programming (ILP)*.

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 Magazine*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*.

Xu, Y., Fern, A., & Yoon, S. W. (2007). Discriminative learning of beam-search heuristics for planning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*.

Yoon, S. W., Fern, A., & Givan, R. (2007). Using learned policies in heuristic-search planning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*.

Yoon, S., Fern, A., & Givan, R. (2006). Discrepancy search with reactive policies for planning. In *AAAI-06 Workshop on Learning for Search*.

Yoon, S., Fern, A., & Givan, R. (2002). Inductive policy selection for first-order MDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Younes, H. L., & Littman, M. L. (2004). PPDDL1.0: an extension to PDDL for expressing planning domains with probabilistic effects. Tech. rep., CMU.

Younes, H. L., Littman, M. L., Weissman, D., & Asmuth, J. (2005). The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research (JAIR)*.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., & Battaglia, P. (2019). Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations (ICLR)*.