Compact and Efficient Encodings for Planning in Factored State and Action Spaces with Learned Binarized Neural Network Transition Models[☆]

Buser Say, Scott Sanner
{bsay,ssanner}@mie.utoronto.ca

Department of Mechanical & Industrial Engineering, University of Toronto, Canada
Vector Institute, Canada

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 Weighted Partial Maximum Boolean Satisfiability (FD-SAT-Plan+) as well as Binary Linear Programming (FD-BLP-Plan+). Theoretically, we show that our SAT-based Bi-Directional Neuron Activation Encoding is asymptotically the most compact encoding in the literature and maintains the generalized arc-consistency property through unit propagation – an important property that facilitates efficiency in SAT solvers. Experimentally, we validate the computational efficiency of our Bi-Directional Neuron Activation Encoding in comparison to an existing neuron activation encoding and demonstrate the effectiveness of learning complex transition models with BNNs. We test the runtime efficiency of both FD-SAT-Plan+ and FD-BLP-Plan+ on the learned factored planning problem showing that FD-SAT-Plan+ scales better with increasing BNN size and complexity. Finally, we present a finite-time incremental constraint generation algorithm based on generalized landmark constraints to improve the planning accuracy of

[☆]Parts of this work appeared in preliminary form in Say and Sanner, 2018 [1].

our encodings through simulated or real-world interaction.

Keywords: data-driven planning, binarized neural networks, Weighted Partial Maximum Boolean Satisfiability, Binary Linear Programming

1. Introduction

Deep neural networks (DNNs) have significantly improved the ability of autonomous systems to perform complex tasks, such as image recognition [2], speech recognition [3] and natural language processing [4], and can outperform humans and human-designed super-human systems in complex planning tasks such as Go [5] and Chess [6].

In the area of learning and planning, recent work on HD-MILP-Plan [7] has explored a two-stage framework that (i) learns transitions models from data with ReLU-based DNNs 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 DNNs, Binarized Neural Networks (BNNs) [8] 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 [9] discretized 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 (or prove the re-training of BNN is necessary)?

To answer question (i), we present two novel compilations of the learned factored planning problem with BNNs based on reductions to Weighted Partial Maximum Boolean Satisfiability (FD-SAT-Plan+) and Binary Linear Programming (FD-BLP-Plan+). Theoretically, we show that the SAT-based Bi-Directional Neuron Activation Encoding is asymptotically the most compact

encoding in the literature [10] and has the generalized arc-consistency property through unit propagation. Experimentally, we demonstrate the computational efficiency of our Bi-Directional Neuron Activation Encoding compared to the existing neuron activation encoding [1]. Then, we test the effectiveness of learning complex state transition models with BNNs, and test the runtime efficiency of both FD-SAT-Plan+ and FD-BLP-Plan+ on the learned factored planning problems over four factored planning domains with multiple size and horizon settings. While there are methods for learning PDDL models from data [11, 12] and excellent PDDL planners [13, 14], 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 [15, 16] including AlphaGo [5] and AlphaGoZero [5] *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 a finite-time incremental algorithm based on generalized landmark constraints from the decomposition-based cost-optimal classical planner [17], where we detect and constrain invalid sets of action selections from the decision space of the planners and efficiently improve their planning accuracy.

In summary, this work provides the first two planners 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 Weighted Partial Maximum Boolean Satisfiability and Binary Linear Programming encodings for planning purposes. Theoretically we show the compactness and efficiency of our SAT-based encoding, and the finiteness of our incremental algorithm. Empirical results show the computational efficiency of our new Bi-Directional Neuron Activation Encoding, demonstrate strong performance for FD-SAT-Plan+ and FD-BLP-Plan+ 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 Weighted Partial Maximum Boolean Satifiability (WP-MaxSAT) and Binary Linear Programming (BLP) compilations of the learned planning problem, we review the preliminaries motivating this work. We begin this section by describing the formal notation and the problem definition that is used in this work.

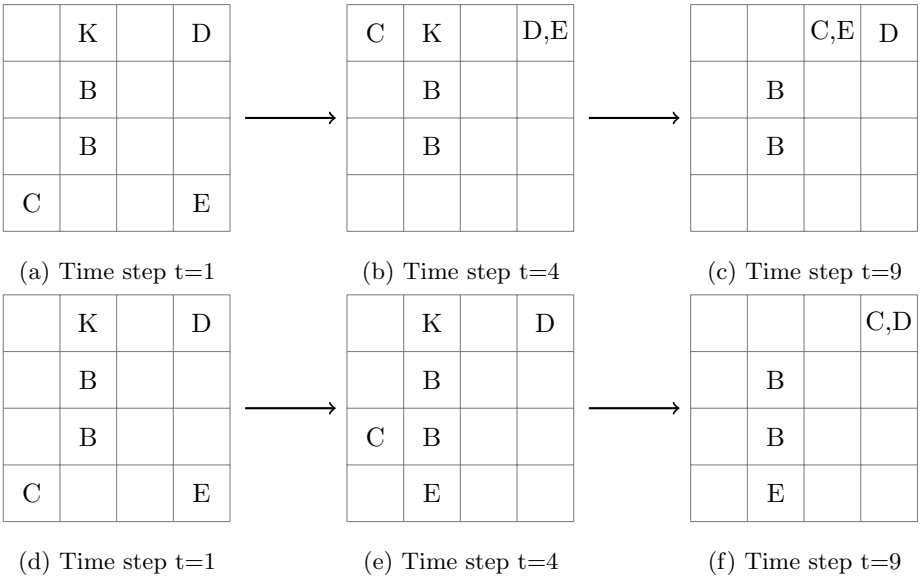
2.1. Problem Definition

A deterministic factored planning problem is a tuple $\Pi = \langle S, A, C, T, I, G, Q \rangle$ 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 A and state S variables satisfy constraints that represent global constraints, $T : S \times A \rightarrow S$ denotes the stationary transition function, and $Q : S \times A \rightarrow \mathbb{R}$ is the reward function. Finally, $I : S \rightarrow \{true, false\}$ is the initial state constraints that assign values to all state variables S, and $G : S_G \rightarrow \{true, false\}$ is the goal state constraints over the subset of state variables $S_G \subseteq S$.

Given a planning horizon H, a solution $\pi = \langle \bar{A}^1, \ldots, \bar{A}^H \rangle$ (i.e. plan) to Π is a value assignment to action $\bar{A}^t$ and state $\bar{S}^t$ variables such that $T(\bar{S}^t, \bar{A}^t) = \bar{S}^{t+1}$, $C(\bar{S}^t, \bar{A}^t) = true$ over global constraints C and time steps $t \in \{1, \ldots, H\}$ and initial and goal state constraints are satisfied such that $I(\bar{S}^1) = true$ and $G(\{\bar{s}^{H+1} | s \in S_G\}) = true$, respectively. Similarly, given a planning horizon H, an optimal solution to Π is a plan that maximizes the total reward function $\sum_{t=1}^{H} Q(S^{t+1}, A^t)$.

Next, we introduce an example domain with a complex transition structure.

2.2. Example Domain: Cellda



(a) Time step t=1 (b) Time step t=4 (c) Time step t=9

(d) Time step t=1 (e) Time step t=4 (f) Time step t=9

Figure 1: Vizualization of the Cellda domain in a 4-by-4 maze for two plans (top) $\pi_1 = \langle \bar{up}^1, \bar{up}^2, \bar{up}^3, \bar{right}^4, \bar{right}^5, \bar{right}^6 \rangle$ (1a-1c) and (bottom) $\pi_2 = \langle \bar{up}^3, \bar{up}^4, \bar{up}^5, \bar{right}^6, \bar{right}^7, \bar{right}^8 \rangle$ (1d-1f) where any unassigned action variable is assumed false. The meaning of (B)lock, (C)ellda, (D)oor, (E)nemy, and (K)ey are described in the text. A plan that ignores the adversarial policy of the enemy E (e.g., π_1) will get hit by the enemy, as opposed to a plan that takes into account the adversarial policy of the enemy E (e.g., π_2). With plan π_2, Cellda C avoids getting hit by waiting for two time steps to trap her enemy who will try to move up for the remaining of time steps.

Influenced by the famous video game The Legend of Zelda [18], Cellda domain models an agent in a two dimensional (4-by-4) dungeon cell. As visualized by Figure 1, the agent Cellda (C) must escape a dungeon through an initially locked door (D) by obtaining its key (K) without getting hit by her enemy (E). The gridworld-like dungeon is made up of two types of cells: i) regular cells (blank) on which Cellda and her enemy can move from/to deterministically up, down, right or left, and ii) blocks (B) that neither Cellda nor her enemy can walk-through. The state variables of this domain include two integer variables for describing the location of Cellda, two integer variables for describing the

location of the enemy, one boolean variable for describing whether the key is obtained or not, and one boolean variable for describing whether Cellda is alive or not. The action variables of this domain include four mutually exclusive boolean variables for describing the movement of Cellda (i.e., up, down, right or left). The enemy has an adversarial deterministic policy that is unknown to Cellda that will try to minimize the total Manhattan distance between itself and Cellda by breaking the symmetry first in vertical axis. The goal of this domain is to learn the unknown policy of the enemy from previous plays (i.e., data) and escape the dungeon without getting hit. The complete description of this domain can be found in Appendix C.

Given that the state transition function T that describes the location of the enemy must be learned, a planner that fails to learn the adversarial policy of the enemy E (e.g., π_1 as visualized in Figure 1(1a-1c)) will get hit by the enemy. In contrast, a planner that learns the adversarial policy of the enemy E (e.g., π_2 as visualized in Figure 1(1d-1f)) can avoid getting hit by the enemy in this scenario by waiting for two time steps to trap her enemy (who will try to move up for the remaining of time steps and fail to intercept Cellda).

To solve this problem, next we describe a learning and planning framework that i) learns an unknown transition function T from data, and ii) plans optimally with respect to the learned deterministic factored planning problem.

2.3. Factored Planning with Deep Neural Network Learned Transition Models

Factored planning with DNN learned transition models is a two-stage framework for learning and solving nonlinear factored planning problems as first introduced in HD-MILP-Plan [7] that we briefly review now. Given samples of state transition data, the first stage of the HD-MILP-Plan framework learns the transition function $\tilde{T}$ using a DNN with Rectified Linear Units (ReLUs) [19] and linear activation units. In the second stage, the learned transition function $\tilde{T}$ is used to construct the learned factored planning problem $\tilde{\Pi} = \langle S, A, C, \tilde{T}, I, G, Q \rangle$. That is, the trained DNN with fixed weights is used to predict the state S^{t+1} at time step $t+1$ for free state S^t and action A^t variables at time step t such

that $\tilde{T}(S^t, A^t) = S^{t+1}$. As visualized in Figure 2, 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 [7]. 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 $S^c \subseteq S$.

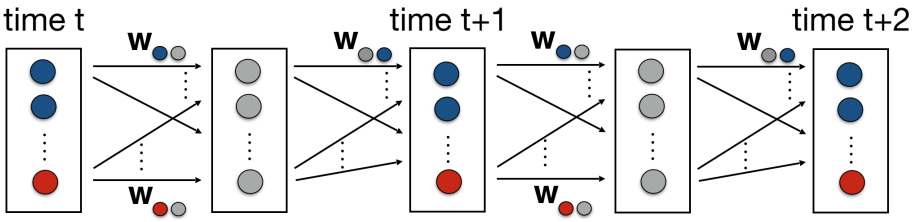


Figure 2: Visualization of the learning and planning framework [7], where blue circles represent state variables S, red circles represent action variables A, gray circles represent hidden units (i.e., ReLUs and linear activation units for HD-MILP-Plan [7], and binary hidden units for FD-SAT-Plan+ and FD-BLP-Plan+) and $\mathbf{w}$ represent the weights of a DNN. During the learning stage, the weights $\mathbf{w}$ are learned from data. In the planning stage, the weights are fixed and the planner optimizes a given reward function with respect to the free action A and state variables S.

Next, we describe an efficient DNN structure for learning discrete models, namely Binarized Neural Networks (BNNs) [8].

2.4. Binarized Neural Networks

Binarized Neural Networks (BNNs) are neural networks with binary weights and activation functions [8]. As a result, BNNs naturally learn discrete models by replacing most arithmetic operations with bit-wise operations. Before we describe how BNN learned transitions relate to HD-MILP-Plan in Figure 2, we first provide a technical description of the BNN architecture, where 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 $\tilde{x} = \sum_{i=1}^{m} 2^{i-1} x^i$ [8].

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 [20] 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.

In order to place BNN-learned transition models in the same planning and learning framework of HD-MILP-Plan [7], we simply note that once the above BNN layers are learned, the Batch Normalization layers reduce to simple linear transforms. This results in a BNN with layers as visualized in Figure 2, where (i) all weights $\mathbf{w}$ are restricted to either +1 or -1 and (ii) all nonlinear transfer functions at BNN units are restricted to thresholded counts of inputs. The benefit of the BNN encoding over the ReLU-based DNNs of HD-MILP-Plan [7] is that it can directly model discrete variable transitions and BNNs can be translated to both Binary Linear Programming (BLP) and Weighted Partial Maximum Boolean Satisfiability (WP-MaxSAT) problems discussed next.

2.5. Weighted Partial Maximum Boolean Satisfiability Problem

In this work, one of the planning encodings that we focus on is Weighted Partial Maximum Boolean Satisfiability (WP-MaxSAT). WP-MaxSAT is the problem of finding a value assignment to the variables of a Boolean formula that consists of **hard clauses** and weighted **soft clauses** such that i) *all hard clauses evaluate to true (i.e., standard SAT)* [21], and ii) *the total weight of the*

unsatisfied soft clauses is minimized. While WP-MaxSAT is known to be *NP-hard*, state-of-the-art WP-MaxSAT solvers are experimentally shown to scale well for large instances [22].

2.6. Boolean Cardinality Constraints

When compiling BNNs to satisfiability encodings, it is critical to encode the counting (cardinality) threshold of the binarization layer as compactly as possible. Boolean cardinality constraints describe bounds on the number of Boolean variables that are allowed to be true, and are in the form of $Most_p(\{x_1, \ldots, x_n\}) = \sum_{i=1}^{n} x_i \leq p$. Cardinality Networks $CN_k^{\leq}([x_1, \ldots, x_n] \rightarrow [c_1, \ldots, c_n])$ provide an efficient encoding in conjunctive normal form (CNF) for counting an upper bound on the number of true assignments to Boolean variables $\{x_1, \ldots, x_n\}$ using auxiliary Boolean counting variables c_i such that $\min(\sum_{j=1}^{n} x_j, i) \leq \sum_{j=1}^{i} c_j$ holds for all $i \in \{1, \ldots, k\}$ where $k = \lceil \log_2(p) \rceil$ [23]. The detailed CNF encoding of $CN_k^{\leq}$ is outlined in Appendix A. Given $CN_k^{\leq}$, Boolean cardinality constraint $Most_p(\{x_1, \ldots, x_n\})$ is defined as

$$Most_p(\{x_1, \ldots, x_n\}) = \bigwedge_{i=n+1}^{r} (\neg x_i) \wedge (\neg c_{p+1}) \wedge CN_k^{\leq}([x_1, \ldots, x_{n+r}] \rightarrow [c_1, \ldots, c_k]) \quad (1)$$

where $r = \lceil \frac{n}{k} \rceil k - n$ is the size of additional input variables. Boolean cardinality constraint $Most_p(\{x_1, \ldots, x_n\})$ is encoded using $O(nlog_2^2 k)$ number of variables and hard clauses [23].

Similarly, Boolean cardinality constraints of the form $Least_p(\{x_1, \ldots, x_n\}) = \sum_{i=1}^{n} x_i \geq p$ are encoded given the CNF encoding of the Cardinality Networks $CN_k^{\geq}([x_1, \ldots, x_n] \rightarrow [c_1, \ldots, c_n])$ that count a lower bound on the number of true assignments to Boolean variables $\{x_1, \ldots, x_n\}$ such that $\sum_{j=1}^{i} c_j \leq \min(\sum_{j=1}^{n} x_j, i)$ holds for all $i \in \{1, \ldots, k\}$. The detailed CNF encoding of $CN_k^{\geq}$ is also outlined in Appendix A. Given $CN_k^{\geq}$, Boolean cardinality con-

straint $Least_p(\{x_1, \ldots, x_n\})$ is defined as follows.

$$Least_p(\{x_1, \ldots, x_n\}) = \bigwedge_{i=n+1}^{r} (\neg x_i) \wedge (c_p) \\ \wedge CN_k^{\geq}([x_1, \ldots, x_{n+r}] \rightarrow [c_1, \ldots, c_k]) \quad (2)$$

Note that the cardinality constraint $\sum_{i=1}^{n} x_i \leq p$ is equivalent to $\sum_{i=1}^{n}(1-x_i) \geq n-p$. Since Cardinality Networks require the value of p to be less than or equal to $\frac{n}{2}$, Boolean cardinality constraints of the form $\sum_{i=1}^{n} x_i \leq p$ with $p > \frac{n}{2}$ must be converted into $Least_{n-p}(\{\neg x_1, \ldots, \neg x_n\})$.

Finally, a Boolean cardinality constraint $Most_p(\{x_1, \ldots, x_n\})$ is **generalized arc-consistent** if and only if *for every value assignment* $x_i = true/false$ *to every Boolean variable in the set* $\{x_1, \ldots, x_n\}$*, there exists feasible a value assignment* $x_j = true/false$ *to all the remaining Boolean variables* $x_{j \neq i} \in \{x_1, \ldots, x_n\}$. In practice, the ability to maintain generalized arc-consistency (GAC) through efficient algorithms such as unit propagation (as opposed to search) is one of the most important properties for the efficiency of a Boolean cardinality constraint encoded in CNF [24, 25, 23, 26]. It has been shown that both $Most_p(\{x_1, \ldots, x_n\})$ and $Least_p(\{x_1, \ldots, x_n\})$ encodings maintain GAC through unit propagation [23].

2.7. Binary Linear Programming Problem

As an alternative to WP-MaxSAT encodings of BNN transition models, we can also leverage Binary Linear Programs (BLPs). The 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 WP-MaxSAT, BLP is *NP-hard*. The state-of-the-art BLP solvers [27] utilize branch-and-bound algorithms and can handle cardinality constraints efficiently in the size of its encoding.

2.8. Generalized Landmark Constraints

In this section, we review generalized landmark constraints that are necessary for improving the planning accuracy of the learned models when infeasible

plans are generated. 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 [17]. The decomposition-based planner, *OpSeq* [17], incrementally updates generalized landmark constraints to find cost-optimal plans to classical planning problems.

3. Weighted Partial Maximum Boolean Satisfiability 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 WP-MaxSAT, which we denote as Factored Deep SAT Planner (FD-SAT-Plan+). FD-SAT-Plan+ uses the same learning and planning framework with HD-MILP-Plan [7] as visualized in Figure 2 where the ReLU-based DNN is replaced by a BNN [8] and the compilation of $\tilde{\Pi}$ is a WP-MaxSAT instead of a Mixed-Integer Linear Program (MILP).

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

3.2. Parameters

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

- $\bar{S}^{i,s}_I$ is the initial (i.e., at $t = 1$) value of the i-th bit of state variable $s \in S$.

- $In(x, i)$ 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 $j \in J(L)$.

The global constraints C and goal state constraints G are in the form of $\sum_{i=1}^{n} a_i x_i \leq p$, and the reward function Q is in the form of $\sum_{i=1}^{n} b_i x_i$ for state and action variables $x_i \in S \cup A$ where $a_i \in \mathbb{N}_{\geq 0}$ and $b_i \in \mathbb{R}_{\geq 0}$.

3.3. The WP-MaxSAT Compilation

Below, we define the WP-MaxSAT encoding of the learned factored planning problem $\tilde{\Pi}$ with BNNs. First, we present the hard clauses (i.e., clauses that must be satisfied) used in FD-SAT-Plan+.

3.3.1. Initial State Clauses

The following conjunction of hard clauses encode the initial state constraints I.

$$\bigwedge_{s \in S} \bigwedge_{1 \leq i \leq m} (Y^i_{s,1} \leftrightarrow \bar{S}^{i,s}_I) \tag{3}$$

where hard clause (3) set the initial values of the state variables at time step $t = 1$.

3.3.2. Bi-Directional Neuron Activation Encoding

Previous work [1] presented a neuron activation encoding for BNNs that is not efficient with respect to its encoding size using $O(nk)$ number of variables and hard clauses, and the computational effort required to maintain GAC. In this section, we present an efficient CNF encoding to model the activation behaviour of BNN binary unit $j \in J(l), l \in \{1, \ldots, L\}$ that requires only $O(nlog_2^2 k)$ variables and hard clauses, and maintains GAC through unit propagation.

Given input $\{x_1, \ldots, x_n\}$, activation threshold p and binary activation function $y = 1$ if $\sum_{i=1}^{n} x \geq p$, else $y = -1$, the output y of a binary neuron can be efficiently encoded in CNF by defining the Boolean variable v to represent the activation of the binary neuron such that $v = true$ if and only if $y = 1$, combining the cardinality networks $CN_k^{\leq}$ and $CN_k^{\geq}$ to count the number of variables from set $\{x_1, \ldots, x_n\}$ that are assigned to true, and adding the unit hard clauses for the auxiliary input variables in conjunction with a bi-directional activation hard clause as follows.

$$\bigwedge_{i=n+1}^{r} (\neg x_i) \wedge (v \leftrightarrow c_p) \tag{4}$$

In order to efficiently combine the cardinality networks, $CN_k^{\leq}$ and $CN_k^{\geq}$, and reduce the number of variables required in half, the conjunction of hard clauses defined in Appendix A are taken representing

1. the base case of Half Merging Networks (i.e., hard clauses (A.1) $\wedge$ hard clauses (A.21)),
2. the base case of Simplified Merging Networks (i.e., hard clauses (A.11) $\wedge$ hard clauses (A.21)),
3. the recursive case of Half Merging Networks (i.e., hard clauses (A.5) $\wedge$ hard clauses (A.22)),
4. the recursive case of Simplified Merging Networks (i.e., hard clauses (A.15) $\wedge$ hard clauses (A.23))

rather than naively taking the conjunction of the set of hard clauses in $CN_k^{\leq}$ and $CN_k^{\geq}$ using two separate sets of auxiliary Boolean counting variables c_j [23]. Intuitively, cardinality networks $CN_k^{\leq}$ and $CN_k^{\geq}$ together count $\min(\sum_{j=1}^{n} x_j, i) = \sum_{j=1}^{i} c_j$ by combining the respective bounds $\min(\sum_{j=1}^{n} x_j, i) \leq \sum_{j=1}^{i} c_j$ and $\sum_{j=1}^{i} c_j \leq \min(\sum_{j=1}^{n} x_j, i)$ for all $i \in \{1, \ldots, k\}$.

Instead of the neuron activation encoding used in prior work [1] that utilizes two separate sets of auxiliary Boolean counting variables c_i, where $v \rightarrow [\sum_{i=1}^{n} x_i \geq p]$ and $\neg v \rightarrow [\sum_{i=1}^{n} x_i \leq p - 1]$ are encoded with two different sets of auxiliary Boolean counting variables, the Bi-Directional encoding we use

here shares the same set of decision variables. Further, while the neuron activation encoding of prior work [1] uses Sequential Counters [24] for encoding the cardinality constraints using $O(np)$ number of variables and hard clauses, the Bi-Directional encoding presented here uses only $O(nlog_2^2k)$ number of variables and hard clauses as we will show by Lemma 1. Finally, the previous neuron activation encoding [1] has been shown not to preserve the GAC property through unit propagation [10] in contrast to the Bi-Directional Neuron Activation Encoding we use here which we show preserves GAC in Theorem 1. From here out for notational clarity, we will refer to the conjunction of hard clauses in Bi-Directional Neuron Activation Encoding as $Act_p(v, \{x_1, \ldots, x_n\})$. Furthermore, we will refer to the previous neuron activation encoding [1] as the Uni-Directional Neuron Activation Encoding.

3.3.3. BNN Clauses

Given the efficient CNF encoding $Act_p(v, \{x_1, \ldots, x_n\})$, we present the conjunction of hard clauses to model the complete BNN model.

$$\bigwedge_{1\leq t\leq H}\bigwedge_{s\in S}\bigwedge_{1\leq i\leq m}(Y^i_{s,t} \leftrightarrow Z_{In(s,i),1,t}) \tag{5}$$

$$\bigwedge_{1\leq t\leq H}\bigwedge_{a\in A}\bigwedge_{1\leq i\leq m}(X^i_{a,t} \leftrightarrow Z_{In(a,i),1,t}) \tag{6}$$

$$\bigwedge_{1\leq t\leq H}\bigwedge_{s\in S}\bigwedge_{1\leq i\leq m}(Y^i_{s,t+1} \leftrightarrow Z_{Out(s,i),L,t}) \tag{7}$$

$$\bigwedge_{1\leq t\leq H}\bigwedge_{2\leq l\leq L}\bigwedge_{\substack{j\in J(l),\\ p^*_j\leq\lceil\frac{|J(l-1)|}{2}\rceil}} Act_{p_j}(Z_{j,l,t}, \{Z_{i,l-1,t}|i\in J(l-1), w_{i,j,l-1}=1\}$$
$$\cup\{\neg Z_{i,l-1,t}|i\in J(l-1), w_{i,j,l-1}=-1\}) \tag{8}$$

$$\bigwedge_{1\leq t\leq H}\bigwedge_{2\leq l\leq L}\bigwedge_{\substack{j\in J(l),\\ p^*_j>\lceil\frac{|J(l-1)|}{2}\rceil}} Act_{p_j}(\neg Z_{j,l,t}, \{Z_{i,l-1,t}|i\in J(l-1), w_{i,j,l-1}=-1\}$$
$$\cup\{\neg Z_{i,l-1,t}|i\in J(l-1), w_{i,j,l-1}=1\}) \tag{9}$$

where activation constant p_j in hard clauses (8-9) are computed using the batch normalization parameters for binary unit $j \in J(l)$ in layer $l \in \{2, \ldots, L\}$ at

training time such that:

$$p_j^* = \left\lceil \frac{|J(l-1)| + \mu_{j,l} - \frac{\beta_{j,b}\sqrt{\sigma_{j,l}^2 + \epsilon_{j,l}}}{\gamma_{j,l}}}{2} \right\rceil$$
$$\text{if } p_j^* \leq \lceil \frac{|J(l-1)|}{2} \rceil \text{ then } p_j = p_j^*$$
$$\text{else } p_j = |J(l-1)| - p_j^* + 1$$

where $|x|$ denotes the size of set x. The computation of the activation constant $p_j, j \in J(l)$ ensures that p_j is less than or equal to the half size of the previous layer $|J(l-1)|$, as Bi-Directional Neuron Activation Encoding only counts upto $\lceil \frac{|J(l-1)|}{2} \rceil$.

Hard clauses (5-6) map the binary units at the input layer of the BNN (i.e., $l = 1$) to a unique state or action variable, respectively. Similarly, hard clause (7) maps the binary units at the output layer of the BNN (i.e., $l = L$) to a unique state variable. Hard clauses (8-9) encode the binary activation of every unit in the BNN.

3.3.4. Global Constraint Clauses

The following conjunction of hard clauses encode the global constraints C.

$$\bigwedge_{1 \leq t \leq H} C(\{Y_{s,t}^i | s \in S, 1 \leq i \leq m\}, \{X_{a,t}^i | a \in A, 1 \leq i \leq m\}) \tag{10}$$

where hard clause (10) represents domain-dependent global constraints on state and action variables. Some common examples of global constraints 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 by hard clauses (11-12) as follows.

$$\bigwedge_{1 \leq t \leq H} AtMost_1(\{X_{a,t}^1 | a \in A\}) \tag{11}$$

$$\bigwedge_{1 \leq t \leq H} AtMost_1(\{Y_{s,t}^1 | s \in S\}) \wedge (\bigvee_{s \in S} Y_{s,t}^1) \tag{12}$$

In general, linear global constraints in the form of $\sum_{i=1}^{n} a_i x_i \leq p$, such as bounds on state and action variables, can be encoded in CNF where a_i

are positive integer coefficients and x_i are decision variables with non-negative integer domains [28].

3.3.5. Goal State Clauses

The following conjunction of hard clauses encode the goal state constraints G.

$$G(\{Y^i_{s,H+1} | s \in S^G, 1 \leq i \leq m\}) \tag{13}$$

where hard clause (13) set the goal constraints on the state variables S_G at time step $t = H + 1$.

3.3.6. Reward Clauses

Given the reward function Q for each time step t is in the form of $\sum_{a \in A} \sum_{i=1}^{m} f^i_a X^i_{a,t} + \sum_{s \in S} \sum_{i=1}^{m} g^i_s Y^i_{s,t+1}$ the following weighted soft clauses (i.e., optional weighted clauses that may or may not be satisfied where each weight corresponds to the penalty of not satisfying a clause):

$$\bigwedge_{1 \leq t \leq H} \bigwedge_{1 \leq i \leq m} \Big(\bigwedge_{a \in A} (f^i_a, X^i_{a,t}) \wedge \bigwedge_{s \in S} (g^i_s, Y^i_{s,t+1}) \Big) \tag{14}$$

can be written to represent Q where $f^i_a, g^i_s \in \mathbb{R}_{\geq 0}$ are the weights of the soft clauses for each bit of action and state variables, respectively.

4. Binary Linear Programming Compilation of the Learned Factored Planning Problem

Given FD-SAT-Plan+, we present the Binary Linear Programming (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 hard clauses (3) and (5-7) with equivalent linear constraints as follows.

$$Y^i_{s,1} = \bar{S}^{i,s}_I \qquad \forall s \in S, 1 \leq i \leq m \tag{15}$$

$$Y^i_{s,t} = Z_{In(s,i),1,t} \qquad \forall 1 \leq t \leq H, s \in S, 1 \leq i \leq m \tag{16}$$

$$X^i_{a,t} = Z_{In(a,i),1,t} \qquad \forall 1 \leq t \leq H, a \in A, 1 \leq i \leq m \tag{17}$$

$$Y^i_{s,t+1} = Z_{Out(s,i),L,t} \qquad \forall 1 \leq t \leq H, s \in S, 1 \leq i \leq m \tag{18}$$

Given the activation constant p^*_j of binary unit $j \in J(l)$ in layer $l \in \{2, \ldots, L\}$, FD-BLP-Plan+ replaces hard clauses (8-9) representing the activation of binary unit j with the following linear constraints:

$$p^*_j Z_{j,l,t} \leq \sum_{\substack{i \in J(l-1), \\ w_{i,j,l-1}=1}} Z_{i,l-1,t} + \sum_{\substack{i \in J(l-1), \\ w_{i,j,l-1}=-1}} (1 - Z_{i,l-1,t})$$
$$\forall 1 \leq t \leq H, 2 \leq l \leq L, j \in J(l) \tag{19}$$

$$p'_j(1 - Z_{j,l,t}) \leq \sum_{\substack{i \in J(l-1), \\ w_{i,j,l-1}=-1}} Z_{i,l-1,t} + \sum_{\substack{i \in J(l-1), \\ w_{i,j,l-1}=1}} (1 - Z_{i,l-1,t})$$
$$\forall 1 \leq t \leq H, 2 \leq l \leq L, j \in J(l) \tag{20}$$

where $p'_j = |J(l-1)| - p^*_j + 1$.

Global constraint hard clauses (10) and goal state hard clauses (13) are compiled into linear constraints given they are in the form of $\sum_{i=1}^{n} a_i x_i \leq p$. Finally, the reward function Q with linear expressions is maximized over time steps $1 \leq t \leq H$ such that:

$$\max \sum_{t=1}^{H} \sum_{i=1}^{m} (\sum_{a \in A} f^i_a X^i_{a,t} + \sum_{s \in S} g^i_s Y^i_{s,t+1}) \tag{21}$$

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

Given that the plans found for the learned factored planning problem $\tilde{\Pi}$ by FD-SAT-Plan+ and FD-BLP-Plan+ can be infeasible to the factored planning

problem Π, we introduce an incremental algorithm for finding plans for Π by iteratively excluding invalid plans from the search space of FD-SAT-Plan+ and FD-BLP-Plan+. Similar to *OpSeq* [17], FD-SAT-Plan+ and FD-BLP-Plan+ are updated with the following generalized landmark hard clauses or constraints

$$\bigvee_{1 \leq t \leq H} \bigvee_{a \in A} \Big(\bigvee_{\substack{1 \leq i \leq m \\ (t,a,i) \in \pi_n}} (\neg X^i_{a,t}) \vee \bigvee_{\substack{1 \leq i \leq m \\ (t,a,i) \notin \pi_n}} (X^i_{a,t}) \Big) \tag{22}$$

$$\sum_{t=1}^{H} \sum_{a \in A} \Big(\sum_{\substack{i=1, \\ (t,a,i) \in \pi_n}}^{m} (1 - X^i_{a,t}) + \sum_{\substack{i=1, \\ (t,a,i) \notin \pi_n}}^{m} X^i_{a,t} \Big) \geq 1 \tag{23}$$

respectively, where π_n is the set of $1 \leq i \leq m$ bits of actions $a \in A$ executed at time steps $1 \leq t \leq H$ at the n-th iteration of the algorithm outlined by Algorithm 1.

Algorithm 1 Incremental Factored Planning Algorithm

1: $n = 1$, planner = FD-SAT-Plan+ or FD-BLP-Plan+
2: $\pi_n \leftarrow$ Solve planner.
3: **if** π_n is infeasible or π_n is a plan for Π **then**
4: **return** π_n
5: **else**
6: **if** planner = FD-SAT-Plan+ **then**
7: planner $\leftarrow$ hard clause (22)
8: **else**
9: planner $\leftarrow$ constraint (23)
10: $n \leftarrow n + 1$, go to line 2.

For a given horizon H, Algorithm 1 iteratively computes a set of actions π_n, or returns infeasibility for the learned factored planning problem $\tilde{\Pi}$. If the set of actions π_n is non-empty, we evaluate whether π_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 [29]. If the set of actions π_n constitutes a plan for Π, Algorithm 1 returns π_n as a plan. Otherwise, the planner is updated with the new set of generalized landmarks to exclude π_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. The outline of the proof can be found in Appendix B.

Next, we present a theoretical analysis of Bi-Directional Neuron Activation Encoding.

6. Theoretical Results

We now present some theoretical results on Bi-Directional Neuron Activation Encoding with respect to its encoding size and the number of variables used.

Lemma 1 (Encoding Size)**.** *Bi-Directional Neuron Activation Encoding requires* $O(nlog_2^2k)$ *variables and hard clauses.*

Proof. The Bi-Directional Neuron Activation Encoding shares the same set of variables as $CN_k^{\leq}$ and $CN_k^{\geq}$ encodings with the exception of variable v, and both $CN_k^{\leq}$ and $CN_k^{\geq}$ utilize $O(nlog_2^2k)$ variables [23]. Further, Bi-Directional Neuron Activation Encoding takes the conjunction of hard clauses for the base cases of Half Merging Networks and Simplified Merging Networks of $CN_k^{\leq}$ and $CN_k^{\geq}$, which can only increase the number of hard clauses required by a multiple of a linear constant (i.e., at most 2 times). Similarly, Bi-Directional Neuron Activation Encoding takes the conjunction of hard clauses for the recursive cases of Half Merging Networks and Simplified Merging Networks, which can only increase the number of hard clauses required by a multiple of a linear constant (i.e., at most 2 times). Given Bi-Directional Neuron Activation Encoding uses the same recursion structure as $CN_k^{\leq}$ and $CN_k^{\geq}$, the number of hard clauses used in Bi-Directional Neuron Activation Encoding is asymptotically bounded by the total encoding size of $CN_k^{\leq}$ plus $CN_k^{\geq}$, which is still $O(nlog_2^2k)$. □

Next, we will prove that Bi-Directional Neuron Activation Encoding has the GAC property through unit propagation, which is considered to be one of the

most important theoretical properties that facilitate the efficiency of a Boolean cardinality constraint encoded in CNF [24, 25, 23, 26].

Definition 1 (Generalized Arc-Consistency of Neuron Activation Encoding)**.** *A neuron activation encoding* $v \leftrightarrow [\sum_{i=1}^{n} x_i \geq p]$ *has the generalized arc-consistency property through unit propagation if and only if unit propagation is sufficient to deduce the following:*

1. *For any set* $X' \subset \{x_1, \ldots, x_n\}$ *with size* $|X'| = n - p$*, value assignment to variables* $v = true$*, and* $x_i = false$ *for all* $x_i \in X'$*, the remaining* p *variables from the set* $\{x_1, \ldots, x_n\} \backslash X'$ *are assigned to true,*
2. *For any set* $X' \subset \{x_1, \ldots, x_n\}$ *with size* $|X'| = p - 1$*, value assignment to variables* $v = false$*, and* $x_i = true$ *for all* $x_i \in X'$*, the remaining* $n - p + 1$ *variables from the set* $\{x_1, \ldots, x_n\} \backslash X'$ *are assigned to false,*
3. *Partial value assignment of* p *variables from* $\{x_1, \ldots, x_n\}$ *to true assigns variable* $v = true$*, and*
4. *Partial value assignment of* $n - p + 1$ *variables from* $\{x_1, \ldots, x_n\}$ *to false assigns variable* $v = false$

where $|x|$ *denotes the size of set* x*.*

Theorem 1 (Generalized Arc-Consistency of $Act_p(v, \{x_1, \ldots, x_n\})$)**.** *Bi-Directional Neuron Activation Encoding* $Act_p(v, \{x_1, \ldots, x_n\})$ *has the generalized arc-consistency (GAC) property through unit propagation.*

Proof. To show $Act_p(v, \{x_1, \ldots, x_n\})$ maintains GAC property through unit propagation, we need to show exhaustively for all four cases of Definition 1 that unit propagation is sufficient to maintain the GAC.

Case 1 ($\forall X' \subset \{x_1, \ldots, x_n\}$ where $|X'| = n - p$, $v = true$ and $x_i = false$ $\forall x_i \in X' \rightarrow x_i = true$ $\forall x_i \in \{x_1, \ldots, x_n\} \backslash X'$ by unit propagation): When $v = true$, unit propagation assigns $c_p = true$ using the hard clause $(\neg v \vee c_p)$. Given value assignment $x_i = false$ to variables $x_i \in X'$ for any set $X' \subset \{x_1, \ldots, x_n\}$ with size $|X'| = n - p$, it has been shown that unit propagation will set the remaining p variables from the set $\{x_1, \ldots, x_n\} \backslash X'$ to true using the conjunction

of hard clauses that encode $Least_p(\{x_1, \ldots, x_n\})$ [23] excluding the unit clause (c_p).

Case 2 ($\forall X' \subset \{x_1, \ldots, x_n\}$ where $|X'| = p-1$, $v = false$ and $x_i = true$ $\forall x_i \in X' \rightarrow x_i = false$ $\forall x_i \in \{x_1, \ldots, x_n\} \backslash X'$ by unit propagation): When $v = false$, unit propagation assigns $c_p = false$ using the hard clause $(v \vee \neg c_p)$. Given value assignment $x_i = true$ to variables $x_i \in X'$ for any set $X' \subset \{x_1, \ldots, x_n\}$ with size $|X'| = p-1$, it has been shown that unit propagation will set the remaining $n-p+1$ variables from the set $\{x_1, \ldots, x_n\} \backslash X'$ to false using the conjunction of hard clauses that encode $Most_p(\{x_1, \ldots, x_n\})$ [23] excluding the unit clause $(\neg c_{p+1})$.

Cases 3 ($\forall X' \subset \{x_1, \ldots, x_n\}$ where $|X'| = p$, $x_i = true$ $\forall x_i \in X' \rightarrow v = true$ by unit propagation) When p variables from the set $\{x_1, \ldots, x_n\}$ are set to true, it has been shown that unit propagation assigns the counting variable $c_p = true$ using the conjunction of hard clauses that encode $Most_p(\{x_1, \ldots, x_n\})$ [23] excluding the unit clause $(\neg c_{p+1})$. Given the assignment $c_p = true$, unit propagation assigns $v = true$ using the hard clause $(v \vee \neg c_p)$.

Cases 4 ($\forall X' \subset \{x_1, \ldots, x_n\}$ where $|X'| = n-p+1$, $x_i = false$ $\forall x_i \in X' \rightarrow v = false$ by unit propagation) When $n-p+1$ variables from the set $\{x_1, \ldots, x_n\}$ are set to false, it has been shown that unit propagation assigns the counting variable $c_p = false$ using the conjunction of hard clauses that encode $Least_p(\{x_1, \ldots, x_n\})$ [23] excluding the unit clause (c_p). Given the assignment $c_p = false$, unit propagation assigns $v = false$ using the hard clause $(\neg v \vee c_p)$. □

We now discuss the importance of our theoretical results in the context of both related work and the contributions of our paper. Amongst the state-of-the-art CNF encodings [10] that preserve GAC through unit propagation for constraint $v \rightarrow [\sum_{i=1}^{n} x_i \geq p]$, Bi-Directional Neuron Activation Encoding uses the smallest number of variables and hard clauses. The previous state-of-the-art CNF encoding for constraint $v \rightarrow [\sum_{i=1}^{n} x_i \geq p]$ is an extension of the Sorting Networks [30] and uses $O(nlog_2^2 n)$ number of variables and hard clauses [10].

In contrast, Bi-Directional Neuron Activation Encoding is an extension of the Cardinality Networks [23], and only uses $O(nlog_2^2k)$ number of variables and hard clauses as per Lemma 1 while maintaining GAC through unit propagation as per Theorem 1.

7. 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 compare the runtime efficiency of Bi-Directional Neuron Activation Encoding against the existing Uni-Directional Neuron Activation Encoding [1]. Fourth, 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. Finally, we demonstrate the effectiveness of Algorithm 1 to find a plan for the factored planning problem Π.

7.1. Domain Descriptions

The RDDL [29] formalism is extended to handle goal-specifications and used to describe the problem Π. Below, we summarize the extended deterministic RDDL domains used in the experiments, namely Navigation [31], Inventory Control (Inventory) [32], System Administrator (SysAdmin) [33, 31], and Cellda [18]. Detailed presentation of the RDDL domains and instances are provided in Appendix C.

Navigation. Models an agent in a two-dimensional (q-by-q) 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 [31]. Both the action and the state space are Boolean. We report the results on instances with three maze sizes q-by-q and three horizon settings H per maze size where $q \in \{3, 4, 5\}$, $H \in \{4, 5, 6, 7, 8, 9, 10\}$.

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 time, 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 per demand cycle length where $q \in \{2, 4\}$ and $H \in \{5, 6, 7, 8\}$.

SysAdmin. Models the behavior of a computer network of size q 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 [31]. 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 q and three horizon settings H where $q \in \{4, 5\}$ and $H \in \{2, 3, 4\}$.

Cellda. Models an agent in a two dimensional (4-by-4) dungeon cell. The agent Cellda must escape her cell through an initially locked door by obtaining the key without getting hit by her enemy. Each grid of the cell is made up of a grid type: i) regular grids which Cellda and her enemy can move from (or

to) deterministically up, down, right or left, and ii) blocks that neither Cellda nor her enemies can stand on. The enemy has a deterministic policy that is *unknown* to Cellda that will try to minimize the total Manhattan distance between itself and Cellda. Given the location of Cellda and the enemy, the adversarial deterministic policy P_q will always try to minimize the distance between the two by trying to move the enemy on axis $q \in \{x, y\}$. The state space is mixed; integer to describe the locations of Cellda and the enemy, and Boolean to describe whether the key is obtained or not and whether Cellda is alive or not. The action space is Boolean for moving up, down, right or left. The transition function T updates states as a function of the previous locations of Cellda and the enemy, the moving direction of Cellda, whether the key was obtained or not and whether Cellda was alive or not. We report results on instances with two adversarial deterministic policies P_q and three horizon settings H per policy where $q \in \{x, y\}$ and $H \in \{8, 9, 10, 11, 12\}$.

7.2. 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 from the RDDL-based domain simulator RDDLsim [29] using a simple stochastic exploration policy. For each instance of a domain, state transitions 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 [8]. Overall, Navigation instances required the smallest BNN structures for learning due to their purely Boolean state and action spaces, while both Inventory, SysAdmin and Cellda 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)	13:36:36:9	0.0
Navigation(4)	20:96:96:16	0.0
Navigation(5)	29:128:128:25	0.0
Inventory(2)	7:96:96:5	0.018
Inventory(4)	8:128:128:5	0.34
SysAdmin(4)	16:128:128:12	2.965
SysAdmin(5)	20:128:128:128:15	0.984
Cellda(x)	12:128:128:4	0.645
Cellda(y)	12:128:128:4	0.65

7.3. Planning Performance on the Learned Factored Planning Problems

In this section, we present the results of two computational comparisons. First, we test the efficiency of Bi-Directional Neuron Activation Encoding to the existing Uni-Directional Neuron Activation Encoding [1] to select the best WP-MaxSAT-based encoding for FD-SAT-Plan+. Second, we compare the effectiveness of using the selected WP-MaxSAT-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 a MacBookPro with 2.8 GHz Intel Core i7 16GB memory. For FD-SAT-Plan+ and FD-BLP-Plan+, we used MaxHS [22] with underlying LP-solver CPLEX 12.7.1 [27], and CPLEX 12.7.1 [27] solvers respectively, with 1 hour total time limit per domain instance.

7.3.1. Comparison of neuron activation encodings

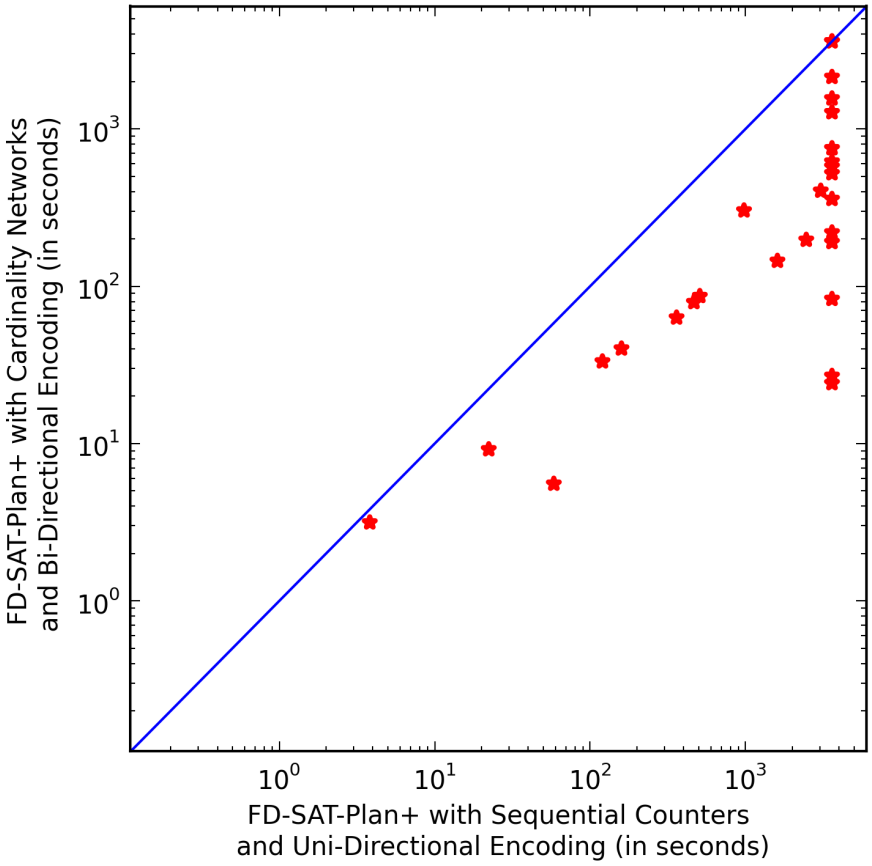


Figure 3: Timing comparison between for FD-SAT-Plan+ with Sequential Counters [24] and Uni-Directional Encoding [1] (x-axis) and Cardinality Networks [23] and Bi-Directional Encoding (y-axis). Over all problem settings, FD-SAT-Plan+ with Cardinality Networks and Bi-Directional Encoding signigicantly outperformed FD-SAT-Plan+ with Sequential Counters and Uni-Directional Encoding on all problem instances due to its i) smaller encoding size, and ii) generalized arc-consistency property.

The runtime efficiency of both neuron activation encodings are tested for the learned factored planning problems over 27 problem instances where we test our Bi-Directional Neuron Activation Encoding that utilizes Cardinality Networks [23] against the previous Uni-Directional Neuron Activation Encod-

ing [1] that uses Sequential Counters [24].

Figure 3 visualizes the runtime comparison of both neuron activation encodings. The inspection of Figure 3 clearly demonstrate that FD-SAT-Plan+ with Bi-Directional Neuron Activation Encoding signigicantly outperforms FD-SAT-Plan+ with Uni-Directional Neuron Activation Encoding in all problem instances due to its i) smaller encoding size (i.e., $O(nlog_2^2k)$ versus $O(np)$) with respect to both the number of variables and the number of hard clauses used as per Lemma 1, and ii) generalized arc-consistency property as per Theorem 1. Therefore, we use FD-SAT-Plan+ with Bi-Directional Neuron Activation Encoding in the remaining experiments and omit the results for FD-SAT-Plan+ with Uni-Directional Neuron Activation Encoding.

7.3.2. Comparison of FD-SAT-Plan+ and FD-BLP-Plan+

Next, we test the runtime efficiency of FD-SAT-Plan+ and FD-BLP-Plan+ for solving the learned factored planning problem.

Table 2: Summary of the computational results presented in Appendix D including the average runtimes in seconds for both FD-SAT-Plan+ and FD-BLP-Plan+ over all four domains for the learned factored planning problem within 1 hour time limit.

Domains	FD-SAT-Plan+	FD-BLP-Plan+
Navigation	529.11	1282.82
Inventory	54.88	0.54
SysAdmin	1627.35	3006.27
Cellda	344.03	285.45
Coverage	27/27	20/27
Optimality Proved	25/27	19/27

In Table 2, we present the summary of the computational results including the average runtimes in seconds, the total number of instances for which a feasible solution is returned (i.e., coverage), and the total number of instances for which an optimal solution is returned (i.e., optimality proved), for both FD-

SAT-Plan+ and FD-BLP-Plan+ over all four domains for the learned factored planning problem within 1 hour time limit. The analysis of Table 2 shows that FD-SAT-Plan+ covers all problem instances by returning an incumbent solution to the learned factored planning problem compared to FD-BLP-Plan+ which runs out of 1 hour time limit in 7 out of 27 instances before finding an incumbent solution. Similarly, FD-SAT-Plan+ proves the optimality of the solutions found in 25 out of 27 problem instances compared to FD-BLP-Plan+ which only proves the optimality of 19 out of 27 solutions within 1 hour time limit.

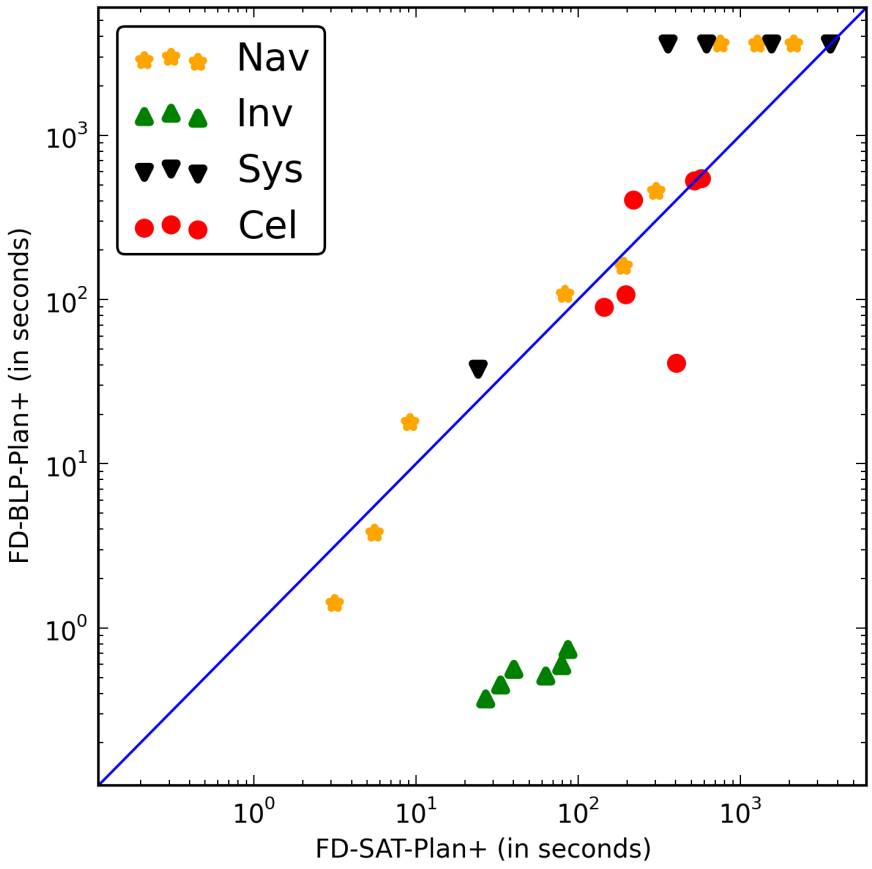


Figure 4: Timing comparison between FD-SAT-Plan+ (x-axis) and FD-BLP-Plan+ (y-axis). Over all problem settings, FD-BLP-Plan+ performed better on instances that require less than approximately 100 seconds to solve (i.e., computationally easy instances) whereas FD-SAT-Plan+ outperformed FD-BLP-Plan+ on instsances that require more than approximately 100 seconds to solve (i.e., computationally hard instances).

In Figure 4, we compare the runtime performances of FD-SAT-Plan+ (x-axis) and FD-BLP-Plan+ (y-axis) per instance labeled by their domain. The analysis of Figure 4 across all 27 intances shows that FD-BLP-Plan+ proved the optimality of problem instances from domains which require less computational demand (e.g., Inventory) more efficiently compared to FD-SAT-Plan+. In contrast, FD-SAT-Plan+ proved the optimality of problem instances from

domains which require more computational demand (e.g., SysAdmin) more efficiently compared to FD-BLP-Plan+. As the instances got harder to solve, FD-BLP-Plan+ timed-out more compared to FD-SAT-Plan+, mainly due to its inability to find incumbent solutions as evident from Table 2.

The detailed inspection of Figure 4 and Table 2 together with Table 1 shows that the computational efforts required to solve the benchmark instances increase signigicantly more for FD-BLP-Plan+ compared to FD-SAT-Plan+ as the learned BNN structure gets more complex (i.e., from smallest BNN structure of Inventory, to moderate size BNN structures of Navigation and Cellda, to the largest BNN structure of SysAdmin). Detailed presentation of the run time results for each instance are provided in Appendix D.

7.4. Planning Performance on the Factored Planning Problems

Finally, we test the planning efficiency of the incremental factored planning algorithm, namely Algorithm 1, for solving the factored planning problem Π.

Table 3: Summary of the computational results presented in Appendix D including the average runtimes in seconds for both FD-SAT-Plan+ and FD-BLP-Plan+ over all four domains for the factored planning problem within 1 hour time limit.

Domains	FD-SAT-Plan+	FD-BLP-Plan+
Navigation	529.11	1282.82
Inventory	68.88	0.66
SysAdmin	2463.79	3006.27
Cellda	512.51	524.53
Coverage	23/27	19/27
Optimality Proved	23/27	19/27

In Table 3, we present the summary of the computational results including the average runtimes in seconds, the total number of instances for which a feasible solution is returned (i.e., coverage), and the total number of instances for which an optimal solution is returned (i.e., optimality proved), for both

FD-SAT-Plan+ and FD-BLP-Plan+ using Algorithm 1 over all four domains for the factored planning problem within 1 hour time limit. The analysis of Table 3 shows that FD-SAT-Plan+ with Algorithm 1 covers 23 out of 27 problem instances by returning an incumbent solution to the factored planning problem compared to FD-BLP-Plan+ with Algorithm 1 which runs out of 1 hour time limit in 8 out of 27 instances before finding an incumbent solution. Similarly, FD-SAT-Plan+ with Algorithm 1 proves the optimality of the solutions found in 23 out of 27 problem instances compared to FD-BLP-Plan+ with Algorithm 1 which only proves the optimality of 19 out of 27 solutions within 1 hour time limit.

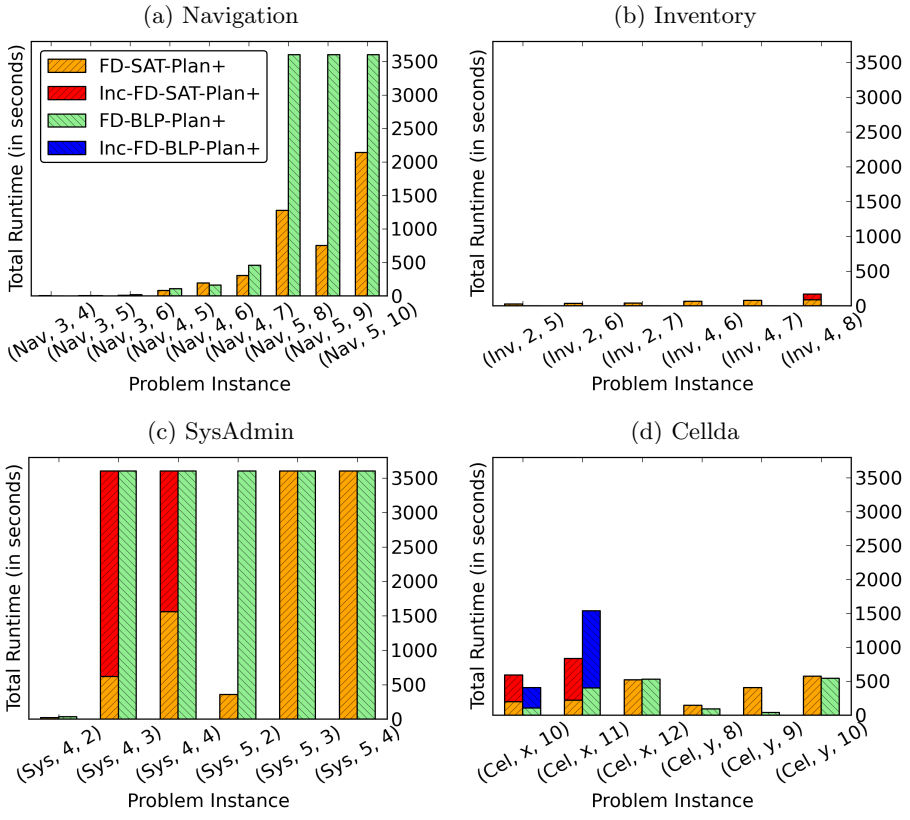


Figure 5: Timing comparison between FD-SAT-Plan+ (orange/red) and FD-BLP-Plan+ (green/blue) for solving the learned factored planning problems (orange/green) and the factored planning problems (red/blue) per domain.

Figures (5a-5d) visualize the comparative runtime performace of Algorithm 1 with i) FD-SAT-Plan+ (orange/red) and ii) FD-BLP-Plan+ (green/blue) per domain where the additional computational requirement of solving the factored planning problem is stacked on top of the computational requirement of solving the learned factored planning problem. The detailed inspection of Figures 5a, 5b and 5d demonstrate that the constraint generation algorithm successfully verified the plans found for the factored planning problem Π in three out of four domains with low computational cost. In the contrast, the incremental factored planning algorithm spent significantly more time in SysAdmin domain as evident in Figure 5c. Over all instances, we observed that at most 5 instances required constraint generation to find a plan where the maximum number of constraints required was at least 6; namely for (Sys,4,3) instance. Detailed presentation of the run time results and the number of generalized landmark constraints generated for each instance are provided in Appendix D.

8. 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 WP-MaxSAT (FD-SAT-Plan+) and a BLP (FD-BLP-Plan+) encodings, 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. Theoretically have shown that our SAT-based Bi-Directional Neuron Activation Encoding is asymptotically the most compact encoding in the literature, and has the generalized arc-consistency property through unit propagation, which is one of the most important efficiency indicators of a SAT-based encoding.

We further introduced a finite-time incremental factored planning algorithm based on generalized landmark constraints that improve planning accuracy of both FD-SAT-Plan+ and FD-BLP-Plan+. Experimentally, we demonstrate the

computational efficiency of our Bi-Directional Neuron Activation Encoding in comparison to the Uni-Directional Neuron Activation Encoding [1]. Overall, our empirical results showed we can accurately learn complex state transition models using BNNs and demonstrated strong performance in both the learned and original domains. In sum, this work provides two novel and efficient factored state and action transition learning and planning encodings for BNN-learned transition models, thus providing new and effective tools for the data-driven model-based planning community.

Appendices

Appendix A. CNF Encoding of the Cardinality Networks

The CNF encoding of k-Cardinality Networks ($CN_k^{\leq}$) is as follows [23].

Half Merging Networks. Given two sequences of Boolean variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, Half Merging (HM) Networks merge inputs into a single sequence of size $2n$ using the CNF encoding as follows.

For input size $n = 1$:

$$HM([x_1], [y_1] \rightarrow [c_1, c_2]) = (\neg x_1 \vee \neg y_1 \vee c_2) \wedge (\neg x_1 \vee c_1) \wedge (\neg y_1 \vee c_1) \quad \text{(A.1)}$$

For input size $n > 1$:

$$HM([x_1, \ldots, x_n], [y_1, \ldots, y_n] \rightarrow [d_1, c_2, \ldots, c_{2n-1}, e_n]) = (H_o \wedge H_e \wedge H') \quad \text{(A.2)}$$

$$H_o = HM([x_1, x_3 \ldots, x_{n-1}], [y_1, y_3 \ldots, y_{n-1}] \rightarrow [d_1, \ldots, d_n]) \quad \text{(A.3)}$$

$$H_e = HM([x_2, x_4 \ldots, x_n], [y_2, y_4 \ldots, y_n] \rightarrow [e_1, \ldots, e_n]) \quad \text{(A.4)}$$

$$H' = \bigwedge_{i=1}^{n-1} (\neg d_{i+1} \vee \neg e_i \vee c_{2i+1}) \wedge (\neg d_{i+1} \vee c_{2i}) \wedge (\neg e_i \vee c_{2i}) \quad \text{(A.5)}$$

Half Sorting Networks. Given a sequence of Boolean variables $x_1, \ldots, x_{2n}$, Half Sorting (HS) Networks sort the variables with respect to their value assignment as follows.

For input size $2n = 2$:

$$HS([x_1, x_2] \to [c_1, c_2]) = HM([x_1], [x_2] \to [c_1, c_2]) \tag{A.6}$$

For input size $2n > 2$:

$$HS([x_1, \ldots, x_{2n}] \to [c_1, \ldots, c_{2n}]) = (H_1 \wedge H_2 \wedge H_M) \tag{A.7}$$

$$H_1 = HS([x_1, \ldots, x_n] \to [d_1, \ldots, d_n]) \tag{A.8}$$

$$H_2 = HS([x_{n+1}, \ldots, x_{2n}] \to [d'_1, \ldots, d'_n])) \tag{A.9}$$

$$H_M = HM([d_1, \ldots, d_n], [d'_1, \ldots, d'_n] \to [c_1, \ldots, c_{2n}]) \tag{A.10}$$

Simplified Merging Networks. Given two sequences of Boolean variables $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$, Simplified Merging (SM) Networks merge inputs into a single sequence of size $2n$ using the CNF encoding as follows.

For input size $n = 1$:

$$SM([x_1], [y_1] \to [c_1, c_2]) = HM([x_1], [y_1] \to [c_1, c_2]) \tag{A.11}$$

For input size $n > 1$:

$$SM([x_1, \ldots, x_n], [y_1, \ldots, y_n] \to [d_1, c_2, \ldots, c_{n+1}]) = (S_o \wedge S_e \wedge S') \tag{A.12}$$

$$S_o = SM([x_1, x_3 \ldots, x_{n-1}], [y_1, y_3 \ldots, y_{n-1}] \to [d_1, \ldots, d_{\frac{n}{2}+1}]) \tag{A.13}$$

$$S_e = SM([x_2, x_4 \ldots, x_n], [y_2, y_4 \ldots, y_n] \to [e_1, \ldots, e_{\frac{n}{2}+1}]) \tag{A.14}$$

$$S' = \bigwedge_{i=1}^{n/2} (\neg d_{i+1} \vee \neg e_i \vee c_{2i+1}) \wedge (\neg d_{i+1} \vee c_{2i}) \wedge (\neg e_i \vee c_{2i}) \tag{A.15}$$

Note that unlike HM, SM counts the number of variables assigned to true from input sequences $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$ upto $n + 1$ bits instead of $2n$.

k-Cardinality Networks. Given a sequence of Boolean variables $x_1, \ldots, x_n$ with $n = ku$ where $p, u \in \mathbb{N}$ and $k = \lceil \log_2(p) \rceil$, the CNF encoding of k-Cardinality Networks ($CN_k^{\leq}$) is as follows.

For input size $n = k$:

$$CN_k^{\leq}([x_1, \ldots, x_n] \to [c_1, \ldots, c_n]) = HS([x_1, \ldots, x_n] \to [c_1, \ldots, c_n]) \tag{A.16}$$

For input size $n > k$:

$$CN_k^{\leq}([x_1, \ldots, x_n] \rightarrow [c_1, \ldots, c_k]) = (C_1 \wedge C_2 \wedge C_M) \tag{A.17}$$

$$C_1 = CN_k^{\leq}([x_1, \ldots, x_k] \rightarrow [d_1, \ldots, d_k]) \tag{A.18}$$

$$C_2 = CN_k^{\leq}([x_{k+1}, \ldots, x_n] \rightarrow [d'_1, \ldots, d'_k]) \tag{A.19}$$

$$C_M = SM([d_1, \ldots, d_k], [d'_1, \ldots, d'_k] \rightarrow [c_1, \ldots, c_{k+1}]) \tag{A.20}$$

The CNF Encoding of $CN_k^{\geq}$. The CNF encoding of $CN_k^{\geq}([x_1, \ldots, x_{n+r}] \rightarrow [c_1, \ldots, c_k])$ replaces the hard clauses (A.1) and (A.11) of HM and SM with the hard clause:

$$(x_1 \vee y_1 \vee \neg c_1) \wedge (x_1 \vee \neg c_2) \wedge (y_1 \vee \neg c_2) \tag{A.21}$$

, replaces the hard clause (A.5) of HM with the hard clauses:

$$\bigwedge_{i=1}^{n-1} (d_{i+1} \vee e_i \vee \neg c_{2i}) \wedge (d_{i+1} \vee \neg c_{2i+1}) \wedge (e_i \vee \neg c_{2i+1}) \tag{A.22}$$

, replaces the hard clause (A.15) of SM with the hard clauses:

$$\bigwedge_{i=1}^{n/2} (d_{i+1} \vee e_i \vee \neg c_{2i}) \wedge (d_{i+1} \vee \neg c_{2i+1}) \wedge (e_i \vee \neg c_{2i+1}) \tag{A.23}$$

.

Appendix B. Proof for Algorithm 1

Given hard clauses (5-9) and Theorem 1, Corollary 1 follows.

Corollary 1 (Forward Pass)**.** *Given complete value assignments to action $\bar{X}_{a,t}^i$ and state $\bar{Y}_{s,t}^i$ variables for time steps $t \in \{1, \ldots, H\}$, hard clauses (5-9) assign values to all state variables $\bar{Y}_{s,t+1}^i$ through unit propagation such that $\tilde{T}(\bar{Y}_{s,t}, \bar{X}_{a,t}) = \bar{Y}_{s,t+1}$ where $\bar{X}_{a,t} = \sum_{i=1}^m 2^{i-1} \bar{X}_{a,t}^i$ and $\bar{Y}_{s,t} = \sum_{i=1}^m 2^{i-1} \bar{Y}_{s,t}^i$.*

Theorem 2 (Correctness of Encodings)**.** *Let $\tilde{V} = \{\langle \bar{A}_1^1, \ldots, \bar{A}_1^H \rangle, \ldots, \langle \bar{A}_j^1, \ldots, \bar{A}_j^H \rangle, \ldots, \langle \bar{A}_n^1, \ldots, \bar{A}_n^H \rangle\}$ be the set of all feasible action variable assignments for the learned deterministic factored planning problem $\tilde{\Pi}$ upto m-bits of precision for given horizon H. Further let V' denote the feasibility space of the planner of choice. Sets V' and $\tilde{V}$ are equivalent.*

Proof by Contradiction. Case 1 (There exists $\pi_j = \langle \bar{A}_j^1, \ldots, \bar{A}_j^H \rangle$ such that $\pi_j \in \tilde{V}$ and $\pi_j \notin V'$): From the definition of the decision variable $X_{a,t}^i$ in Section 3.1 and the values of every action variable $\bar{a}_j^t \in \bar{A}_j^t$ for all time steps $t \in \{1, \ldots, H\}$, the value of every $\bar{X}_{a,t}^i$ is set using the binarization formula $\bar{a}_j^t = \sum_{i=1}^m 2^{i-1} X_{a,t}^i$. Similarly, given the initial values of the state variables, the binarized values can be set to $\bar{S}_I^{i,s}$ for each state variable $s \in S$ and bit i which sets the values of each decision variable $\bar{Y}_{s,1}^i$ using hard clauses (3). Given the values of $\bar{X}_{a,t}^i$ and $\bar{Y}_{s,1}^i$, values of $\bar{Y}_{s,t+1}^i$ can be propagated sequentially for time steps $t \in \{1, \ldots, H\}$ through unit propagation (from Corollary 1). The value assignment to all state variables $\bar{Y}_{s,t}^i$ for time steps $t \in \{2, \ldots, H+1\}$ is feasible and implies $\pi_j \in V'$ since there is one-to-one correspondece between C and hard clauses (10), and G and hard clause (13), which is a contradiction.

Case 2 (There exists $\pi_j = \langle \bar{A}_j^1, \ldots, \bar{A}_j^H \rangle$ such that $\pi_j \notin \tilde{V}$ and $\pi_j \in V'$): Every action variable $a_j^t \in A_j^t$ is assigned to its respective value $\bar{a}_j^t$, given the values of its respective decision variables $\bar{X}_{a,t}^i$ such that $a_j^t = \sum_{i=1}^m 2^{i-1} \bar{X}_{a,t}^i$ for all time steps $t \in \{1, \ldots, H\}$. Similarly, every state variable $s_j^t \in S_j^t$ is assigned to its respective value $\bar{s}_j^t$ for all time steps $t \in \{1, \ldots, H+1\}$. Initial I and goal state G constraints are satisfied for values $\bar{S}_j^1$ and $\bar{S}_j^{H+1}$ since there is one-to-one correspondece between I and hard clauses (3), and G and hard clause (13). Similarly, global constraints C are satisfied for values of action $\bar{A}_j^t$ and state $\bar{S}_j^t$ variables since there is one-to-one correspondece between C and hard clauses (10). Given the learned transition function $\tilde{T}$ is a BNN, and the values of $\bar{A}_j^t$ and $\bar{S}_j^t$, values of state variables for the next time step $t+1$ can be propagated sequentially for time steps $t \in \{1, \ldots, H\}$ through forward inference such that $\tilde{T}(\bar{S}_j^t, \bar{A}_j^t) = \bar{S}_j^{t+1}$ where $\bar{a}_j^t = \sum_{i=1}^m 2^{i-1} \bar{X}_{a,t}^i$ and $\bar{s}_j^t = \sum_{i=1}^m 2^{i-1} \bar{Y}_{s,t}^i$ (from Corollary 1), and implies $\pi_j \in \tilde{V}$, which is a contradiction. □

Theorem 3 (Finiteness of Algorithm 1)**.** *Let* $\Pi = \langle S, A, C, T, I, G, Q \rangle$ *be a tuple representing a deterministic factored planning problem. Further let* $\tilde{\Pi} = \langle S, A, C, \tilde{T}, I, G, Q \rangle$ *represent the learned deterministic factored planning problem. For a given horizon* H *and* m*-bit precision, Algorithm 1 returns either a*

feasible plan $\pi = \langle \bar{A}^1, \ldots, \bar{A}^H \rangle$ *to* Π *, or proves in finite number of iterations* $n \leq 2^{|A| \times m \times H}$ *that there does not exist a plan to both* $\tilde{\Pi}$ *and* Π*.*

Proof by Induction. Let $\tilde{V}$ be the set of all feasible action variable assignments for the learned deterministic factored planning problem $\tilde{\Pi}$ upto m-bits of precision for given horizon H. Similarly, let V be the set of all feasible action variable assignments for the deterministic factored planning problem Π. From Theorem 2, $\tilde{V}$ is initially (i.e., iteration $n = 1$) equal to the feasibility space V' of the planner of choice such that $V' = \tilde{V}$.

Base Case ($n = 1$): In the first iteration $n = 1$, Algorithm 1 either proves infeasibility of $\tilde{V}$ if and only if $V' = \emptyset$, or a value assignment to action variables $\pi_1 = \langle \bar{A}_1^1, \ldots, \bar{A}_1^H \rangle$ if and only if $V' \neq \emptyset$ from Theorem 2. If the planner returns infeasibility of V', Algorithm 1 terminates. Otherwise, value assignments to action variables π_1 are sequentially simulated for time steps $t \in \{1, \ldots, H\}$ from initial state $S^t = \bar{S}^I$ using state transition function T with respect to i) the domains of state variables, ii) global constraints C and iii) goal state constraint G. If the domain simulator verifies all the propagated values of state variables as feasible with respect to i), ii) and iii), Algorithm 1 terminates. Otherwise, value assignments to action variables $\pi_1 = \langle \bar{A}_1^1, \ldots, \bar{A}_1^H \rangle$ are used to generate a generalized landmark hard clause (or constraint) that is added back to the planner, which only removes π_1 from the solution space V' such that $V' \leftarrow V' \setminus \pi_1$.

Induction Hypothesis ($n < i$): Assume that upto iteration $n < i$, Algorithm 1 removes exactly n unique solutions where each solution π_j corresponds to a unique value assignment to action variables $\pi_j = \langle \bar{A}_j^1, \ldots, \bar{A}_j^H \rangle$ for iterations $j \in \{1, \ldots, n\}$ from the solution space V' such that $V' = \tilde{V} \setminus \{\pi_1, \ldots, \pi_j, \ldots, \pi_n\}$.

Induction Step ($n = i$): Let $n = i$ be the next iteration of Algorithm 1. In iteration $n = i$, Algorithm 1 either proves $V \cap \tilde{V} = \emptyset$ if and only if $V' = \emptyset$, or a value assignment to action variables $\pi_i = \langle \bar{A}_i^1, \ldots, \bar{A}_i^H \rangle$ if and only if $V' \neq \emptyset$ from Theorem 2. If the planner returns infeasibility of V', Algorithm 1 terminates. Otherwise, value assignments to action variables π_i are sequentially

simulated for time steps $t \in \{1, \ldots, H\}$ from initial state $S^t = \bar{S}^I$ using state transition function T with respect to i) the domains of state variables, ii) global constraints C and iii) goal state constraint G. If the domain simulator verifies all the propagated values of state variables as feasible with respect to i), ii) and iii), Algorithm 1 terminates. Otherwise, value assignments to action variables $\pi_i = \langle \bar{A}_i^1, \ldots, \bar{A}_i^H \rangle$ are used to generate a generalized landmark hard clause (or constraint) that is added back to the planner, which only removes π_i from the solution space V' such that $V' \leftarrow V' \setminus \pi_i$.

By induction in at most $n = |A| \times m \times H$ iterations, Algorithm 1 either i) proves $V \cap \tilde{V} = \emptyset$ by removing all possible unique value assignments to action variables $\{\pi_1, \ldots, \pi_j, \ldots, \pi_n\}$ from the solution space of V' (i.e., $V' = \emptyset$), or ii) returns a value assignment to action variables $\pi_n = \langle \bar{A}_n^1, \ldots, \bar{A}_n^H \rangle$ that is also feasible to the deterministic factored planning problem Π such that $\pi_n \in V$, and terminates.

□

Appendix C. Domain Specifications

Appendix C.1. Navigation,3

```
// Navigation
// Original Author: Scott Sanner
// Modified by: Buser Say

domain navigation
requirements = {
reward-deterministic
};

types {
xpos : object;
ypos : object;
```

};

pvariables {

NORTH(ypos, ypos) : {non-fluent, bool, default = false};
SOUTH(ypos, ypos) : {non-fluent, bool, default = false};
EAST(xpos, xpos) : {non-fluent, bool, default = false};
WEST(xpos, xpos) : {non-fluent, bool, default = false};
P(xpos, ypos) : {non-fluent, real, default = 0.0};
GOAL-robot-at(xpos,ypos) : {non-fluent, bool, default = false};

// state variables
robot-at(xpos, ypos) : {state-fluent, bool, default = false};

// action variables
move-north : {action-fluent, bool, default = false};
move-south : {action-fluent, bool, default = false};
move-east : {action-fluent, bool, default = false};
move-west : {action-fluent, bool, default = false};
};

cpfs {

robot-at'(?x,?y) =
if ((move-north ∧ exists_{?y2 : ypos} [SOUTH(?y,?y2) ∧ robot-at(?x,?y) ∧ P(?x, ?y2) < 0.5]) | (move-south ∧ exists_{?y2 : ypos} [NORTH(?y,?y2) ∧ robot-at(?x,?y) ∧ P(?x, ?y2) < 0.5]) | (move-east ∧ exists_{?x2 : xpos} [WEST(?x,?x2) ∧ robot-at(?x,?y) ∧ P(?x2, ?y) < 0.5]) | (move-west ∧ exists_{?x2 : xpos} [EAST(?x,?x2) ∧ robot-at(?x,?y) ∧ P(?x2, ?y) < 0.5]))
    then false
else if ((move-north ∧ exists_{?y2 : ypos} [SOUTH(?y,?y2) ∧ robot-at(?x,?y)

```
∧ P(?x, ?y2) > 0.5] ) | ( move-south ∧ exists_{?y2 : ypos} [ NORTH(?y,?y2)
∧ robot-at(?x,?y) ∧ P(?x, ?y2) > 0.5] ) | ( move-east ∧ exists_{?x2 : xpos}
[ WEST(?x,?x2) ∧ robot-at(?x,?y) ∧ P(?x2, ?y) > 0.5] ) | ( move-west ∧ ex-
ists_{?x2 : xpos} [ EAST(?x,?x2) ∧ robot-at(?x,?y) ∧ P(?x2, ?y) > 0.5] ))
    then true
else if (( move-north ∧ exists_{?y2 : ypos} [ NORTH(?y,?y2) ∧ robot-at(?x,?y2)
] ) | ( move-south ∧ exists_{?y2 : ypos} [ SOUTH(?y,?y2) ∧ robot-at(?x,?y2) ]
) | ( move-east ∧ exists_{?x2 : xpos} [ EAST(?x,?x2) ∧ robot-at(?x2,?y) ] ) | (
move-west ∧ exists_{?x2 : xpos} [ WEST(?x,?x2) ∧ robot-at(?x2,?y) ] ))
    then if ( P(?x, ?y) ¿ 0.5 ) then false
    else true
else robot-at(?x,?y);

};

// reward: minimize the number of actions
reward = -1*(move-up + move-down + move-right + move-left);

state-action-constraints {

// constraint 1: mutual exclusion of actions
move-north + move-south + move-west + move-east ≤ 1;

};

}

non-fluents nf_navigation_3 {
domain = navigation;
objects {
xpos : {x1,x2,x3};
```

```
ypos : {y1,y2,y3};
};
non-fluents {
NORTH(y2,y1);
NORTH(y3,y2);
SOUTH(y1,y2);
SOUTH(y2,y3);
EAST(x1,x2);
EAST(x2,x3);
WEST(x2,x1);
WEST(x3,x2);
P(x2,y2) = 0.9;
P(x3,y2) = 0.9;
};
}

instance navigation_3
domain = navigation;
non-fluents = nf_navigation_3;
init-state {
robot-at(x2,y1);
};
goal-state {
robot-at(x2,y3);
};
max-nondef-actions = 1;
horizon = 4; //5,6
discount = 1.0;
}
```

Appendix C.2. Navigation,4

The domain and instance files are identical to that of Navigation,3 except the following changes in the instance file as follows.

```
non-fluents nf_navigation_4 {
domain = navigation;
objects {
xpos : {x1,x2,x3,x4};
ypos : {y1,y2,y3,y4};
};
non-fluents {
NORTH(y2,y1);
NORTH(y3,y2);
NORTH(y4,y3);
SOUTH(y1,y2);
SOUTH(y2,y3);
SOUTH(y3,y4);
EAST(x1,x2);
EAST(x2,x3);
EAST(x3,x4);
WEST(x2,x1);
WEST(x3,x2);
WEST(x4,x3);
P(x3,y2) = 0.9;
P(x4,y2) = 0.9;
P(x3,y3) = 0.9;
P(x4,y3) = 0.9;
};
}

instance navigation_4
```

```
domain = navigation;
non-fluents = nf_navigation_4;
init-state {
robot-at(x3,y4);
};
goal-state {
robot-at(x3,y1);
};
max-nondef-actions = 1;
horizon = 5; //6,7
discount = 1.0;
}
```

Appendix C.3. Navigation,5

The domain and instance files are identical to that of Navigation,3 except the following changes in the instance file as follows.

```
non-fluents nf_navigation_5 {
domain = navigation;
objects {
xpos : {x1,x2,x3,x4,x5};
ypos : {y1,y2,y3,y4,x5};
};
non-fluents {
NORTH(y2,y1);
NORTH(y3,y2);
NORTH(y4,y3);
NORTH(y5,y4);
SOUTH(y1,y2);
SOUTH(y2,y3);
SOUTH(y3,y4);
```

```
SOUTH(y4,y5);
EAST(x1,x2);
EAST(x2,x3);
EAST(x3,x4);
EAST(x4,x5);
WEST(x2,x1);
WEST(x3,x2);
WEST(x4,x3);
WEST(x5,x4);
P(x3,y3) = 0.9;
P(x4,y3) = 0.9;
P(x5,y3) = 0.9;
P(x3,y4) = 0.9;
P(x4,y4) = 0.9;
P(x5,y4) = 0.9;
};
}

instance navigation_5
domain = navigation;
non-fluents = nf_navigation_5;
init-state {
robot-at(x4,y5);
};
goal-state {
robot-at(x4,y1);
};
max-nondef-actions = 1;
horizon = 8; //9,10
discount = 1.0;
}
```

Appendix C.4. Inventory 2

```
domain inventory {

types {
commodity : object;
};

pvariables {
MAX_INVENTORY : { non-fluent, int, default = 15 };
THRESHOLD(commodity) : { non-fluent, int, default = 2 };
RESUPPLY_CONSTANT(commodity) : { non-fluent, int, default = 5 };
PREV(commodity) : { non-fluent, int, default = c1 };

// state variables
quant(commodity) : { state-fluent, int, default = 0 };
threshold_met(commodity) : { state-fluent, bool, default = true };
month : { state-fluent, int, default = 0 };

// intermediate fluents
resupply_quant(commodity) : { interm-fluent, int };
unmet(commodity) : { interm-fluent, int };
after_demand_quant(commodity) : { interm-fluent, int };
after_demand_sum : { interm-fluent, int };
rcum_prev(commodity) : { interm-fluent, int };
rcum(commodity) : { interm-fluent, int };

// action variables
resupply(commodity) : { action-fluent, bool, default = 0 };

};
```

```
cpfs {

resupply_quant(?c) = resupply(?c)*RESUPPLY_CONSTANT(?c);

after_demand_quant(?c) =
if (month == 0) then max[ 0, ceil[ quant(?c) - 0 ]]
else max[ 0, ceil[ quant(?c) - 3 ]] ;

after_demand_sum = sum_{ ?c : commodity } after_demand_quant(?c);

unmet(?c) =
if (month == 0) then max[ 0, ceil[0 - quant(?c)]]
else max[ 0, ceil[3 - quant(?c)]];

rcum_prev(?c) =
if(?c == $c1) then 0
else rcum(PREV(?c));

rcum(?c) =
if((after_demand_sum + resupply_quant(?c) + rcum_prev(?c)) < MAX_INVENTORY)
    then (resupply_quant(?c) + rcum_prev(?c))
else (max[ 0, (MAX_INVENTORY - (rcum_prev(?c) + after_demand_sum)) ] +
rcum_prev(?c));

threshold_met'(?c) =
if( (unmet(?c) ≤ THRESHOLD(?c)) ∧ (THRESHOLD(?c) ≥ after_demand_quant(?c))
)
    then true
else false;

quant'(?c) = after_demand_quant(?c) + (rcum(?c) - rcum_prev(?c));
```

```
month' = if(month == 1) then 0
else month + 1;

};

// reward: minimize ordering cost
reward = -(sum_{?c : commodity} quant(?c));

action-preconditions {

// constraint 1: capacity constraints
(sum_{?c : commodity} quant(?c)) ≤ MAX_INVENTORY;

// constraint 2: capacity constraints
(sum_{?c : commodity} quant(?c)) ≥ 0;

// constraint 3: threshold quantity must be met
forall_{?c : commodity} [ threshold_met(?c) ];

};

}

non-fluents nf_inventory_2{
domain = inventory;
objects {
commodity : { c1 };
};

non-fluents {
```

```
DEMAND(c1) = 3;
PREV(c1) = $c1;
};
}

instance inventory_2 {
domain = inventory;
non-fluents = nf_inventory_2;
max-nondef-actions = 1;
horizon = 5; //6,7
discount = 1.0;
}
```

Appendix C.5. Inventory 4

The domain and instance files are identical to that of Inventory,2 except the following changes in the domain and instance files as follows.

```
domain inventory {
...

cpfs {
...
after_demand_quant(?c) =
if (month == 0) then max[ 0, ceil[ quant(?c) - 0 ]]
else if (month == 1) then max[ 0, ceil[ quant(?c) - 1 ]]
else if (month == 2) then max[ 0, ceil[ quant(?c) - 2 ]]
else max[ 0, ceil[ quant(?c) - 3 ]] ;
...
unmet(?c) =
if (month == 0) then max[ 0, ceil[0 - quant(?c)]]
else if (month == 1) then max[ 0, ceil[1 - quant(?c)]]
```

```
else if (month == 2) then max[ 0, ceil[2 - quant(?c)]]
else max[ 0, ceil[3 - quant(?c)]];
...
month' = if(month == 3) then 0
else month + 1;

};
...
}

non-fluents nf_inventory_4{
...
}

instance inventory_4 {
domain = inventory;
non-fluents = nf_inventory_4;
max-nondef-actions = 1;
horizon = 6; //7,8
discount = 1.0;
}
```

Appendix C.6. System Admin,4

```
// System Admin
// Original Author: Scott Sanner
// Modified by: Buser Say

domain sysadmin {

requirements = {
reward-deterministic
```

};

types {
computer : object;
};

pvariables {

REBOOT-EXP : { non-fluent, real, default = 1.5 };
REBOOT-PENALTY : { non-fluent, real, default = 0.75 };
CONNECTED(computer, computer) : { non-fluent, bool, default = false };
MAX-AGE : { non-fluent, int, default = 3 };
REBOOT-CAPACITY : { non-fluent, int, default = 2 };

// state variables
running(computer) : { state-fluent, bool, default = false };
age(computer) : { state-fluent, int, default = 0 };

// action variables
reboot(computer) : { action-fluent, bool, default = false };
};

cpfs {

running'(?x) =
if (reboot(?x)) then true
else if (running(?x)) then
    if (age(?x) ≥ MAX-AGE - 1) then false
    else if ((age(?x) * [1.0 - [sum_{?y : computer} (CONNECTED(?y,?x) ∧ running(?y))] / [1 + sum_{?y : computer} CONNECTED(?y,?x)]]) ≥ REBOOT-EXP) then false

```
    else true
else false;

age'(?x) = if (reboot(?x) | running(?x)) then 0
else age(?x) + 1;
};

reward = sum_{?x : computer} [-1.0*reboot(?x))];

state-action-constraints {

// constraint 1: computers must be running
forall_{?x : computer} [running(?x)];

// constraint 2: capacity constraint
sum_{?x : computer} [reboot(?x)] ≤ REBOOT-CAPACITY

};

}

non-fluents nf_sysadmin_4 {
domain = sysadmin;
objects {
computer : {c1,c2,c3,c4};
};
non-fluents {
CONNECTED(c1,c2);
CONNECTED(c2,c1);
CONNECTED(c1,c4);
CONNECTED(c4,c1);
```

```
CONNECTED(c3,c4);
CONNECTED(c4,c3);
};
}

instance sysadmin_4 {
domain = sysadmin;
non-fluents = nf_sysadmin_4;
init-state {
running(c1);
running(c2);
running(c3);
running(c4);
};
goal-state {
running(c1);
running(c2);
running(c3);
running(c4);
};
max-nondef-actions = 4;
horizon = 2; //3,4
discount = 1.0;
}
```

Appendix C.7. System Admin,5

The domain and instance files are identical to that of System Admin,4 except the following changes in the instance file as follows.

```
non-fluents nf_sysadmin_5 {
...
```

```
objects {
computer : {c1,c2,c3,c4,c5};
};
non-fluents {
...
CONNECTED(c4,c5);
CONNECTED(c5,c4);
REBOOT-CAPACITY = 3;
};
}

instance sysadmin_5 {
domain = sysadmin;
non-fluents = nf_sysadmin_5;
init-state {
...
running(c5);
};
goal-state {
...
running(c5);
};
max-nondef-actions = 5;
...
}
```

Appendix C.8. Cellda,y

```
// Cellda - 2018
// Original Author: Buser Say

domain cellda_dom_y {
```

```
requirements = {
reward-deterministic
};

types {
dim : object;
enemy: object;
block: object;
};

pvariables {

CELL-MIN(dim) : {non-fluent, int, default = 0};
CELL-MAX(dim) : {non-fluent, int, default = 7};
BLOCK-LOC(block, dim) : {non-fluent, int, default = 0};
KEY-LOC(dim) : {non-fluent, int, default = 0};

// intermediate fluents
proposed-cellda-move(dim) : {interm-fluent, int, level = 1};
proposed-cellda-loc(dim) : {interm-fluent, int, level = 2};
proposed-enem-loc(enemy, dim) : {interm-fluent, int, level = 3};

// state variables
cellda-loc(dim) : {state-fluent, int, default = 0};
enem-loc(enemy, dim) : {state-fluent, int, default = 0};
cellda-alive : {state-fluent, bool, default = true};
has-key : {state-fluent, bool, default = false};

// action variables
move-up : {action-fluent, bool, default = false};
move-down : {action-fluent, bool, default = false};
```

```
    move-right : {action-fluent, bool, default = false};
    move-left : {action-fluent, bool, default = false};

    };

    cpfs {

    proposed-cellda-move(?d) =
    if (  cellda-alive ) then 0
    else if ( (?d == $y) ∧ move-up ) then 1
    else if ( (?d == $y) ∧ move-down ) then -1
    else if ( (?d == $x) ∧ move-right ) then 1
    else if ( (?d == $x) ∧ move-left ) then -1
    else 0;

    proposed-cellda-loc(?d) =
    if (  cellda-alive ) then cellda-loc(?d)
    else if ( exists_?b : block[ forall_{?d2: dim}[ cellda-loc(?d2) + proposed-cellda-
    move(?d2) == BLOCK-LOC(?b, ?d2)] ] )
        then cellda-loc(?d)
    else if ( exists_{?d2 : dim}[ cellda-loc(?d2) + proposed-cellda-move(?d2) >
    CELL-MAX(?d2) | cellda-loc(?d2) + proposed-cellda-move(?d2) < CELL-MIN(?d2)]
    )
        then cellda-loc(?d)
    else cellda-loc(?d) + proposed-cellda-move(?d);

    proposed-enem-loc(?e, ?d) =
    if (  cellda-alive ) then enem-loc(?e, ?d)
    else if ( (?d == $y) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) > 0 )
        then enem-loc(?e, ?d) - 1
    else if ( (?d == $y) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) < 0 )
```

then enem-loc(?e, ?d) + 1
else if ((?d == $x) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) > 0 ∧ enem-loc(?e, $y) - proposed-cellda-loc($y) == 0)
then enem-loc(?e, ?d) - 1
else if ((?d == $x) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) < 0 ∧ enem-loc(?e, $y) - proposed-cellda-loc($y) == 0)
then enem-loc(?e, ?d) + 1
else enem-loc(?e, ?d);

enem-loc'(?e, ?d) =
if (cellda-alive) then enem-loc(?e, ?d)
else if (exists_{?b : block}[forall_{?d2: dim}[proposed-enem-loc(?e, ?d2) == BLOCK-LOC(?b, ?d2)]])
then enem-loc(?e, ?d)
else if (exists_{?e2 : enemy}[(?e = ?e2) ∧ forall_{?d2: dim}[proposed-enem-loc(?e, ?d2) == proposed-enem-loc(?e2, ?d2)]])
then enem-loc(?e, ?d)
else proposed-enem-loc(?e, ?d);

cellda-alive' =
if (cellda-alive) then false
else if (exists_{?e : enemy}[forall_{?d: dim}[enem-loc(?e, ?d) == proposed-cellda-loc(?d) | enem-loc(?e, ?d) == cellda-loc(?d)]])
then false
else true;

cellda-loc'(?d) = proposed-cellda-loc(?d);

has-key' =
if (has-key) then true
else if (forall_{?d: dim}[proposed-cellda-loc(?d) == KEY-LOC(?d)])

```
    then true
else false;

};

// reward: minimize the number of actions
reward = -1*(move-up + move-down + move-right + move-left);

state-action-constraints {

// constraint 1: cell boundaries
forall_{?d : dim} [cellda-loc(?d) ≤ CELL-MAX(?d)];
forall_{?d : dim} [cellda-loc(?d) ≥ CELL-MIN(?d)];

// constraint 2: mutual exclusion of actions
move-up + move-down + move-right + move-left ≤ 1;

};

}

non-fluents nf_cellda_y {
domain = cellda_dom_y;
objects {
dim : {x, y};
enemy : {e1};
block : {b1, b2};
};
non-fluents {
CELL-MIN(x) = 0;
CELL-MAX(x) = 3;
```

```
CELL-MIN(y) = 0;
CELL-MAX(y) = 3;
BLOCK-LOC(b1, x) = 1;
BLOCK-LOC(b1, y) = 1;
BLOCK-LOC(b2, x) = 1;
BLOCK-LOC(b2, y) = 2;
KEY-LOC(x) = 1;
KEY-LOC(y) = 3;
};
}

instance cellda_y {
domain = cellda_dom_y;
non-fluents = nf_cellda_y;
init-state {
cellda-loc(x) = 0;
cellda-loc(y) = 0;
cellda-alive;
enem-loc(e1, x) = 3;
enem-loc(e1, y) = 0;
};
goal-state {
cellda-loc(x) = 3;
cellda-loc(y) = 3;
cellda-alive;
has-key;
};
max-nondef-actions = 1;
horizon = 8; //9,10
discount = 1.0;
}
```

Appendix C.9. Cellda,x

The domain and instance files are identical to that of Cellda,y except the following changes in the domain and instance files as follows.

```
domain cellda_dom_x {
...
proposed-enem-loc(?e, ?d) = if (  cellda-alive ) then enem-loc(?e, ?d)
else if ( (?d == $x) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) > 0 )
    then enem-loc(?e, ?d) - 1
else if ( (?d == $x) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) < 0 )
    then enem-loc(?e, ?d) + 1
else if ( (?d == $y) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) > 0 ∧ enem-
loc(?e, $x) - proposed-cellda-loc($x) == 0)
    then enem-loc(?e, ?d) - 1
else if ( (?d == $y) ∧ enem-loc(?e, ?d) - proposed-cellda-loc(?d) < 0 ∧ enem-
loc(?e, $x) - proposed-cellda-loc($x) == 0)
    then enem-loc(?e, ?d) + 1
else enem-loc(?e, ?d);
...
}

non-fluents nf_cellda_x {
domain = cellda_dom_x;
...
}
instance cellda_x {
domain = cellda_dom_x;
non-fluents = nf_cellda_x;
...
goal-state {
cellda-loc(x) = 3;
```

```
    cellda-loc(y) = 0;
    cellda-alive;
    has-key;
    };
    ...
    horizon = 10; //11,12
    ...
    }
```

Appendix D. Computational Results

Table D.4: Computational results including the runtimes and the total number of generalized landmark constraints generated for both FD-SAT-Plan+ and FD-BLP-Plan+ over all 27 instances within 1 hour time limit.

Non-Incremental Runtimes			Incremental Runtimes		No. of Generalized Landmarks	
Instances	FD-SAT-Plan+	FD-BLP-Plan+	FD-SAT-Plan+	FD-BLP-Plan+	FD-SAT-Plan+	FD-BLP-Plan+
Nav,3x3,4	3.15	1.41	3.15	1.41	0	0
Nav,3x3,5	5.55	3.78	5.55	3.78	0	0
Nav,3x3,6	9.19	17.82	9.19	17.82	0	0
Nav,4x4,5	82.99	107.59	82.99	107.59	0	0
Nav,4x4,6	190.77	159.42	190.77	159.42	0	0
Nav,4x4,7	303.05	455.32	303.05	455.32	0	0
Nav,5x5,8	1275.36	3600<,no sol.	1275.36	3600<	0	0
Nav,5x5,9	753.27	3600<,no sol.	753.27	3600<	0	0
Nav,5x5,10	2138.62	3600<,no sol.	2138.62	3600<	0	0
Inv,2,5	26.92	0.37	26.92	0.37	0	0
Inv,2,6	33.25	0.45	33.25	0.45	0	0
Inv,2,7	40.15	0.56	40.15	0.56	0	0
Inv,4,6	63.18	0.51	63.18	0.51	0	0
Inv,4,7	79.19	0.59	79.19	0.59	0	0
Inv,4,8	86.57	0.74	170.56	1.49	1	1
Sys,4,2	24.19	37.63	24.19	37.63	0	0
Sys,4,3	619.57	3600<,100%	3600<	3600<	6≤	n/a
Sys,4,4	1561.78	3600<,no sol.	3600<	3600<	3≤	n/a
Sys,5,2	358.53	3600<,no sol.	358.53	3600<	0	n/a
Sys,5,3	3600<,75%	3600<,no sol.	3600<	3600<	n/a	n/a
Sys,5,4	3600<,100%	3600<,no sol.	3600<	3600<	n/a	n/a
Cellda,x,10	197.16	106.93	592.44	405.52	2	2
Cellda,x,11	219.72	403.24	835.36	1539.12	2	3
Cellda,x,12	522.15	527.56	522.15	527.56	0	0
Cellda,y,8	144.95	89.64	144.95	89.64	0	0
Cellda,y,9	404.39	40.9	404.39	40.9	0	0
Cellda,y,10	575.78	544.45	575.78	544.45	0	0
Coverage	27/27	20/27	23/27	19/27		
Opt. Proved	25/27	19/27	23/27	19/27		

References

[1] B. Say, S. Sanner, Planning in factored state and action spaces with learned binarized neural network transition models, in: 27th IJCAI, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 4815–4821.

[2] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: 25th NIPS, 2012, pp. 1097–1105.
URL http://dl.acm.org/citation.cfm?id=2999134.2999257

[3] L. Deng, G. E. Hinton, B. 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, 2013, pp. 8599–8603.

[4] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, JMLR 12 (2011) 2493–2537.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature (2016) 484–503.
URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis, Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
URL https://arxiv.org/abs/1712.01815

[7] B. Say, G. Wu, Y. Q. Zhou, S. Sanner, Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming, in: 26th IJCAI, 2017, pp. 750–756.

[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: 29th NIPS, Curran Associates, Inc., 2016, pp. 4107–4115.
URL http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf

[9] C. Boutilier, T. Dean, S. Hanks, Decision-theoretic planning: Structural assumptions and computational leverage, JAIR 11 (1) (1999) 1–94.
URL http://dl.acm.org/citation.cfm?id=3013545.3013546

[10] A. Boudane, S. Jabbour, B. Raddaoui, L. Sais, Efficient sat-based encodings of conditional cardinality constraints, 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning 57 (2018) 181–195.

[11] Q. Yang, K. Wu, Y. Jiang, Learning action models from plan examples using weighted max-sat, AIJ 171 (2) (2007) 107–143.

[12] E. Amir, A. Chang, Learning partially observable deterministic action models, JAIR 33 (2008) 349–402.

[13] M. Helmert, The fast downward planning system, JAIR 26 (1) (2006) 191–246.
URL http://dl.acm.org/citation.cfm?id=1622559.1622565

[14] S. Richter, M. Westphal, The lama planner: Guiding cost-based anytime planning with landmarks, JAIR 39 (1) (2010) 127–177.
URL http://dl.acm.org/citation.cfm?id=1946417.1946420

[15] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: ECML, 2006, pp. 282–293.

[16] T. Keller, M. Helmert, Trial-based heuristic tree search for finite horizon MDPs, in: 23rd ICAPS, 2013, pp. 135–143.
URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6026

[17] T. O. Davies, A. R. Pearce, P. J. Stuckey, N. Lipovetzky, Sequencing operator counts, in: 25th ICAPS, 2015, pp. 61–69.
URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS15/paper/view/10618

[18] Nintendo, The legend of zelda (1986).

[19] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: 27th ICML, 2010, pp. 807–814.
URL http://www.icml2010.org/papers/432.pdf

[20] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: 32nd ICML, 2015, pp. 448–456.
URL http://dl.acm.org/citation.cfm?id=3045118.3045167

[21] M. Davis, H. Putnam, A computing procedure for quantification theory, Journal of the ACM 7 (3) (1960) 201–215.

[22] J. Davies, F. Bacchus, Solving MAXSAT by solving a sequence of simpler SAT instances, in: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, 2013.

[23] R. Asin, R. Nieuwenhuis, Cardinality networks and their applications and oliveras, albert and rodriguez-carbonell, enric, in: International Conference on Theory and Applications of Satisfiability Testing, 2009, pp. 167–180.

[24] C. Sinz, Towards an Optimal CNF Encoding of Boolean Cardinality Constraints, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 827–831.

[25] O. Bailleux, Y. Boufkhad, O. Roussel, A translation of pseudo boolean constraints to SAT, Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 191–200.

[26] S. Jabbour, L. Sas, Y. Salhi, A pigeon-hole based encoding of cardinality constraints, in: ISAIM, 2014.

[27] IBM, IBM ILOG CPLEX Optimization Studio CPLEX User's Manual (2017).

[28] I. Abío, P. J. Stuckey, Encoding linear constraints into sat, in: Principles and Practice of Constraint Programming, Springer Int Publishing, 2014, pp. 75–91.

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

[30] N. En, N. Srensson, Translating pseudo-boolean constraints into sat, Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 1–26.

[31] S. Sanner, S. Yoon, International probabilistic planning competition (2011).

[32] T. Mann, S. Mannor, Scaling up approximate value iteration with options: Better policies with fewer iterations, in: 21st ICML, Vol. 1, 2014.

[33] C. Guestrin, D. Koller, R. Parr, Max-norm projections for factored MDPs, in: 17th IJCAI, 2001, pp. 673–680.