Planning in Factored State and Action Spaces with Learned Binarized Neural Network Transition Models

Buser Say, Scott Sanner
Department of Mechanical & Industrial Engineering, University of Toronto, Canada
{bsay,ssanner}@mie.utoronto.ca

Abstract

In this paper, we leverage the efficiency of Binarized Neural Networks (BNNs) to learn complex state transition models of planning domains with discretized factored state and action spaces. In order to directly exploit this transition structure for planning, we present two novel compilations of the learned factored planning problem with BNNs based on reductions to Boolean Satisfiability (FD-SAT-Plan) as well as Binary Linear Programming (FD-BLP-Plan). Experimentally, we show the effectiveness of learning complex transition models with BNNs, and test the runtime efficiency of both encodings on the learned factored planning problem. After this initial investigation, we present an incremental constraint generation algorithm based on generalized landmark constraints to improve the planning accuracy of our encodings. Finally, we show how to extend the best performing encoding (FD-BLP-Plan+) beyond goals to handle factored planning problems with rewards.

1 Introduction

Deep neural networks have significantly improved the ability of autonomous systems to perform complex tasks, such as image recognition [Krizhevsky *et al.*, 2012], speech recognition [Deng *et al.*, 2013] and natural language processing [Collobert *et al.*, 2011], and can outperform humans and human-designed super-human systems in complex planning tasks such as Go [Silver *et al.*, 2016] and Chess [Silver *et al.*, 2017].

In the area of learning and online planning, recent work on HD-MILP-Plan [Say *et al.*, 2017] has explored a two-stage framework that (i) learns transitions models from data with ReLU-based deep networks and (ii) plans optimally with respect to the learned transition models using mixed-integer linear programming, but did not provide encodings that are able to learn and plan with *discrete* state variables. As an alternative to ReLU-based deep networks, Binarized Neural Networks (BNNs) [Hubara *et al.*, 2016] have been introduced with the specific ability to learn compact models over discrete variables, providing a new formalism for transition learning and planning in factored [Boutilier *et al.*, 1999] discrete state and action spaces that we explore in this paper. However planning with these BNN transition models poses two non-trivial questions: (i) What is the most efficient compilation of BNNs for planning in domains with factored state and (concurrent) action spaces? (ii) Given that BNNs may learn incorrect domain models, how can a planner repair BNN compilations to improve their planning accuracy?

To answer question (i), we present two novel compilations of the learned factored planning problem with BNNs based on reductions to Boolean Satisfiability (FD-SAT-Plan) and Binary Linear Programming (FD-BLP-Plan). Over three factored planning domains with multiple size and horizon settings, we test the effectiveness of learning complex state transition models with BNNs, and test the runtime efficiency of both encodings on the learned factored planning problems. While there are methods for learning PDDL models from data [Yang *et al.*, 2007; Amir and Chang, 2008] and excellent PDDL planners [Helmert, 2006; Richter and Westphal, 2010], we remark that BNNs are strictly more expressive than PDDL-based learning paradigms for learning concurrent effects in factored action spaces that may depend on the joint execution of one or more actions. Furthermore, while Monte Carlo Tree Search (MCTS) methods [Kocsis and Szepesvári, 2006; Keller and Helmert, 2013] including AlphaGo [Silver *et al.*, 2016] and AlphaGoZero [Silver *et al.*, 2016] *could* technically plan with a BNN-learned black box model of transition dynamics, unlike this work, they would not be able to exploit the BNN transition structure and they would not be able to provide optimality guarantees with respect to the learned model.

To answer question (ii), we introduce an incremental algorithm based on generalized landmark constraints from the decomposition-based cost-optimal classical planner [Davies *et al.*, 2015], where during online planning we detect and constrain invalid sets of action selections from the decision space of the planners and efficiently improve their planning accuracy. Finally, building on the above answers to (i) and (ii), we extend the best performing encoding to handle factored planning problems with general rewards (FD-BLP-Plan+).

In summary, this work provides the first planner capable of learning complex transition models in domains with mixed (continuous and discrete) factored state and action spaces as BNNs and capable of exploiting their structure in satisfiability (or optimization) encodings for planning purposes. Empirical results demonstrate strong performance in both goal-oriented and reward-oriented planning in both the learned and original domains, and provide a new transition learning and planning formalism to the data-driven model-based planning community.

2 Preliminaries

Before we present the SAT and BLP compilation of the learned planning problem, we review the preliminaries motivating this work.

2.1 Problem Definition

A deterministic factored planning problem is a tuple $\Pi = \langle S, A, C, T, I, G \rangle$ where $S = \{S^d, S^c\}$ is a mixed set of state variables with discrete S^d and continuous S^c domains, $A = \{A^d, A^c\}$ is a mixed set of action variables with discrete A^d and continuous A^c domains, $C : S \times A \rightarrow \{true, false\}$ is a function that returns true if action $\boldsymbol{a} \in A$ and state $\boldsymbol{s} \in S$ pair satisfies constraints that represent global constraints on state and action variables, $T : S \times A \rightarrow S$ denotes the stationary transition function between time steps t and $t+1$, $T(\boldsymbol{s}^t, \boldsymbol{a}^t) = \boldsymbol{s}^{t+1}$ if $c(\boldsymbol{s}^t, \boldsymbol{a}^t) = true$ for all global constraints $c \in C$, and is undefined otherwise. Finally, $I \subset C$ is the initial state and $G \subset C$ is the goal state constraints over the set and subset of state variables S, respectively. Given a planning horizon H, a solution (i.e. plan) to Π is a value assignment to action $\boldsymbol{a}^t$ and state $\boldsymbol{s}^t$ variables such that $T(\boldsymbol{s}^t, \boldsymbol{a}^t) = \boldsymbol{s}^{t+1}$ and $c(\boldsymbol{s}^t, \boldsymbol{a}^t) = true$ for all global constraints $c \in C$ and time steps $t \in \{1, \ldots, H\}$. The RDDL [Sanner, 2010] formalism is extended to handle goal-specifications and used to describe the problem Π.

2.2 Factored Planning with Deep-Net Learned Transition Models

Factored planning with deep-net learned transition models is a two-stage framework for learning and solving nonlinear factored planning problems as first introduced in HD-MILP-Plan [Say *et al.*, 2017]. Given samples of state transition data, the first stage of the framework learns the transition function $\tilde{T}$ using a deep-neural network with Rectified Linear Units (ReLUs) [Nair and Hinton, 2010] and linear activation units. The data is sampled from the RDDL-based domain simulator RDDLsim [Sanner, 2010]. In the second stage, the learned transition function $\tilde{T}$ is used to construct the factored planning problem $\tilde{\Pi} = \langle S, A, C, \tilde{T}, I, G \rangle$. That is, the trained deep-neural network with fixed weights is used to predict the state $\boldsymbol{s}^{t+1}$ at time $t+1$ for free state $\boldsymbol{s}^t$ and action $\boldsymbol{a}^t$ variables at time t such that $\tilde{T}(\boldsymbol{s}^t, \boldsymbol{a}^t) = \boldsymbol{s}^{t+1}$. As visualized in Figure 1, the learned transition function $\tilde{T}$ is sequentially chained over horizon $t \in \{1, \ldots, H\}$, and compiled into a Mixed-Integer Linear Program yielding the planner HD-MILP-Plan. Since HD-MILP-Plan utilizes only ReLUs and linear activation units in its learned transition models, the state variables $s \in S^c$ are restricted to have only continuous domains $S^c \subseteq S$.

2.3 Binarized Neural Networks

Binarized Neural Networks (BNNs) are neural networks with binary weights and activation functions [Hubara *et al.*, 2016].

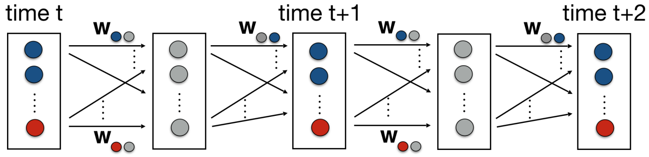

Figure 1: Visualization of HD-MILP-Plan, where blue circles represent state variables $s \in S$, red circles represent action variables $a \in A$, gray circles represent hidden units (i.e., ReLUs and linear activation units) and **w** represent the weights of a deep-neural network. During the learning stage, the weights **w** are learned from data. In the planning stage, the weights are fixed and HD-MILP-Plan optimizes a given reward function with respect to the free action $a \in A$ and state variables $s \in S$.

During inference, BNNs reduce memory requirements of a system by replacing most arithmetic operations with bit-wise operations. BNN layers are stacked in the following order:

Real or Binary Input Layer: Binary units in all layers, with the exception of the first layer, receive binary input. When the input of the first layer has real-valued domains $x \in \mathbb{R}$, m bits of precision can be used for a practical representation such that $\tilde{x} = \sum_{i=1}^{m} 2^{i-1} x^i w^i$ [Hubara *et al.*, 2016].

Binarization Layer: Given input $x_{j,l}$ of binary unit $j \in J(l)$ at layer $l \in \{1, \ldots, L\}$ the deterministic activation function used to compute output $y_{j,l}$ is: $y_{j,l} = 1$ if $x_{j,l} \geq 0$, -1 otherwise, where L denotes the number of layers and $J(l)$ denotes the set of binary units in layer $l \in \{1, \ldots, L\}$.

Batch Normalization Layer: For all layers $l \in \{1, \ldots, L\}$, Batch Normalization [Ioffe and Szegedy, 2015] is a method for transforming the weighted sum of outputs at layer $l-1$ in $\Delta_{j,l} = \sum_{i \in J(l-1)} w_{i,j,l-1} y_{i,l-1}$ to input $x_{j,l}$ of binary unit $j \in J(l)$ at layer l such that: $x_{j,l} = \frac{\Delta_{j,l} - \mu_{j,l}}{\sqrt{\sigma^2_{j,l} + \epsilon_{j,l}}} \gamma_{j,l} + \beta_{j,l}$ where parameters $w_{i,j,l-1}$, $\mu_{j,l}$, $\sigma^2_{j,l}$, $\epsilon_{j,l}$, $\gamma_{j,l}$ and $\beta_{j,l}$ denote the weight, input mean, input variance, numerical stability constant (i.e., epsilon), input scaling and input bias respectively, where all parameters are computed at training time.

2.4 Boolean Satisfiability Problem

The Boolean Satisfiability Problem (SAT) is the problem of determining whether there exists a value assignment to the variables of a Boolean formula such that the formula evaluates to true (i.e., satisfiable) [Davis and Putnam, 1960]. While the theoretical worst-case complexity of SAT is *NP-Complete*, state-of-art SAT solvers are shown to scale experimentally well for large instances with millions of variables and constraints [Biere *et al.*, 2009].

Boolean Cardinality Constraints

Boolean cardinality constraints describe bounds on the number of Boolean variables that are allowed to be true, and are in the form of $AtMost_k(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \leq k$. An efficient encoding of $AtMost_k(x_1, \ldots, x_n)$ in conjunctive normal form (CNF) [Sinz, 2005] is presented below:

$$\Big(\bigwedge_{1<j\leq k}(\neg s_{1,j})\Big)\wedge\Big(\bigwedge_{1\leq i<n}(\neg x_i\vee s_{i,1})\Big)$$
$$\wedge\Big(\bigwedge_{1<i<n}\bigwedge_{1\leq j\leq k}(\neg s_{i-1,j}\vee s_{i,j})\Big)\wedge\Big(\bigwedge_{1<i\leq n}(\neg x_i\vee\neg s_{i-1,k})\Big)$$
$$\wedge\Big(\bigwedge_{1<i<n}\bigwedge_{1<j\leq k}(\neg x_i\vee\neg s_{i-1,j-1}\vee s_{i,j})\Big)$$

Here, $s_{i,j}$ are auxiliary Boolean variables implicitly defined in the constraints above. Note that the cardinality constraint $\sum_{i=1}^{n} x_i \leq k$ is equivalent to $\sum_{i=1}^{n}\neg x_i \geq n-k$. For notational clarity, we use $AtLeast_k(x_1,\dots,x_n)$ to denote the cardinality constraint $\sum_{i=1}^{n} x_i \geq k$, which is equivalent to $AtMost_{n-k}(\neg x_1,\dots,\neg x_n)$.

2.5 Binary Linear Programming Problem

The Binary Linear Programming (BLP) problem requires finding the optimal value assignment to the variables of a mathematical model with linear constraints, linear objective function, and binary decision variables. Similar to SAT, the theoretical worst-case complexity of BLP is *NP-Complete*. The state-of-the-art BLP solvers [IBM, 2017] utilize branch-and-bound algorithms and can handle cardinality constraints efficiently in the size of its encoding.

2.6 Generalized Landmark Constraints

A generalized landmark constraint is a linear inequality in the form of $\sum_{a\in L}(x_a \geq k_a) \geq 1$ where $L \subset A$ denotes the set of action landmarks and k_a denotes counts on actions $a \in L$, that is, the minimum number of times an action must occur in a plan [Davies *et al.*, 2015]. Davies et al. introduced a decomposition-based planner, *OpSeq*, that incrementally updates generalized landmark constraints to find cost-optimal plans to classical planning problems.

3 SAT Compilation of the Learned Factored Planning Problem

In this section, we show how to reduce the learned factored planning problem $\tilde{\Pi}$ with BNNs into SAT which we denote as Factored Deep SAT Planner (FD-SAT-Plan).

3.1 Propositional Variables

First, we describe the set of propositional variables used in FD-SAT-Plan. We use three sets of propositional variables: action variables, state variables and BNN binary units, where variables use a bitwise encoding.

- $X^i_{a,t}$ denotes if i-th bit of action $a \in A$ is executed at time $t \in \{1,\dots,H\}$.
- $Y^i_{s,t}$ denotes if i-th bit of state $s \in S$ is true at time $t \in \{1,\dots,H+1\}$.
- $Z_{j,l,t}$ denotes if binary unit $j \in J(l)$ at layer $l \in \{1,\dots,L\}$ is activated at time $t \in \{1,\dots,H\}$.

3.2 Parameters

Next we define the additional parameters used in FD-SAT-Plan.

- I^i_s is the initial (i.e., at $t=1$) value of the i-th bit of state variable $s \in S$.
- S^G is the set of state variables $S^G \subseteq S$ specified by the goal constraints G.
- G^i_s is the goal (i.e., at $t = H+1$) value of the i-th bit of state variable $s \in S^G$.
- $In(x,i)$ is the function that maps the i-th bit of a state or an action variable $x \in S \cup A$ to the corresponding binary unit in the input layer of the BNN such that $In(x,i) = j$ where $j \in J(1)$.
- $Out(s,i)$ is the function that maps the i-th bit of a state variable $s \in S$ to the corresponding binary unit in the output layer of the BNN such that $Out(s,i) = j$ where $j \in J(L)$.

3.3 The SAT Compilation

Below, we define the SAT encoding of the learned factored planning problem $\tilde{\Pi}$ with BNNs.

Initial and Goal State Constraints

$$\bigwedge_{1\leq i\leq m}\Big(\big(\bigwedge_{s\in S}(Y^i_{s,1}\leftrightarrow I^i_s)\big)\wedge\big(\bigwedge_{s\in S^G}(Y^i_{s,H+1}\leftrightarrow G^i_s)\big)\Big) \quad (1)$$

where clause (1) set the initial and goal values of the state variables at times $t = 1$ and $t = H+1$, respectively.

BNN Constraints

$$\bigwedge_{1\leq t\leq H}\ \bigwedge_{s\in S}\ \bigwedge_{1\leq i\leq m}(Y^i_{s,t}\leftrightarrow Z_{In(s,i),1,t}) \quad (2)$$

$$\bigwedge_{1\leq t\leq H}\ \bigwedge_{a\in A}\ \bigwedge_{1\leq i\leq m}(X^i_{a,t}\leftrightarrow Z_{In(a,i),1,t}) \quad (3)$$

$$\bigwedge_{1\leq t\leq H}\ \bigwedge_{s\in S}\ \bigwedge_{1\leq i\leq m}(Y^i_{s,t+1}\leftrightarrow Z_{Out(s,i),L,t}) \quad (4)$$

$$\bigwedge_{1\leq t\leq H}\ \bigwedge_{2\leq l\leq L}\Big(Z_{j,l,t}\rightarrow AtLeast_k\big(Z_{i,l-1,t}\ w_{i,j,l-1}=1, i\in J(l-1), \neg Z_{i,l-1,t}\ w_{i,j,l-1}=-1, i\in J(l-1)\big)\Big) \quad (5)$$

$$\bigwedge_{1\leq t\leq H}\ \bigwedge_{2\leq l\leq L}\Big(\neg Z_{j,l,t}\rightarrow AtLeast_{k'}\big(\neg Z_{i,l-1,t}\ w_{i,j,l-1}=1, i\in J(l-1), Z_{i,l-1,t}\ w_{i,j,l-1}=-1, i\in J(l-1)\big)\Big) \quad (6)$$

where k in the cardinality constraints is the binary activation threshold computed at training time such that:

$$k=\left\lceil\frac{|J(l-1)|+\mu_{j,l}-\frac{\beta_{j,l}\sqrt{\sigma^2_{j,l}+\epsilon_{j,l}}}{\gamma_{j,l}}}{2}\right\rceil$$

for binary unit $j \in J(l)$ in layer $l \in \{2,\dots,L\}$, where $|x|$ denotes the size of set x, and $k' = |J(l-1)| - k + 1$ Clauses

(2-3) map the binary units at the input layer of the BNN (i.e., $l = 1$) to a unique state or action variable, respectively. Similarly, clause (4) maps the binary units at the output layer of the BNN (i.e., $l = L$) to a unique state variable. Clauses (5-6) encode the binary activation of every unit in the BNN.

Global Constraints

$$\bigwedge_{c \in C} (c(Y^i_{s,t}\ 1 \leq t \leq H+1, s \in S, 1 \leq i \leq m, \\ X^i_{a,t}\ 1 \leq t \leq H, a \in A, 1 \leq i \leq m)) \quad (7)$$

where clause (7) represents domain-dependent global constraints on state and action variables. Some common examples of global constraints $c \in C$ such as mutual exclusion on Boolean action variables and one-hot encodings for the output of the BNN (i.e., exactly one Boolean state variable must be true) are respectively encoded below by clauses (8-9):

$$AtMost_1(X_{a,t}\ a \in A) \quad (8)$$

$$AtMost_1(Y_{s,t}\ s \in S) \wedge (\bigvee_{s \in S} Y_{s,t}) \quad (9)$$

In general, linear global constraints in the form of $\sum_i a_i x_i \leq k$, such as bounds on state and action variables, can be encoded in SAT where a_i are positive integer coefficients and x_i are decision variables with non-negative integer domains [Abío and Stuckey, 2014].

4 BLP Compilation of the Learned Factored Planning Problem

Given FD-SAT-Plan, we present the BLP compilation of the learned factored planning problem $\tilde{\Pi}$ with BNNs, which we denote as Factored Deep BLP Planner (FD-BLP-Plan).

4.1 Binary Variables and Parameters

FD-BLP-Plan uses the same set of decision variables and parameters as FD-SAT-Plan.

4.2 The BLP Compilation

FD-BLP-Plan replaces clauses (1-9) with equivalent linear constraints. Clauses (1-4) are replaced by the following equality constraints:

$$Y^i_{s,1} = I^i_s \qquad \forall_{s \in S, 1 \leq i \leq m} \quad (10)$$

$$Y^i_{s,H+1} = G^i_s \qquad \forall_{s \in S^G, 1 \leq i \leq m} \quad (11)$$

$$Y^i_{s,t} = Z_{In(s,i),1,t} \qquad \forall_{1 \leq t \leq H, s \in S, 1 \leq i \leq m} \quad (12)$$

$$X^i_{a,t} = Z_{In(a,i),1,t} \qquad \forall_{1 \leq t \leq H, a \in A, 1 \leq i \leq m} \quad (13)$$

$$Y^i_{s,t+1} = Z_{Out(s,i),L,t} \qquad \forall_{1 \leq t \leq H, s \in S, 1 \leq i \leq m} \quad (14)$$

Clause (5) is replaced by the following linear constraint:

$$kZ_{j,l,t} \leq \sum_{\substack{w_{i,j,l-1}=1 \\ i \in J(l-1)}} Z_{i,l-1,t} + \sum_{\substack{w_{i,j,l-1}=-1 \\ i \in J(l-1)}} (1 - Z_{i,l-1,t}) \quad (15)$$

for all $1 \leq t \leq H, 2 \leq l \leq L$. Similarly, clause (6) is replaced by the following linear constraint:

$$k'(1 - Z_{j,l,t}) \leq \sum_{\substack{w_{i,j,l-1}=-1 \\ i \in J(l-1)}} Z_{i,l-1,t} + \sum_{\substack{w_{i,j,l-1}=1 \\ i \in J(l-1)}} (1 - Z_{i,l-1,t}) \quad (16)$$

for all $1 \leq t \leq H, 2 \leq l \leq L$. Finally, clauses (7-9) are replaced by linear constraints in the form of $\sum_i a_i x_i \leq k$.

5 Incremental Factored Planning Algorithm for FD-SAT-Plan and FD-BLP-Plan

Now we introduce an incremental algorithm for excluding invalid plans from the search space of FD-SAT-Plan and FD-BLP-Plan. Similar to *OpSeq* [Davies *et al.*, 2015], we update our planners with generalized landmark constraints

$$\sum_{a \in A} (\sum_{\substack{1 \leq t \leq H \\ (a,t) \in L_n}} (1 - X_{a,t}) + \sum_{\substack{1 \leq t \leq H \\ (a,t) \notin L_n}} X_{a,t}) \geq 1 \quad (17)$$

where L_n is the set of actions $a \in A$ executed at time $1 \leq t \leq H$ at the n-th iteration of the algorithm outlined below.

Algorithm 1 Incremental Factored Planning Algorithm for FD-SAT-Plan and FD-BLP-Plan

```
1: $n = 1$, planner = FD-SAT-Plan or FD-BLP-Plan
2: $L_n \leftarrow$ Solve planner.
3: if $L_n$ is $\emptyset$ or $L_n$ is a plan for $\Pi$ then
4:     return $L_n$
5: else planner $\leftarrow$ Constraint (17)
6: $n \leftarrow n + 1$, go to line 2.
```

For a given horizon H, Algorithm 1 iteratively computes a set of actions L_n, or returns infeasibility for the learned factored planning problem $\tilde{\Pi}$. If the set of actions L_n is non-empty, we evaluate whether L_n is a valid plan for the original factored planning problem Π (i.e., line 3) either in the actual domain or using a high fidelity domain simulator – in our case RDDLsim. If the set of actions L_n constitutes a plan for Π, Algorithm 1 returns L_n as a plan. Otherwise, the planner is updated with the new set of generalized landmark constraints to exclude L_n and the loop repeats. Since the original action space is discretized and represented upto m bits of precision, Algorithm 1 can be shown to terminate in no more than $n = 2^{|A| \times m \times H}$ iterations by constructing an inductive proof similar to the termination criteria of *OpSeq* where either a feasible plan for Π is returned or there does not exist a plan to both $\tilde{\Pi}$ and Π for the given horizon H.

6 Experimental Results

In this section, we evaluate the effectiveness of factored planning with BNNs. First, we present the benchmark domains used to test the efficiency of our learning and factored planning framework with BNNs. Second, we present the accuracy of BNNs to learn complex state transition models for factored planning problems. Third, we test the efficiency and scalability of planning with FD-SAT-Plan and FD-BLP-Plan on the learned factored planning problems $\tilde{\Pi}$ across multiple problem sizes and horizon settings. Fourth, we demonstrate the effectiveness of Algorithm 1 to find a plan for the factored planning problem Π. Finally we test the effectiveness of factored planning with the best performing encoding over the benchmark domains with reward specifications.

6.1 Domain Descriptions

The deterministic RDDL domains used in the experiments, namely Navigation [Sanner and Yoon, 2011], Inventory Control (Inventory) [Mann and Mannor, 2014], and System Administrator (SysAdmin) [Guestrin *et al.*, 2001; Sanner and Yoon, 2011] are described below.

Navigation models an agent in a two-dimensional (m-by-n) maze with obstacles where the goal of the agent is to move from the initial location to the goal location at the end of horizon H. The transition function T describes the movement of the agent as a function of the topological relation of its current location to the maze, the moving direction and whether the location the agent tries to move to is an obstacle or not. This domain is a deterministic version of its original from IPPC2011 [Sanner and Yoon, 2011]. Both the action and the state space is Boolean. We report the results on instances with two maze sizes m-by-n and three horizon settings H where $m = 3, 4$, $n = 3$, $H = 4, 6, 8$.

Inventory describes the inventory management control problem with alternating demands for a product over time where the management can order a fixed amount of units to increase the number of units in stock at any given time. The transition function T updates the state based on the change in stock as a function of demand, the current order quantity, and whether an order has been made or not. The action space is Boolean (either order a fixed positive integer amount, or do not order) and the state space is non-negative integer. We report the results on instances with two demand cycle lengths d and three horizon settings H where $d = 2, 4$ and $H = 4, 6, 8$.

SysAdmin models the behavior of a computer network P where the administrator can reboot a limited number of computers to keep the number of computers running above a specified safety threshold over time. The transition function T describes the status of a computer which depends on its topological relation to other computers, its age and whether it has been rebooted or not, and the age of the computer which depends on its current age and whether it has been rebooted or not. This domain is a deterministic modified version of its original from IPPC2011 [Sanner and Yoon, 2011]. The action space is Boolean and the state space is a non-negative integer where concurrency between actions are allowed. We report the results on instances with two network sizes $|P|$ and three horizon settings H where $|P| = 3, 4$ and $H = 2, 3, 4$.

6.2 Transition Learning Performance

In Table 1, we present test errors for different configurations of the BNNs on each domain instance where the sample data was generated using a simple stochastic exploration policy. For each instance of a domain, state transition pairs were collected and the data was treated as independent and identically distributed. After random permutation, the data was split into training and test sets with 9:1 ratio. The BNNs were trained on MacBookPro with 2.8 GHz Intel Core i7 16 GB memory using the code available [Hubara *et al.*, 2016]. Overall, Navigation instances required the smallest BNN structures for learning due to their purely Boolean state and action spaces, while both Inventory and SysAdmin instances required larger BNN structures for accurate learning, owing to their non-Boolean state and action spaces.

Table 1: Transition Learning Performance Table measured by error on test data (in %) for all domains and instances.

Domain	Network Structure	Test Error (%)
Navigation(3,3)	12:64:64:9	2.12
Navigation(4,3)	16:80:80:12	6.59
Inventory(2)	7:96:96:5	5.51
Inventory(4)	8:128:128:5	4.58
SysAdmin(3)	12:128:128:9	3.73
SysAdmin(4)	16:96:96:96:12	8.99

6.3 Planning Performance on the Learned Factored Planning Problems

We compare the effectiveness of using a SAT-based encoding against a BLP-based encoding, namely FD-SAT-Plan and FD-BLP-Plan to find plans for the learned factored planning problem $\tilde{\Pi}$. We ran the experiments on MacBookPro with 2.8 GHz Intel Core i7 16GB memory. For FD-SAT-Plan and FD-BLP-Plan, we used Glucose [Audemard and Simon, 2014] and CPLEX 12.7.1 [IBM, 2017] solvers respectively, with 30 minutes total time limit per domain instance.

Overall Performance Analysis

In Figure 2, we present the runtime performance of FD-SAT-Plan (Red Bar) and FD-BLP-Plan (Blue Bar) over Navigation (Figure 2a), Inventory (Figure 2b) and SysAdmin (Figure 2c) domains. The analysis of the runtime performances across all three domains show that the increase in the size of the underlying BNN structure (as presented in Table 1) significantly increases the computational effort required to find a plan. Similarly for both planners, we observed that increasing the size of the horizon, with the exception of instance (Nav,3,8), increases the cost of computing a plan.

Comparative Performance per Encoding

The pairwise comparison of FD-SAT-Plan and FD-BLP-Plan over all problem settings show a clear dominance of FD-BLP-Plan over FD-SAT-Plan in terms of runtime performance. Overall, FD-SAT-Plan computed plans for 15 out of 18 instances while FD-BLP-Plan successfully found plans for all instances. Moreover, with the exception of instances (Nav,4,6) and (Nav,4,8), FD-BLP-Plan found plans for the learned factored planning problems under 20 seconds. On average, FD-BLP-Plan is two orders of magnitude faster than FD-SAT-Plan.

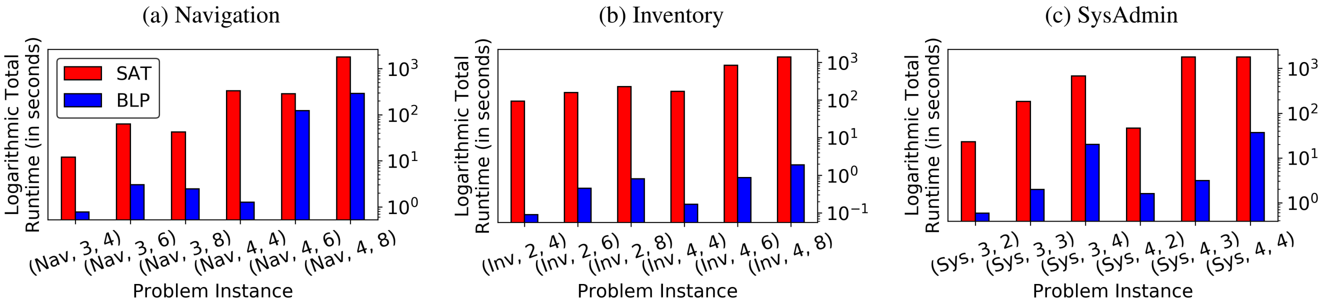

Figure 2: Timing comparison between FD-SAT-Plan (Red Bar) and FD-BLP-Plan (Blue Bar). Over all problem settings, the BLP encoding of the learned factored planning problem consistently outperformed its SAT equivalent.

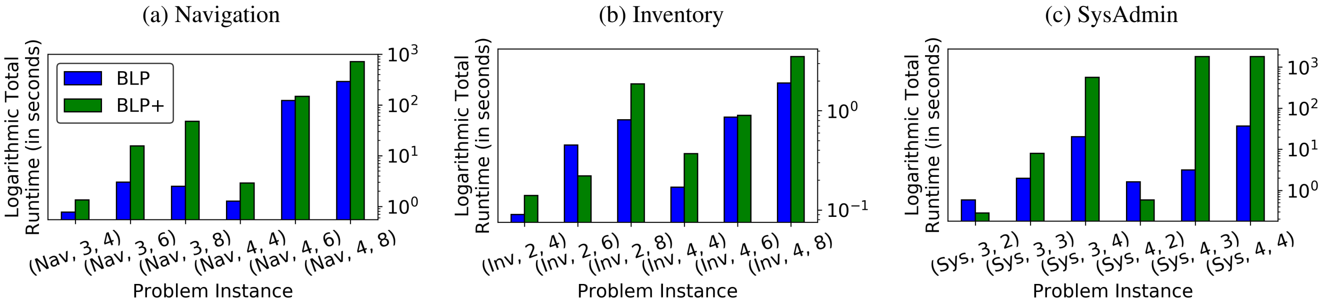

Figure 3: Timing comparison between FD-BLP-Plan (Blue Bar) and FD-BLP-Plan+ (Green Bar). The additional computational resources required to solve the factored planning problem with reward specifications varied across different domains where the computational effort for finding a plan increased minimally, moderately and significantly in Inventory, Navigation and SysAdmin domains, respectively.

6.4 Planning Performance on the Factored Planning Problems

We test the planning efficiency of the incremental factored planning algorithm using the best performing planner, that is FD-BLP-Plan, for solving the factored planning problem Π. Overall only three instances, namely (Inv,4,8), (Sys,4,3) and (Sys,4,4), required constraint generation to find a plan where the maximum number of constraints required was equal to one. The instances that required the generation of landmark constraints, the runtime performance of FD-BLP-Plan was almost identical to the results presented in Figure 2.

6.5 Planning Performance on the Factored Planning Problems with Reward Specifications

Finally, we extend FD-BLP-Plan to handle domains with reward specifications. Hereby, we extend the definition of the factored planning problem Π to include the reward function $Q : S \times A \rightarrow \mathbb{R}$ such that $\Pi^{+} = \langle S, A, C, T, I, G, Q \rangle$. Given a planning horizon H, an optimal solution to Π^{+} is a plan that maximizes the total reward function over all time steps such that $\sum_{t=1}^{H} Q(\boldsymbol{s}^{t+1}, \boldsymbol{a}^{t})$. Similar to HD-MILP-Plan [Say *et al.*, 2017], we assume the knowledge on the reward function Q and add Q as an objective function to our BLP encoding, which we denote as FD-BLP-Plan+.

In Figure 3, we show results for solving the factored planning problems with reward specifications using Algorithm 1. The pairwise comparison of FD-BLP-Plan and FD-BLP-Plan+ over all domain instances shows that the additional computational effort required to solve Π^{+} is minimal, moderate and significant in Inventory, Navigation and SysAdmin domains, respectively. Especially in (Sys,4,3) and (Sys,4,4) instances, FD-BLP-Plan ran out of time at its fifth iteration of Algorithm 1, as the solver could not prove the optimality of the incumbent solution found.

7 Conclusion

In this work, we utilized the efficiency and ability of BNNs to learn complex state transition models of factored planning domains with discretized state and action spaces. We introduced two novel compilations, a SAT (FD-SAT-Plan) and a BLP (FD-BLP-Plan) encoding, that directly exploit the structure of BNNs to plan for the learned factored planning problem, which provide optimality guarantees with respect to the learned model if they successfully terminate. We further introduced an incremental factored planning algorithm based on generalized landmark constraints that improve planning accuracy of both encodings. Finally, we extended the best performing encoding to handle factored planning problems with reward specifications (FD-BLP-Plan+). Empirical results showed we can accurately learn complex state transition models using BNNs and demonstrated strong performance in goal-oriented and reward-oriented planning in both the learned and original domains. In sum, this work provides a novel and effective factored state and action transition learning and planning formalism to the data-driven model-based planning community.

References

[Abío and Stuckey, 2014] Ignasi Abío and Peter J. Stuckey. Encoding linear constraints into sat. In *Principles and Practice of Constraint Programming*, pages 75–91. Springer Int Publishing, 2014.

[Amir and Chang, 2008] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *JAIR*, 33:349–402, 2008.

[Audemard and Simon, 2014] Gilles Audemard and Laurent Simon. *Lazy Clause Exchange Policy for Parallel SAT Solvers*, pages 197–205. Springer Int. Publishing, 2014.

[Biere *et al.*, 2009] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR*, 11(1):1–94, 1999.

[Collobert *et al.*, 2011] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011.

[Davies *et al.*, 2015] Toby O. Davies, Adrian R. Pearce, Peter J. Stuckey, and Nir Lipovetzky. Sequencing operator counts. In *25th ICAPS*, pages 61–69, 2015.

[Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Deng *et al.*, 2013] Li Deng, Geoffrey E. Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: an overview. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603, 2013.

[Guestrin *et al.*, 2001] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *17th IJCAI*, pages 673–680, 2001.

[Helmert, 2006] Malte Helmert. The fast downward planning system. *JAIR*, 26(1):191–246, 2006.

[Hubara *et al.*, 2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *29th NIPS*, pages 4107–4115. Curran Associates, Inc., 2016.

[IBM, 2017] IBM. *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual*, 2017.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *32nd ICML*, pages 448–456, 2015.

[Keller and Helmert, 2013] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *23rd ICAPS*, pages 135–143, 2013.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, pages 282–293, 2006.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *25th NIPS*, pages 1097–1105, 2012.

[Mann and Mannor, 2014] Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *21st ICML*, volume 1, 2014.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *27th ICML*, pages 807–814, 2010.

[Richter and Westphal, 2010] Silvia Richter and Matthias Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *JAIR*, 39(1):127–177, 2010.

[Sanner and Yoon, 2011] Scott Sanner and Sungwook Yoon. International probabilistic planning competition. 2011.

[Sanner, 2010] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. 2010.

[Say *et al.*, 2017] Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *26th IJCAI*, pages 750–756, 2017.

[Silver *et al.*, 2016] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, pages 484–503, 2016.

[Silver *et al.*, 2017] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 2017.

[Sinz, 2005] Carsten Sinz. *Towards an Optimal CNF Encoding of Boolean Cardinality Constraints*, pages 827–831. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *AIJ*, 171(2):107–143, 2007.