A Unified Framework for Planning with Learned Neural Network Transition Models

Buser Say
Monash University, Melbourne, Victoria, Australia
buser.say@monash.edu

Abstract

Automated planning with neural network transition models is a two stage approach to solving planning problems with unknown transition models. The first stage of the approach learns the unknown transition model from data as a neural network model, and the second stage of the approach compiles the learned model to either a Mixed-Integer Linear Programming (MILP) model or a Recurrent Neural Network (RNN) model, and optimize it using an off-the-shelf solver. The previous studies have shown that both models have their advantages and disadvantages. Namely, the MILP model can be solved optimally using a branch-and-bound algorithm but has been experimentally shown not to scale well for neural networks with multiple hidden layers. In contrast, the RNN model can be solved effectively using a gradient descent algorithm but can only work under very restrictive assumptions. In this paper, we focus on improving the effectiveness of solving the second stage of the approach by introducing (i) a novel Lagrangian RNN architecture that can model the previously ignored components of the planning problem as Lagrangian functions, and (ii) a novel framework that unifies the MILP and the Lagrangian RNN models such that the weakness of one model is complemented by the strength of the other. Experimentally, we show that our unifying framework significantly outperforms the standalone MILP model by solving 80% more problem instances, and showcase the ability of our unifying framework to find high quality solutions to challenging automated planning problems with unknown transition models.

Introduction

Planning is the reasoning side of acting in Artificial Intelligence (Nau, Ghallab, and Traverso 2004). It automates the selection and ordering of actions to reach desired states of the world as best as possible. An automated planning problem represents the dynamics of the world using a model, which can either be manually encoded (Kautz and Selman 1992; Hoffmann and Nebel 2001; Helmert 2006; Pommerening et al. 2014; Davies et al. 2015), or learned from data (Shen and Simon 1989; Gil 1992; Bennett and DeJong 1996; Benson 1997). In this paper, we focus on the latter where we assume the transition model that governs the state evolution is not known but it can be accurately learned from data (Say et al. 2017; Say and Sanner 2018b; Say, Sanner, and Thiébaux 2019; Say and Sanner 2020; Say et al. 2020; Wu, Say, and Sanner 2020; Say 2020).

Automated planning with neural network transition models is a two stage approach to solving planning problems with unknown state transition models (Say et al. 2017). Assuming the availability of data, the first stage of the approach learns the unknown transition model as a neural network model, and the second stage of the approach represents the learned planning problem either as a Mixed-Integer Linear Programming (MILP) model (Say et al. 2017) or a Recurrent Neural Network (RNN) model (Wu, Say, and Sanner 2020), and optimize it using an off-the-shelf solver. As summarized in Table 1, it has been shown that both models have their advantages and disadvantages (Wu, Say, and Sanner 2020). Specifically, the MILP model can be optimally solved using a complete and sound branch-and-bound algorithm, but has been shown not to scale well for neural networks with multiple hidden layers. In contrast, the RNN model can be solved effectively using a gradient descent algorithm but works under very restrictive assumptions which include the ommission of the important parts of a planning problem and the availability of a good starting point.

Model	(+) Advantages and (-) Disadvantages
MILP	+ Can be provably solved to optimality. - Not scalable for deep neural networks.
RNN	+ Scalable for deep neural networks. - Makes restrictive assumptions. - Requires a good starting point.

Table 1: Summary of the MILP (Say et al. 2017) and the RNN (Wu, Say, and Sanner 2020) models.

In this paper, we focus on improving the effectiveness of solving the second stage of the learning and planning approach with the introduction of (i) a novel Lagrangian RNN architecture which can model the previously ignored components of the planning problem as Lagrangian functions (Kuhn and Tucker 1951), and (ii) a novel framework which unifies the MILP and the Lagrangian RNN models such that the weaknesses of one model is complemented by the strengths of the other. Specifically, our unified framework is implemented as an algorithm which solves the MILP

model using branch-and-bound and calls the Lagrangian RNN model as a primal heuristic (i.e., a heuristic for finding feasible solutions). We conduct two sets of computational experiments to test the effectiveness of our contributions. First, we test the effectiveness of our unified framework to solve the learned planning problem. Here, we show that our unifying framework significantly outperforms the standalone MILP model by solving 80% more learned planning problem instances, and can significantly improve its solution quality (i.e., upto 40%) over time. Then, we test the effectiveness of our unified framework to solve the planning problem with an unknown transition model. Here, we first verify the plans found in the first set of experiments using a domain simulator and show that around 95% of those solutions are also plans for the respective planning problem instances. Finally, we test the quality of those verified plans using the duality information obtained from solving the Mixed-Integer Nonlinear Programming model of the respective planning problem instances. We finalize our experiments by showing that our unifying framework can find high quality plans (i.e., with at most 4% duality gap) to the automated planning problem instances without having direct access to their transition models. We conclude our paper with a discussion of our experimental results in relation to the literature with the purpose of starting new areas for future work.

Planning with Learned Neural Network Transition Models

We begin by presenting the definition of the learned automated planning problem and the two previous models for solving the learned automated planning problem.

Problem Definition

A *fixed-horizon learned deterministic automated planning problem* (Say et al. 2017) is a tuple $\tilde{\Pi} = \langle S, A, C, \tilde{T}, V, G, R, H \rangle$, where $S = \{s_1, \ldots, s_n\}$ and $A = \{a_1, \ldots, a_m\}$ are sets of state and action variables with domains $D_{s_1}, \ldots, D_{s_n}$ and $D_{a_1}, \ldots, D_{a_m}$ for positive integers n, m, $C : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow \{true, false\}$ is the global constraint function, $\tilde{T} : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow D_{s_1} \times \cdots \times D_{s_n}$ denotes the learned state transition function, and $R : D_{s_1} \times \cdots \times D_{s_n} \times D_{a_1} \times \cdots \times D_{a_m} \rightarrow \mathbb{R}$ is the reward function. Moreover, V is a tuple of constants $\langle V_1, \ldots, V_n \rangle \in D_{s_1} \times \cdots \times D_{s_n}$ that denotes the initial values of all state variables, $G : D_{s_1} \times \cdots \times D_{s_n} \rightarrow \{true, false\}$ is the goal state function, and $H \in \mathbb{Z}^+$ is the planning horizon.

A *solution* to $\tilde{\Pi}$ is a tuple of values $\bar{A}^t = \langle \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle \in D_{a_1} \times \cdots \times D_{a_m}$ for all action variables A over time steps $t \in \{1, \ldots, H\}$ such that $\tilde{T}(\langle \bar{s}_1^t, \ldots, \bar{s}_n^t, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle) = \langle \bar{s}_1^{t+1}, \ldots, \bar{s}_n^{t+1} \rangle$ and $C(\langle \bar{s}_1^t, \ldots, \bar{s}_n^t, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle) = true$ for time steps $t \in \{1, \ldots, H\}$, $V_i = \bar{s}_i^1$ for all $s_i \in S$ and $G(\langle \bar{s}_1^{H+1}, \ldots, \bar{s}_n^{H+1} \rangle) = true$. Similarly, an *optimal* solution to $\tilde{\Pi}$ is a solution such that the total reward $\sum_{t=1}^{H} R(\langle \bar{s}_1^{t+1}, \ldots, \bar{s}_n^{t+1}, \bar{a}_1^t, \ldots, \bar{a}_m^t \rangle)$ is maximized.

It is assumed that the functions C, G, R and $\tilde{T}$ are known, that C, G can be equivalently represented by a finite set of piecewise linear constraints I_C and I_G, that R is a piecewise linear expression and that $\tilde{T}$ is a densely-connected (Huang et al. 2017a) learned neural network with Rectified Linear Units (ReLUs) (Nair and Hinton 2010) and linear units. Next, we review two models; (i) a Mixed-Integer Linear Programming (MILP) model (Say et al. 2017) and (ii) a Recurrent Neural Network (RNN) model (Wu, Say, and Sanner 2020), for solving the learned planning problem $\tilde{\Pi}$.

Mixed-Integer Linear Programming Model for the Learned Planning Problem

In this section, we present the Mixed-Integer Linear Programming (MILP) model (Say et al. 2017) to solve the learned planning problem $\tilde{\Pi}$.

Parameters. The MILP model uses the following parameters:

- L is the number of layers of the neural network $\tilde{T}$.
- W_l is the width of layer $l \in [1, L]$ where $[1, L]$ denotes the set $\{1, \ldots, L\}$.
- $J(l) = \{u_{1,l}, \ldots, u_{W_l,l}\}$ is the set of neurons in layer $l \in [1, L]$. For notational simplicity, we assume neurons $u_{1,1}, \ldots, u_{n,1} \in J(1)$ represent the state variables S and neurons $u_{n+1,1}, \ldots, u_{n+m,1} \in J(1)$ represent the action variables A. Similarly, we assume $u_{1,L}, \ldots, u_{n,L} \in J(L)$ represent the state variables S.
- $w_{i,j,k,l}$ is the value of the learned weight between neurons $u_{i,k} \in J(k)$ and $u_{j,l} \in J(l)$ in layers $k \in [1, L-1]$ and $l \in [2, L]$ where $k < l$.
- $B(j, l)$ is the bias of neuron $u_{j,l} \in J(l)$ in layer $l \in [2, L]$.
- M is a large constant used in the linearization of indicator constraints (i.e., big-M constraints).

Decision Variables. The MILP model uses the following decision variables:

- $X_{i,t} \in D_{a_i}$ encodes the value of action variable $a_i \in A$ at time step $t \in [1, H]$.
- $Y_{i,t} \in D_{s_i}$ encodes the value of state variable $s_i \in S$ at time step $t \in [1, H+1]$.
- $P_{i,l,t} \in \mathbb{R}$ encodes the value of the output of neuron $u_{i,l} \in J(l)$ in layer $l \in [1, L]$ at time step $t \in [1, H]$.
- $Z_{i,l,t} \in \{0, 1\}$ encodes whether neuron $u_{i,l} \in J(l)$ in layer $l \in [2, L-1]$ is activated (i.e., when its input is positive) at time step $t \in [1, H]$ or not.

Constraints. The MILP model has the following constraints:

$$Y_{i,1} = V_i \; \forall_{s_i \in S} \tag{1}$$
$$G(\langle Y_{1,H+1}, \ldots, Y_{n,H+1} \rangle) = true \tag{2}$$
$$C(\langle Y_{1,t}, \ldots, Y_{n,t}, X_{1,t}, \ldots, X_{m,t} \rangle) = true \; \forall_{t \in [1,H]} \tag{3}$$
$$Y_{i,t} = P_{i,1,t} \; \forall_{s_i \in S, t \in [1,H]} \tag{4}$$
$$X_{i,t} = P_{i+n,1,t} \; \forall_{a_i \in A, t \in [1,H]} \tag{5}$$
$$Y_{i,t+1} = P_{i,L,t} \; \forall_{s_i \in S, t \in [1,H]} \tag{6}$$
$$0 \leq P_{j,l,t} \; \forall_{u_{j,l} \in J(l), l \in [2,L-1], [1,H]} \tag{7}$$
$$In(j,l,t) \leq P_{j,l,t} \; \forall_{u_{j,l} \in J(l), l \in [2,L-1], t \in [1,H]} \tag{8}$$
$$In(j,l,t) \leq M Z_{j,l,t} \; \forall_{u_{j,l} \in J(l), l \in [2,L-1], t \in [1,H]} \tag{9}$$
$$In(j,l,t) \geq -M\big(1 - Z_{j,l,t}\big) + P_{j,l,t} \; \forall_{u_{j,l} \in J(l), l \in [2,L-1], t \in [1,H]} \tag{10}$$

where the input expression $In(j,l,t)$ for neuron $u_{j,l} \in J(l)$ in layer $l \in [2,L]$ at time step $t \in [1,H]$ is equal to $\sum_{u_{i,k} \in J(k), k \in [1,l-1]} w_{i,j,k,l} \cdot P_{i,k,t} + B(j,l)$. In the above MILP model, constraints (1) set the initial value of every state variable. Constraints (2)–(3) encode the global constraint function C and the goal state function G as a set of piecewise linear constraints. Constraints (4)–(6) map the input and output layers of the learned neural network $\tilde{T}$ to the corresponding state and action variables. Note that constraints (6) assume that the predicted state variables have real-valued domains (i.e., $D_{s_i} \subseteq \mathbb{R}$ for all $s_i \in S$). Finally, constraints (7)–(10) model the activation of each ReLU in the learned neural network.

Objective Function. The MILP model has the objective function

$$\max \sum_{t=1}^{H} R(\langle Y_{1,t+1}, \ldots, Y_{n,t+1}, X_{1,t}, \ldots, X_{m,t} \rangle) \tag{11}$$

which maximizes the total reward accumulated over time steps $t \in [1,H]$.

Optimization. The MILP model is solved using the branch-and-bound algorithm to find an optimal solution to the learned automated planning problem $\tilde{\Pi}$.

Experimentally, it has been shown that solving the MILP model using the branch-and-bound algorithm does not scale well with the increasing number of hidden layers in the learned deep neural network $\tilde{T}$ (i.e., for $L \geq 4$). As an alternative, a Recurrent Neural Network (RNN) model (Wu, Say, and Sanner 2020) has been introduced to approximately solve the learned automated planning problem, which we describe next.

Recurrent Neural Network Model for the Learned Planning Problem

In this section, we present the Recurrent Neural Network (RNN) model (Wu, Say, and Sanner 2017, 2020) to approximately solve the learned planning problem $\tilde{\Pi}$.

Parameters. The RNN model uses the following parameters in addition to the set of parameters that are previously described for the MILP model:

- J is the total number of optimization instances.
- α is the optimization rate.

Decision Variables. The RNN model uses the same set of decision variables as the MILP model previously described. We extend the notation $Y_{i,t}$ and $X_{i,t}$ to $Y^j_{i,t}$ and $X^j_{i,t}$ for each optimization instance $j \in [1,J]$.

Recurrent Neural Network Architecture. The RNN has the following architecture:

- Each optimization instance $j \in [1,J]$ at time step $t \in [1,H]$ is represented by an RNN cell with input and output vectors $(Y^j_{1,t}, \ldots, X^j_{m,t})$ and $(Y^j_{1,t+1}, \ldots, Y^j_{n,t+1}, r^j_t)$, respectively, where the reward output r^j_t is equal to $R((Y^j_{1,t+1}, \ldots, X^j_{m,t}))$.
- Given the initial values V_i of state variables $s_i \in S$, the architecture of the learned neural network $\tilde{T}$ is used to build the directed acyclic graph such that $\tilde{T}((Y^j_{1,t}, \ldots, X^j_{m,t})) = (Y^j_{1,t+1}, \ldots, Y^j_{n,t+1})$ for all optimization instances $j \in [1,J]$ and time steps $t \in [1,H]$.
- The total loss function Q is defined over all optimization instances $j \in [1,J]$ such that $Q = -\sum_{j \in [1,J]} (Q^j)^2$ where the loss function Q^j of instance j is defined over all time steps $t \in [1,H]$ such that $Q^j = \sum_{t \in [1,H]} r^j_t$.

Optimization. The total loss function Q is minimized given some initial values of all action variables $a_i \in A$, optimization instances[1] $j \in [1,J]$ and time steps $t \in [1,H]$ using the gradient descent algorithm such that:

$$\boldsymbol{X} \leftarrow \boldsymbol{X} - \alpha \frac{\partial Q}{\partial \boldsymbol{X}} \tag{12}$$

where notation $\boldsymbol{X}$ denotes the vector of all action decision variables $\boldsymbol{X} = (X^1_{1,1}, \ldots, X^J_{m,H})$.

Additional Assumptions. As highlighted in Table 1, the previously introduced RNN model further assumes that the global constraint function C can be encoded as a set of linear bound constraints (i.e., box constraints) on each action variable $a_i \in A$. Moreover, it assumes the goal state function G is satisfied for all values of state variables, and the knowledge on some initial values of all action variables.

Next, we present our first contribution which extends the previous RNN model to relax the restrictive assumptions placed on functions C and G.

[1]We run a batch of multiple action optimizations simultaneously where each is referred to as an optimization instance.

Lagrangian Recurrent Neural Network Model for the Learned Planning Problem

In this section, we present a novel Lagrangian RNN model to solve the learned planning problem $\tilde{\Pi}$. The Lagrangian RNN model introduced in this section extends the previous RNN model (Wu, Say, and Sanner 2020) with real-valued Lagrangian functions to encode the violation of constraints that represent functions C and G.[2]

Parameters. The Lagrangian RNN model uses the following parameters in addition to the set of parameters that are previously described for the MILP model and the previous RNN model:

- λ_C is the Lagrangian multiplier for the global constraint function C.
- λ_G is the Lagrangian multiplier for the goal state function G.

Decision Variables. The Lagrangian RNN model uses the same set of decision variables as the RNN model previously described.

Lagrangian Recurrent Neural Network Architecture. The Lagrangian RNN has the following architecture:

- Each optimization instance $j \in [1, J]$ at time step $t \in [1, H-1]$ is represented by an RNN cell with input and output vectors $(Y^j_{1,t}, \ldots, X^j_{m,t})$ and $(Y^j_{1,t+1}, \ldots, Y^j_{n,t+1}, r^j_t, c^j_t)$, respectively, where the Lagrangian output c^j_t measures the violation of the set of piecewise linear constraints that represent function C (i.e., when function C evaluates to false). Similarly, each optimization instance $j \in [1, J]$ at time step $t = H$ is represented by an RNN cell with input and output vectors $(Y^j_{1,t}, \ldots, X^j_{m,t})$ and $(Y^j_{1,t+1}, \ldots, Y^j_{n,t+1}, r^j_t, c^j_t, g^j)$, respectively, where the Lagrangian output g^j measures the violation of the set of piecewise linear constraints that represent function G.
- The directed acyclic graph is built similarly to the previous RNN model using the learned neural network $\tilde{T}$.
- The total loss function Q is defined similarly to the previous RNN model except the loss function Q^j of optimization instance j is defined over all time steps $t \in [1, H]$ such that $Q^j = \sum_{t \in [1,H]}(r^j_t - c^j_t) - g^j$.

Lagrangian Outputs. The Lagrangian outputs c^j_t and g^j penalize the total violation of the set of constraints that represent functions C and G, respectively. Given each constraint $i \in I_C$ is piecewise linear and is in the form of: $\sum_{k=1}^{n} e_{i,k} \cdot Y^j_{k,t} + \sum_{k=1+n}^{n+m} e_{i,k} \cdot X^j_{k,t} \leq f_i$ for some known constants $e_{i,1}, \ldots, e_{i,n+m}, f_i \in \mathbb{R}$, we define the Lagrangian output c^j_t as:

$$\lambda_C \cdot \sum_{i \in I_C} \max\Big(\sum_{k=1}^{n} e_{i,k} \cdot Y^j_{k,t} + \sum_{k=1+n}^{n+m} e_{i,k} \cdot X^j_{k,t} - f_i, 0\Big) \tag{13}$$

for some Lagrangian multiplier $0 \leq \lambda_C \in \mathbb{R}$. Similarly, given each constraint $i \in I_G$ is piecewise linear and is in the form of: $\sum_{k=1}^{n} e_{i,k} \cdot Y^j_{k,H+1} \leq f_i$ for some known constants $e_{i,1}, \ldots, e_{i,n}, f_i \in \mathbb{R}$, we define the Lagrangian output g^j as:

$$\lambda_G \cdot \sum_{i \in I_G} \max\Big(\sum_{k=1}^{n} e_{i,k} \cdot Y^j_{k,H+1} - f_i, 0\Big) \tag{14}$$

for some Lagrangian multiplier $0 \leq \lambda_G \in \mathbb{R}$. Finally, we encode all $\max$ operators as ReLUs in the RNN cell.

Optimization. The modified total loss function Q is minimized using gradient descent similar to the optimization of the previous RNN model.

Additional Assumptions. The Lagrangian RNN model assumes the values of Lagrangian multipliers λ_C and λ_G to be known a priori. For sufficiently large values of λ_C and λ_G, the Lagrangian RNN model can find feasible solutions to the learned planning problem $\tilde{\Pi}$. However, setting these values to arbitrarily large constants can result in both theoretical (e.g., suboptimality of the solution found) and practical (e.g., exploding gradient) problems.

Next, we will describe how to obtain the values of the Lagrangian multipliers and the action variables to solve the Lagrangian RNN model with gradient descent, and use the solution of the Lagrangian RNN model to increase the effectiveness of branch-and-bound to solve the MILP model. To this end, we present our second contribution which combines the strengths of the MILP model and the Lagrangian RNN model under a unified framework to effectively solve the learned planning problem $\tilde{\Pi}$.

Unified Framework for the Learned Planning Problem

In this section, we present a unified framework that combines the MILP model and the Lagrangian RNN model to effectively solve the learned planning problem $\tilde{\Pi}$. The unified framework solves the MILP model using the branch-and-bound algorithm and calls the Lagrangian RNN model as a primal heuristic. The gradient descent algorithm is (i) initialized with the values of action decision variables $X_{i,t}$ that are obtained from solving the linear relaxation of the MILP model, and (ii) the Lagrangian multipliers λ_C and λ_G are updated based on a measure of solution feasibility. Next, we present Algorithm 1 that implements the unified framework for solving the learned planning problem.

[2] Cf. previous work (Song 2019) has used Boolean-valued functions with user-specified Lagrangian multipliers to approximate function C and has shown limited success for model-based discrete-time planning (Wu, Say, and Sanner 2017).

Algorithm 1 Unified Algorithm

1: Optimization rate: $\alpha = h_{\alpha}^{initial}(\tilde{\Pi})$
2: Lagrangian multipliers: $\lambda_C = h_{\lambda_C}^{initial}(\tilde{\Pi})$, $\lambda_G = h_{\lambda_G}^{initial}(\tilde{\Pi})$
3: Callback frequency: $\beta \in \mathbb{Z}^+$
4: Incumbent: $\tilde{\pi} = ()$
5: Branch-and-bound callback: B&B$(\beta, \tilde{\pi}) \rightarrow \langle \bar{\boldsymbol{X}}, \tilde{\pi} \rangle$
where $\bar{\boldsymbol{X}} \in \{(), \ldots, (\bar{X}^1_{1,1}, \ldots, \bar{X}^{\beta}_{m,H})\}$
6: Gradient descent: SG$(\alpha, \lambda_C, \lambda_G, \bar{\boldsymbol{X}}) \rightarrow \bar{\boldsymbol{Z}}$
where $\bar{\boldsymbol{Z}} = (\bar{Z}_{1,2,1}, \ldots, \bar{Z}_{W_{L-1},L-1,H})$
7: Simplex: LP$(\bar{\boldsymbol{Z}}) \rightarrow \bar{\boldsymbol{X}}_{LP}$
where $\bar{\boldsymbol{X}}_{LP} \in \{(), (\bar{X}_{1,1}, \ldots, \bar{X}_{m,H})\}$
8: $\langle \bar{\boldsymbol{X}}, \tilde{\pi} \rangle \leftarrow$ B&B$(\beta, \tilde{\pi})$
9: **while** $\bar{\boldsymbol{X}} \neq ()$ **do**
10: $\quad \bar{\boldsymbol{Z}} \leftarrow$ SG$(\alpha, \lambda_C, \lambda_G, \bar{\boldsymbol{X}})$
11: $\quad \bar{\boldsymbol{X}}_{LP} \leftarrow$ LP$(\bar{\boldsymbol{Z}})$
12: $\quad$ **if** $\bar{\boldsymbol{X}}_{LP} \neq ()$ and $\bar{\boldsymbol{X}}_{LP}$ has greater reward than $\tilde{\pi}$ **then**
13: $\quad\quad \tilde{\pi} \leftarrow \bar{\boldsymbol{X}}_{LP}$
14: $\quad \beta \leftarrow h_{\beta}^{update}(\tilde{\pi})$
15: $\quad \lambda_C \leftarrow h_{\lambda_C}^{update}(\bar{\boldsymbol{X}}_{LP})$, $\lambda_G \leftarrow h_{\lambda_G}^{update}(\bar{\boldsymbol{X}}_{LP})$
16: $\quad \langle \bar{\boldsymbol{X}}, \tilde{\pi} \rangle \leftarrow$ B&B$(\beta, \tilde{\pi})$
17: **return** $\tilde{\pi}$

Parameters and Definitions. Lines 1-7 initialize the parameters and provides high-level algorithmic definitions as follows. Lines 1-2 set the values of the optimization rate α and Lagrangian multipliers λ_C, λ_G according to the parameters of the learned automated planning problem $\tilde{\Pi}$. Line 3 initializes the value of the callback frequency β, which determines how frequently the gradient descent algorithm should be called as a primal heuristic. Line 4 initializes the incumbent solution $\tilde{\pi}$ to an empty vector. Lines 5-7 provide high-level definitions of the algorithms that are used in Algorithm 1 in terms of their inputs and outputs. Line 5 defines the callback to the branch-and-bound algorithm B&B$(\beta, \tilde{\pi})$ which maintains (i.e., initalizes or updates) a single branch-and-bound tree throughout the execution of Algorithm 1. B&B$(\beta, \tilde{\pi})$ inputs the incumbent solution $\tilde{\pi}$ and the callback frequency β, and returns a vector of action variable values obtained from solving $j \in [0, \beta]$ number of linear relaxations and the best incumbent solution $\tilde{\pi}$ found. Line 6 defines the gradient descent algorithm SG for solving the Lagrangian RNN model previously described. Note that SG returns the value of each decision variable $Z_{i,l,t} \in \{0, 1\}$, which models the activation of ReLU $u_{i,l} \in J(l)$ in layer $l \in [2, L-1]$ at time step $t \in [1, H]$. Finally, Line 7 defines the simplex algorithm which solves a linear programming model for given values of binary decision variables $\bar{\boldsymbol{Z}}$. This linear programming model is equivalent to the MILP model such that the vector of binary decision variables $(Z_{1,2,1}, \ldots, Z_{W_{L-1},L-1,H})$ are substituted by their respective values $\bar{\boldsymbol{Z}}$.

Control Flow. Lines 8-17 describe the control flow of Algorithm 1. Line 8 runs the branch-and-bound algorithm B&B upto β number of nodes, and returns the values of action variables that are obtained from solving the linear relaxation at nodes $j \in [0, \beta]$ as well as the best incumbent solution $\tilde{\pi}$ found. Line 9 terminates Algorithm 1 only when B&B terminates (i.e., denoted by an empty vector). Line 10 runs the gradient descent algorithm SG with initial values of action variables, and returns the values of decision variables $Z_{i,l,t}$. Line 11 solves a linear relaxation of the MILP model for fixed values of ReLU activations. Lines 12-13 update the incumbent if solving the linear relaxation results in an incumbent with greater total reward. Lines 14-15 update the values of the callback frequency β, and the Lagrangian multipliers λ_C and λ_G according to the information about the incumbent solution $\tilde{\pi}$ and the solution of the linear programming model, respectively. Line 16 continues the branch-and-bound algorithm with the updated value of the callback frequency β and the best incumbent solution $\tilde{\pi}$. Line 17 returns $\tilde{\pi}$ as the optimal solution to the learned planning problem $\tilde{\Pi}$. The details of β, λ_C and λ_G will be provided in Table 3.

Before we test the experimental performance of Algorithm 1, we remark that it preserves both soundness and the completeness properties of the branch-and-bound algorithm since the incumbent solution $\tilde{\pi}$ can only be updated by (i) the branch-and-bound algorithm itself (i.e., lines 8 and 16), and (ii) the optimal solution of the simplex algorithm if it yields an incumbent with greater total reward (i.e., lines 11-13).

Experimental Results

In this section, we present results of two sets of experiments. In the first set of experiments, we test the effectiveness of using Algorithm 1 to solve the learned planning problem $\tilde{\Pi}$. We compare Algorithm 1 against the only model that is also capable of solving the learned planning problem $\tilde{\Pi}$, namely the existing MILP model (Say et al. 2017) over multiple instances of the learned planning problem $\tilde{\Pi}$. We report detailed comparative results on the solution quality of both approaches over time and show Algorithm 1 significantly outperforms the standalone MILP model. In the second set of experiments, we test the effectiveness of using Algorithm 1 to solve the planning problem $\Pi = \langle S, A, C, T, V, G, R, H \rangle$ with a known transition function T. Here, we verify the ability of Algorithm 1 to find a plan for the underlying planning problem Π using a domain simulator. Then, we compare Algorithm 1, which only has access to the learned transition function $\tilde{T}$, against the Mixed-Integer Nonlinear Programming (MINLP) model of Π, which can be solved using a spatial branch-and-bound algorithm to bounded optimality.[3] We report results on the solution quality of Algorithm 1 to solve the planning problem Π using the dual bounds obtained from the spatial branch-and-bound algorithm, and showcase the ability of Algorithm 1 to find high quality solutions to Π without having access to the transition function T.

Experimental Domains and Setup

Three challenging domains are selected from the literature to be used in both sets of experiments, namely: Navigation (Faulwasser and Findeisen 2009), Reservoir Control (Yeh 1985) and Inventory Control (Mann and Mannor

[3] We assume function T can be equivalently represented by a finite set of nonlinear constraints.

2014).[4] These domains have been modified mainly with the addition of a goal state function G to produce a total of 64 new instances of the automated planning problem Π. We provide a brief summary of these domains in Table 2.

Domain	n, m	Brief Description
Navigation	2,2	A navigation task for an agent in a two-dimensional maze with exponential function T. A and S have real domains.
Reservoir	8,8 and 10,10	Safely controlling n reservoirs with trigonometric function T. A and S have real domains.
Inventory	17,8 and 21,10	Inventory management of n products with demand cycles with trigonometric function T. A and S have integer domains.

Table 2: Summary of the domains.

We followed the methodology outlined for data generation and neural network training to select the final neural network architectures (Wu, Say, and Sanner 2020). We found that neural networks with two hidden layers (i.e., $L = 4$) and 32 width (i.e., $W_2 = W_3 = 32$) most accurately predicted the data for all three domains. All experiments were run on the CPU of a MacBookPro with 2.8 GHz Intel Core i7 16GB memory, using a single thread with one hour total time limit per problem instance. We used CPLEX 12.10 as the branch-and-bound solver, Tensorflow (Abadi et al. 2016) as the gradient descent optimizer and SCIP 7.0 (Vigerske and Gleixner 2018) as the spatial branch-and-bound solver. In Table 3, we present the initial value of parameter β and the definitions of functions $h_{\alpha}^{initial}$, $h_{\lambda_C}^{initial}$, $h_{\lambda_G}^{initial}$, h_{β}^{update}, $h_{\lambda_C}^{update}$, $h_{\lambda_G}^{update}$. Given the information presented in Table 3, we implemented Algorithm 1 using the *HeuristicCallback* interface of CPLEX.

Parameter / Function	Initialization / Definition
β	1000
$h_{\alpha}^{initial}$	0.01 times the absolute value of the largest coefficient in R.
$h_{\lambda_C}^{initial}$, $h_{\lambda_G}^{initial}$	100 times the absolute value of the largest coefficient in R.
h_{β}^{update}	**if** $\tilde{\pi} \neq ()$ **then** $2 \cdot \beta$ **else** β
$h_{\lambda_C}^{update}$, $h_{\lambda_G}^{update}$	**if** $\bar{\boldsymbol{X}}_{LP} = ()$ **then** $2 \cdot \lambda$ **else** $\lambda/2$

Table 3: Initializations and definitions for Algorithm 1.

Experiment 1: Solving Learned Planning Problems

In the first set of experiments, we compare the effectiveness of solving the learned planning problem $\tilde{\Pi}$ using Algorithm 1 against the standalone MILP model. Figure 1 visualizes the runtime behaviour of Algorithm 1 (red) and the MILP model (blue) as a function of time. Figure 1 is organized such that the subfigures on the left (i.e., subfigures 1a, 1c and 1e) represent the cumulative number of problem instances for which an incumbent solution is found over time and the subfigures on the right (i.e., subfigures 1b, 1d and 1f) represent the improvement of solution quality over time, and each row represents a unique domain.

Problem Coverage. In Figure 1, the inspection of the subfigures on the left highlights the clear benefit of using Algorithm 1 over the standalone MILP model. Overall, we observe that Algorithm 1 covers around 80% more problem instances within the time limit compared to the MILP model. Moreover we observe that for the majority (i.e., around 80%) of problem instances, Algorithm 1 finds its first incumbent solution under 500 seconds. A closer study of subfigures 1a and 1e shows that the MILP model cannot find a single solution to a problem instance in Navigation or Inventory Control, respectively. Similarly, the subfigure 1c shows that the MILP model covers around 30% of the problem instances in Reservoir Control. In contrast to the MILP model, Algorithm 1 finds incumbent solutions to all problem instances in Reservoir Control and Inventory Control, and covers more than 60% of the problem instances in Navigation.

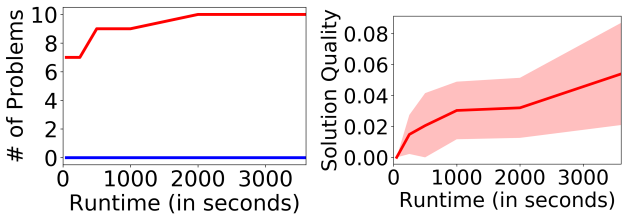
Solution Quality. In Figure 1, the inspection of the subfigures on the right highlights the ability of Algorithm 1 to improve its solution quality over time. The study of subfigures 1b and 1f shows that Algorithm 1 improves its solution quality by around 5% and 20% on average in Navigation and Inventory Control, respectively. Note that the MILP model is not represented in the same subfigures since it cannot find a single feasible solution to the respective problem instances. The study of subfigure 1d reveals that the MILP model can improve its solution quality in Reservoir Control by around 30% on average. Interestingly, Algorithm 1 shows a similar performance to the MILP model over all problem instances in Reservoir Control, which showcases the robustness of Algorithm 1 to the variations in the problem instances.

Experiment 2: Solving Planning Problems

In the second set of experiments, we test the effectiveness of solving the automated planning problem Π using Algorithm 1. We begin with the verification of plans found by Algorithm 1 using a simulator.[5] Then, we compare the solution quality of Algorithm 1 to the best dual bounds found by solving the MINLP model of Π using a spatial branch-and-bound algorithm, and summarize both results in Table 4.

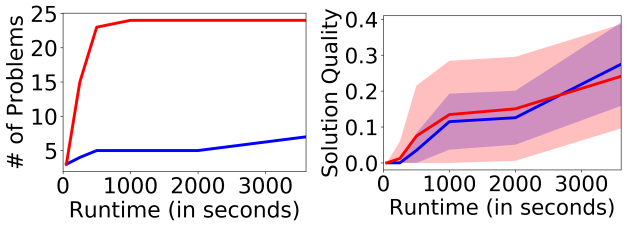
[4] In Inventory Control, we predicted the integer domains of state variables with linear units, replaced constraints (6) with $Y_{i,t+1} + 0.5 \geq P_{i,L,t}$ and $Y_{i,t+1} - 0.5 \leq P_{i,L,t}$ $\forall_{s_i \in S, t \in [1,H]}$, and enforced the domains of state and action variables, $s_i \in S$ and $a_j \in A$, to be integer for all time steps $t \in [1, H]$ such that $Y_{i,t+1}, X_{j,t} \in \mathbb{Z}$.

[5] The domain simulator predicts the values $\bar{S}^t$ of state variables $s_i \in S$ for time steps $t \in [2, H+1]$ given their initial values V_i and the values $\bar{A}^t$ of action variables A for time steps $t \in [1, H]$ using the transition function T, and outputs the potential violation of constraints that represent functions C and G.



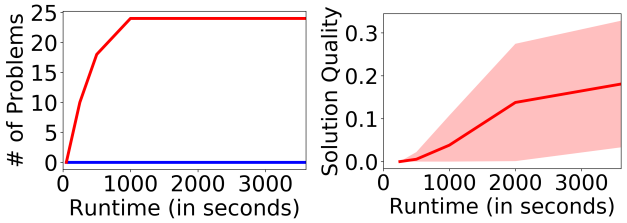
(a) Navigation Cumulative

(b) Navigation Quality



(c) Reservoir Cumulative

(d) Reservoir Quality



(e) Inventory Cumulative

(f) Inventory Quality

Figure 1: Visualization of Experiment 1 which compares Algorithm 1 (red) to the MILP model (Say et al. 2017) (blue) over all instances of the learned automated planning problem $\tilde{\Pi}$ within one hour time limit. The subfigures are organized such that the left column represents the cumulative number of problem instances for which a solution is found and the right column represents the normalized solution qualities, for each domain that is represented by a unique row.

Domain	Total Verif.	Max. Violation	Max. Gap
Navigation	7/10	x $<$ 0.3	x $\leq$ 4%
Reservoir	24/24	-	-
Inventory	24/24	-	-

Table 4: Summary of plan verification and solution quality results for Algorithm 1. We report the total number of problem instances verified, the maximum violation of constraints that represent function C or G, and the maximum duality gap found by solving the MINLP model of Π.

Plan Verification. The inspection of Table 4 highlights the ability of Algorithm 1 to find plans for the automated planning problem Π without having direct access to the transition function T. We observe that Algorithm 1 finds plans for all problem instances in Reservoir Control and Inventory Control, and 70% of the problem instances in Navigation. Among the three violated problem instances, the domain simulator identified constraints that represent function G to be violated by relatively small amounts (i.e., less than 0.3 for constraint $6 \leq Y_{i,H+1} \leq 7$).

Solution Quality. We report the quality of plans found by Algorithm 1 using the dual bounds obtained from solving the MINLP model of the automated planning problem Π. Here, we report results on the only domain that can be solved by the current implementation of the spatial branch-and-bound solver SCIP 7.0 (Vigerske and Gleixner 2018) (i.e., due to the trigonometric functions involved in functions T). Overall as presented in Table 4, the plans found by Algorithm 1 are at most 4% worse than the best computed dual solution. With these results, we experimentally validate the ability of Algorithm 1 to compute high quality solutions to automated planning problems with unknown transition models.

Discussion and Related Work

In this section, we discuss the importance of our contributions and experimental results in relation to the literature with the purpose of opening new areas for future work.

In Experiment 1, we demonstrated the ability of our contributions to effectively solve a variety of challenging learned automated planning problems. Our results suggest that these novel contributions have potential applications to related tasks such as model-based discrete-time planning (Wu, Say, and Sanner 2017), model-based continuous-time planning (Say and Sanner 2018a, 2019), hindsight optimization for model-based planning (Raghavan et al. 2017), probabilistic goal recognition using deep neural network models (Pereira et al. 2019) and automated planning with learned binarized neural networks (Say and Sanner 2018b, 2020), which all rely on effective decision making over models that can be represented as acyclic directed graphs.

In Experiment 2, we demonstrated the ability of our unifying framework to effectively solve a variety of challenging automated planning problems with unknown transition models. Similar to the previous work (Say et al. 2017), we learned models of the world from data using deep neural networks. An important area of future work here is to study how to learn deep neural network models with formal guarantees for the task of automated planning. To this end, the literature on formal methods for deep neural networks including verification (Katz et al. 2017; Ehlers 2017; Huang et al. 2017b; Anderson et al. 2019), robustness evaluation (Tjeng, Xiao, and Tedrake 2019), defenses to adversarial attacks (Wong and Kolter 2018) and robust training (Gowal et al. 2019) present important ideas for safe automated planning with formally robust deep neural network models.

Conclusion

In this paper, we studied the important problem of automated planning with learned deep neural networks. We introduced (i) a novel Lagrangian RNN model to encode the full specifications of a planning problem and (ii) a unifying framework that combines the scalable Lagrangian RNN model with the sound and complete MILP model. We experimentally demonstrated the effectiveness of our contributions to solve challenging automated planning problems with unknown transition models using deep neural networks.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the Twelfth USENIX Symposium on Operating Systems Design and Implementation*, 265–283.

Anderson, R.; Huchette, J.; Tjandraatmadja, C.; and Vielma, J. P. 2019. Strong Mixed-Integer Programming Formulations for Trained Neural Networks. In Lodi, A.; and Nagarajan, V., eds., *Integer Programming and Combinatorial Optimization*, 27–42. Cham: Springer International Publishing. ISBN 978-3-030-17953-3.

Bennett, S. W.; and DeJong, G. F. 1996. Real-World Robotics: Learning to Plan for Robust Execution. In *Machine Learning*, volume 23, 121–161.

Benson, S. S. 1997. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. thesis, Stanford University, Stanford, CA, USA.

Davies, T. O.; Pearce, A. R.; Stuckey, P. J.; and Lipovetzky, N. 2015. Sequencing Operator Counts. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling*, 61–69. AAAI Press. ISBN 978-1-57735-731-5.

Ehlers, R. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In D'Souza, D.; and Narayan Kumar, K., eds., *Automated Technology for Verification and Analysis*, 269–286. Cham: Springer International Publishing. ISBN 978-3-319-68167-2.

Faulwasser, T.; and Findeisen, R. 2009. Nonlinear Model Predictive Path-Following Control. In *Nonlinear Model Predictive Control - Towards New Challenging Applications*, 335–343. Berlin, Heidelberg: Springer.

Gil, Y. 1992. *Acquiring Domain Knowledge for Planning by Experimentation*. Ph.D. thesis, Carnegie Mellon University, USA.

Gowal, S.; Dvijotham, K. D.; Stanforth, R.; Bunel, R.; Qin, C.; Uesato, J.; Arandjelovic, R.; Mann, T.; and Kohli, P. 2019. Scalable Verified Training for Provably Robust Image Classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV.

Helmert, M. 2006. The Fast Downward Planning System. In *Journal Artificial Intelligence Research*, volume 26, 191–246. USA: AI Access Foundation. ISSN 1076-9757.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. In *Journal of Artificial Intelligence Research*, volume 14, 253–302. USA: AI Access Foundation. ISSN 1076-9757.

Huang, G.; Liu, Z.; Maaten, L. v. d.; and Weinberger, K. Q. 2017a. Densely Connected Convolutional Networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2261–2269. ISSN 1063-6919.

Huang, X.; Kwiatkowska, M.; Wang, S.; and Wu, M. 2017b. Safety Verification of Deep Neural Networks. In Majumdar, R.; and Kunčak, V., eds., *Computer Aided Verification*, 3–29. Cham: Springer International Publishing. ISBN 978-3-319-63387-9.

Katz, G.; Barrett, C.; Dill, D.; Julian, K.; and Kochenderfer, M. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the Twenty-Ninth International Conference on Computer Aided Verification*, CAV.

Kautz, H.; and Selman, B. 1992. Planning as Satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, ECAI'92, 359–363.

Kuhn, H. W.; and Tucker, A. W. 1951. Nonlinear Programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, 481–492. Berkeley, California: University of California Press.

Mann, T.; and Mannor, S. 2014. Scaling Up Approximate Value Iteration with Options: Better Policies with Fewer Iterations. In Xing, E. P.; and Jebara, T., eds., *Proceedings of the Thirty-First International Conference on Machine Learning*, volume 32 of *Machine Learning Research*, 127–135. Bejing, China: PMLR.

Nair, V.; and Hinton, G. E. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the Twenty-Seventh International Conference on International Conference on Machine Learning*, ICML, 807–814. USA: Omnipress. ISBN 978-1-60558-907-7.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558608567.

Pereira, R. F.; Vered, M.; Meneguzzi, F.; and Ramírez, M. 2019. Online Probabilistic Goal Recognition over Nominal Models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, IJCAI, 5547–5553. International Joint Conferences on Artificial Intelligence Organization.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014. LP-Based Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, ICAPS'14, 226–234. AAAI Press.

Raghavan, A.; Sanner, S.; Tadepalli, P.; Fern, A.; and Khardon, R. 2017. Hindsight Optimization for Hybrid State and Action MDPs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI. San Francisco, USA.

Say, B. 2020. *Optimal Planning with Learned Neural Network Transition Models*. Ph.D. thesis, University of Toronto, Toronto, ON, Canada.

Say, B.; Devriendt, J.; Nordström, J.; and Stuckey, P. 2020. Theoretical and Experimental Results for Planning with Learned Binarized Neural Network Transition Models. In Simonis, H., ed., *Proceedings of the Twenty-Sixth International Conference on Principles and Practice of Constraint*

Programming, 917–934. Cham: Springer International Publishing. ISBN 978-3-030-58475-7.

Say, B.; and Sanner, S. 2018a. Metric Nonlinear Hybrid Planning with Constraint Generation. In *PlanSOpt 2018*, 19–25. 28th ICAPS Workshop on Planning, Search and Optimization (PlanSOpt).

Say, B.; and Sanner, S. 2018b. Planning in Factored State and Action Spaces with Learned Binarized Neural Network Transition Models. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI, 4815–4821. ISBN 978-0-9992411-2-7.

Say, B.; and Sanner, S. 2019. Metric Hybrid Factored Planning in Nonlinear Domains with Constraint Generation. In *Proceedings of the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, CPAIOR, 502–518.

Say, B.; and Sanner, S. 2020. Compact and Efficient Encodings for Planning in Factored State and Action Spaces with learned Binarized Neural Network Transition Models. *Artificial Intelligence* 285: 103291. ISSN 0004-3702.

Say, B.; Sanner, S.; and Thiébaux, S. 2019. Reward Potentials for Planning with Learned Neural Network Transition Models. In Schiex, T.; and de Givry, S., eds., *Proceedings of the Twenty-Fifth International Conference on Principles and Practice of Constraint Programming*, 674–689. Cham: Springer International Publishing. ISBN 978-3-030-30048-7.

Say, B.; Wu, G.; Zhou, Y. Q.; and Sanner, S. 2017. Nonlinear Hybrid Planning with Deep Net Learned Transition Models and Mixed-integer Linear Programming. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI, 750–756. ISBN 978-0-9992411-0-3.

Shen, W.-M.; and Simon, H. A. 1989. Rule Creation and Rule Learning through Environmental Exploration. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, IJCAI'89, 675—-680. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Song, Z. 2019. Extending TensorFlow-based Planner to Constrained State and Action Spaces. Undergraduate thesis, University of Toronto, Toronto, ON, Canada.

Tjeng, V.; Xiao, K.; and Tedrake, R. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the Seventh International Conference on Learning Representations*, ICLR.

Vigerske, S.; and Gleixner, A. 2018. SCIP: Global Optimization of Mixed-Integer Nonlinear Programs in a Branch-and-cut Framework. *Optimization Methods and Software* 33(3): 563–593.

Wong, E.; and Kolter, Z. 2018. Provable Defenses Against Adversarial Examples via the Convex Outer Adversarial Polytope. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning*, ICML.

Wu, G.; Say, B.; and Sanner, S. 2017. Scalable Planning with Tensorflow for Hybrid Nonlinear Domains. In *Proceedings of the Thirty-First International Conference on Neural Information Processing Systems*, NIPS, 6276–6286. USA: Curran Associates Inc. ISBN 978-1-5108-6096-4.

Wu, G.; Say, B.; and Sanner, S. 2020. Scalable Planning with Deep Neural Network Learned Transition Models. *Journal of Artificial Intelligence Research* 68: 571–606.

Yeh, W. G. 1985. Reservoir Management and Operations Models: A State-of-the-art Review. In *Water Resources research*, volume 21,12, 1797–1818.