On the Relationship Between Stochastic Satisfiability and Partially Observable Markov Decision Processes

Ricardo Salmon and **Pascal Poupart**
David R. Cheriton School of Computer Science, Waterloo AI Institute, University of Waterloo, Canada
Vector Institute, Toronto, Canada
{rsalmon, ppoupart}@uwaterloo.ca

Abstract

Stochastic satisfiability (SSAT) and decision-theoretic planning in finite horizon partially observable Markov decision processes (POMDPs) are both PSPACE-Complete. We describe constructive reductions between SSAT and flat POMDPs that open the door to comparisons and future cross-fertilization between the solution techniques of those problems. We also propose a new SSAT solver called Prime that incorporates recent advances from the SAT and #SAT literature. Using our reduction from POMDP to SSAT, we demonstrate the competitiveness of Prime on finite horizon POMDP problems.

1 INTRODUCTION

Partially observable Markov decision processes (POMDPs) provide a flexible framework for planning under uncertainty when action effects are uncertain and the state of the environment is partially observable. However, planning with finite-horizon flat POMDPs is notoriously difficult since the problem is PSPACE-Complete [24]. State of the art solvers for flat POMDPs [14, 26, 34] can tackle problems on the order of 10^4 states (although other statistics such as the covering number have been advocated as better indicators of difficulty [15]). Factored POMDPs can represent succinctly much larger planning problems since the state space is implicitly defined as the cross product of the domains of many state variables, but factored POMDPs are EXP-hard [19] and therefore even harder to solve.

In a separate line of work, tremendous progress has been made in solving Boolean satisfiability (SAT) problems despite the fact that SAT is NP-Complete. State of the art solvers can now solve SAT problems on the order of 10^5 variables and 10^7 clauses reasonably quickly [10]. If we treat each joint assignment of the binary variables as a state, this means that modern solvers effectively search in a space on the order of $2^{(10^5)}$ states. This remarkable success has lead many researchers to investigate reductions of planning to satisfiability [11, 12, 28, 27], which have been quite successful for deterministic planning.

A stochastic extension of satisfiability called stochastic satisfiability (SSAT) has also been considered to model planning problems with uncertain action effects and partially observable states [18, 22]. In fact, SSAT is PSPACE-Complete [25], which means that SSAT and flat POMDPs can express the same space of planning problems. State of the art solvers such as Zander [22], DC-SSAT [21] and APPSAT [20] can tackle SSAT problems on the order of 10^3 variables and clauses, which means that they search a space on the order of $2^{(10^3)}$ states. Nevertheless, solvers and benchmarks for SSAT and POMDPs remain largely separate and to this day there has not been any cross-fertilization.

We make four contributions:

1. a constructive reduction from SSAT to POMDP,
2. a constructive reduction from POMDP to SSAT,
3. a new SSAT solver called Prime that incorporates watch literals, component decomposition and symmetry detection from the SAT and #SAT literature,
4. an empirical comparison of Prime with exact finite horizon POMDP solvers.

The reductions in our encoding demonstrate that

1. clauses in satisfiability correspond to states in flat POMDPs and
2. variables in satisfiability determine the planning horizon in flat POMDPs.

It is possible to design a reduction that maps states in satisfiability to states in POMDPs. However this reduction yields factored POMDPs with exponentially many states that are EXP-hard. Using our reduction from flat POMDP to SSAT, we present the first empirical comparison (to our knowledge) between exact solvers for SSAT and finite horizon POMDPs.

2 BACKGROUND

We briefly review POMDPs and Boolean satisfiability. Boolean satisfiability solvers have improved tremendously in the last 50 years, which encouraged their use in numerous application domains including planning [11], scheduling [9] and hardware verification [32].

2.1 SAT

The Boolean satisfiability problem or SAT is to determine if it is possible to find a joint variable assignment to a Boolean formula F that evaluates to $true$. Without loss of generality, it is sufficient to consider formulas composed of AND, OR, NOT operators such as

$$F = (x_3 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \quad (1)$$

SAT is the first known NP-complete problem [7]. Without loss of generality, we consider SAT problems consisting of formulas in conjunctive normal form (CNF), i.e., a conjunction of clauses where a clause is a conjunction of literals (see Eq. 1 for an example).

2.2 Stochastic Satisfiability (SSAT)

Stochastic satisfiability was first proposed by [25] as a generalization of Boolean satisfiability where each variable x_i has an existential $\exists$ or a randomize $\reflectbox{R}$ quantifier.

$$\exists x_1 \reflectbox{R} x_2 \exists x_3 ... \reflectbox{R} x_n F(x_1, x_2, ..., x_n) \quad (2)$$

A policy tree is a tree that assigns values to existentially quantified variables and branches on randomized variables according to the quantification order. In SSAT, the goal is to find a policy tree, ϕ, that maximizes the probability that a Boolean formula $F(x_1, x_2, ..., x_n)$ is true. This probability depends on the distribution of the randomized variables. We consider discrete variables x_i that can take more than two values. The SSAT problem in Eq. 2 corresponds to the optimization problem

$$\max_{x_1} \sum_{x_2} \Pr(x_2) \max_{x_3} ... \sum_{x_n} \Pr(x_n) \delta(F(x_1, x_2, ..., x_n)) \quad (3)$$

where $\delta(true) = 1$ and $\delta(false) = 0$.

2.3 Partially Observable Markov Decision Processes (POMDPs)

Partially Observable Markov Decision Processes (POMDPs) provide a principled framework for planning under uncertainty. Formally, a (flat) POMDP is specified by a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, R, b_0, h)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{O}$ is a set of observations, $T(s, a, s') = \Pr(s'|s, a)$ is the transition distribution, $\Omega(o, a, s') = \Pr(o|s', a)$ is the observation distribution, $R(s, a)$ is the reward function, $b_0(s) = \Pr(s_0)$ is the initial belief and h is the planning horizon. We will assume a finite horizon h and we will consider non-stationary dynamics by allowing different transition, observation and reward functions at different time steps.

A POMDP solution is also a policy tree, π, that assigns an action at each timestep and branches on the value of each possible observation in a way that the expected sum of rewards is maximized. We can also represent a policy as a mapping from beliefs (distributions over states based on the past history of actions and observations) to actions. A belief b can be updated after executing action a and receiving observation o according to Bayes' theorem: $b_o^a(s') \propto \sum_s Pr(s'|s, a) Pr(o|s', a) b(s)$.

3 RELATED WORK

The satisfiability of Quantified Boolean Formula (QBF) provides another generalization of SAT (with existential and universal quantifiers) that is PSPACE-Complete. In the propositional planning literature, [4, 31, 3] describe reductions between satisfiability problems (including QBF) and various planning problems (including conformant, conditional and contingent planning). Since this work does not deal with numerical values in the form of probabilities and utilities, we focus on reductions between stochastic SAT and decision theoretic planning in the form of flat POMDPs. [16, 17] summarize the complexity of various formalisms for flat and propositional probabilistic planning. Note that propositional planning can be seen as a special case of factored POMDPs and therefore reductions from various forms of (non-stochastic) satisfiability to propositional planning can be used to directly obtain factored POMDPs. However, factored POMDPs are EXP-hard (i.e., harder than PSPACE) and these reductions yield POMDPs with exponentially many states w.r.t. the original encoding. In contrast, we describe reductions to flat POMDPs (in PSPACE) with linearly many states.

In the POMDP literature, several approaches have been proposed to optimize POMDP policies by probabilistic inference [29, 30]. However, there is no known technique for converting POMDPs to inference problems in probabilistic graphical models without doing an approximation or incurring an exponential blow up in the representation since probabilistic inference problems are in lower complexity classes than PSPACE (i.e., NP for MPE inference, #P for plain inference and NP$^{\#P}$ for marginal-MAP inference). For instance, [33] explain how to reduce the complexity of POMDP planning from PSPACE to lower complexity classes by restricting the policy search to various classes of bounded finite state controllers while understanding that these restrictions may prevent an optimal policy from being found.

The solvers Zander and C-MAXPLAN were introduced by [22] to solve contingent planning under uncertainty. Contingent plans are those which depend on observable variables during execution. Unlike C-MAXPLAN, Zander encodes probabilistic plans in Stochastic SAT. In fact, Zander was the first true stochastic SAT solver to incorporate techniques from satisfiability such as a variable ordering heuristic, unit propagation, pure literal elimination, and thresholding.

4 ENCODING PROBLEMS INTO SSAT

In the following subsections we show constructive reductions for encoding POMDPs into SSAT and vice-versa.

4.1 SSAT ⇒ POMDP

In our reduction from SSAT to POMDP, at each time step, the action corresponds to assigning a value to the next existentially quantified variable and the observation corresponds to assigning a value to the next randomized variable. This is because the maximization over actions can be used to encode an existential quantifier and the expectation with respect to observations can be used to encode a randomize quantifier.

We can reduce a SSAT problem with alternating quantifiers to a POMDP as follows:

- **State space** $\mathcal{S} = \{sat, prob, c_1, c_2, ..., c_{|C|}\}$: A state labeled by c_i indicates that clause c_i has not been satisfied yet. The state labeled by sat can be interpreted as the entire formula is satisfied. In addition, the probability of being in state $prob$ is proportional to the probability of the current path for randomized variable assignments.
- **Initial belief** b_0: The initial belief is set to a uniform distribution, i.e., $b_0(s) = 1/|\mathcal{S}|\ \forall s$.
- **Action space** $\mathcal{A} = \{true, false\}$: Each action is an assignment of $true$ or $false$ to the lowest unassigned existentially quantified variable.
- **Transition function** $\Pr(s_{t+1}|s_t, a_t)$: The transition function is deterministic. When in state $s_t = c_i$, the process will transition to the sat state if the current action satisfies clause c_i. Otherwise, the process remains in the current state.

$$Pr(s_{t+1}|s_t, a_t) \quad (4)$$
$$= \begin{cases} 1 & \text{if } s_t = c_i,\ a_t \text{ satisfies } c_i \text{ and } s_{t+1} = sat \\ 1 & \text{if } s_t = c_i,\ a_t\ \neg\text{satisfy } c_i \text{ and } s_{t+1} = c_i \\ 1 & \text{if } s_t = s_{t+1} = sat \\ 1 & \text{if } s_t = s_{t+1} = prob \\ 0 & \text{otherwise} \end{cases}$$

- **Reward function** $R(s_t, a_t)$: We design a reward function that effectively yields a reward of 1 when the belief at the last time step corresponds to a satisfiable joint assignment and 0 otherwise. A belief that corresponds to a satisfiable assignment has all its mass in the sat and $prob$ states. By giving a reward of $|\mathcal{S}|$ to state $prob$, we cancel the initial probability $b_0(prob) = 1/|\mathcal{S}|$ and the belief effectively earns a reward of 1. A belief that corresponds to a non-satisfiable joint assignment has part of its mass in some state c_i (unsatisfied clause). By assigning a reward of $-|\mathcal{S}|$ to each c_i we penalize the belief by effectively canceling the reward of $|\mathcal{S}|$ in state $prob$. Since the mass in $prob$ is less than the mass in all the c_i's combined, the overall reward is negative. However, since an optimal policy will choose the action with the highest reward and the reward is 0 when $a_t = false$, the effective reward will be 0.

$$R(s_t, a_t) = \begin{cases} |\mathcal{S}| & \text{if } t = \frac{|X|}{2},\ s_t{=}prob,\ a_t = true \\ -|\mathcal{S}| & \text{if } t = \frac{|X|}{2},\ s_t = c_i,\ a_t = true \\ 0 & \text{otherwise} \end{cases}$$

- **Observation space** $\mathcal{O} = \{true, false\}$: Each observation is an assignment of $true$ or $false$ to the lowest unassigned randomized variable.
- **Observation distribution** $\Pr(o_{t+1}|a_t, s_{t+1})$: The observation distribution ensures that $b(c_i)$ becomes 0 when clause c_i is satisfied by the truth value assigned to the observation of the current universally quantified variable. Otherwise, $b(c_i)$ remains greater than 0 when clause c_i has not been satisfied yet. We achieve this effect by defining a deterministic observation distribution for the c_i states and a stochastic observation distribution for the sat and $prob$ states. The stochastic distribution over the observations for the sat and $prob$ states ensures that both truth values are considered for randomized variables. We denote by λ_t the probability that $x_t = true$. The $prob$ state will effectively keep track of the probability of the observation sequence.

States	b_0	b_1^a	$\hat{b}_1^{ao}$	$\hat{b}_2^a$	$\hat{b}_2^{ao}$	$\hat{b}_3^a$
$prob$	1/5	1/5	1/6	1/6	1/7	1/7
sat	1/5	2/5	1/5	1/5	1/10	1/10
$c_1 = \ x_3 \vee \ x_4 \vee \neg x_5$	1/5	1/5	1/5	1/5	0	0
$c_2 = \neg x_1 \vee \neg x_2 \vee \ x_4$	1/5	0	0	0	0	0
$c_3 = \ x_1 \vee \neg x_2 \vee \ x_5$	1/5	1/5	0	0	0	0

Table 1: intermediate (unnormalized) beliefs in SSAT example after processing $a_0 = (x_1 := false)$, $o_1 = (x_2 := false)$, $a_1 = (x_3 := false)$, $o_2 = (x_4 := true)$ and $a_3 = (x_5 := true)$ where $\Pr(o_1 = (x_2 := true)) = 1/6$ and $\Pr(o_2 = (x_4 := true)) = 6/7$.

$$Pr(o_{t+1}|a_t, s_{t+1}) \tag{5}$$
$$= \begin{cases} 0 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1}{:=}o_{t+1}) \text{ satisfies } c_i \\ 1 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1}{:=}o_{t+1}) \text{ ¬satisfy } c_i \\ \frac{1}{2} & s_{t+1} = sat \\ \lambda_t & \text{if } s_{t+1} = prob \text{ and } o_{t+1} = true \\ 1{-}\lambda_t & \text{if } s_{t+1} = prob \text{ and } o_{t+1} = false \end{cases}$$

- **Horizon** $h = |X|/2 + 1$: Two variables (one existential and one randomized) are processed per time step. An additional time step is created at the end for the final reward. Hence there are $|X|/2 + 1$ time steps (from time step 0 to time step $|X|/2$).

Consider again the Boolean formula in Eq. 1. Suppose that we quantify the 5 variables as follows: $\exists x_1, \text{Ɐ} x_2, \exists x_3, \text{Ɐ} x_4, \exists x_5$ where $\Pr(x_2 = true) = \lambda_2 = 1/6$ and $\Pr(x_4 = true) = \lambda_4 = 6/7$. Table 1 reports the intermediate (unnormalized) beliefs that are obtained when we process the sequence of actions and observations $a_0 = (x_1 := false), o_1 = (x_2 := false), a_1 = (x_3 := false), o_2 = (x_4 := true), a_2 = (x_5 := true)$.

Theorem 1. *The reduction from SSAT to POMDP guarantees that there exists a POMDP policy π for time steps 0 to $|X|/2 - 1$ and optimal action at time step $|X|/2$ with value function $V^\pi = Pr(\phi)$ iff there exists a policy tree ϕ with satisfiability probability $Pr(\phi)$.*

See Appendix A for the proof.

4.2 POMDP ⇒ SSAT

Since SAT is NP-Complete while POMDPs are PSPACE-Complete, it is unknown whether it is possible to reduce POMDPs to SAT without an exponential blow up. Hence, encoding POMDPs as SAT is not viewed as tractable. In contrast, since POMDPs and QBF are both PSPACE-Complete it is possible in theory to reduce POMDPs to QBF in polynomial time and space. However, in practice, one needs to convert real values (i.e., probabilities and rewards) into binary encodings and real arithmetic into binary operations. This is obviously possible since current computers perform real arithmetic up to some precision via binary operations. Abio and Stuckey [1] recently showed how to convert integer linear constraints into binary constraints, however it remains impractical to express explicitly real arithmetic as binary operations in the context of a reduction from POMDP to QBF. Hence we restrict our discussion to a reduction of POMDPs to SSAT since the probabilities in SSAT can be used to encode the probabilities and rewards of POMDPs.

The domain of the reward function is $\mathbb{R}$. However, rewards can be scaled and translated without changing the optimal policy. We define a new reward function

$$r(s,a) = \frac{R(s,a) - \min_{a',s'} R(s',a')}{\sum_{a,s}[R(s,a) - \min_{a',s'} R(s',a')]} \tag{6}$$

that can be interpreted as a distribution that sums to 1 and with values in $[0,1]$ for all s and a. We will also work with a generalization of SSAT that is not limited to binary variables, but allows multi-valued discrete variables.

The general idea is to represent the POMDP parameters (probabilities and rewards) as probabilities of randomized variables and the POMDP actions as existential variables in SSAT.

As a starting point, consider a simple POMDP with only one time step (i.e., $h = 1$). In this simple setting, the optimal policy is obtained by computing the expected reward for each action and by selecting the best action:

$$a^* = \arg\max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b_0(s) r(s,a) \tag{7}$$

In the corresponding SSAT encoding, we introduce a variable x_a for the action such that $x_a \in \{0, ..., |\mathcal{A}| - 1\}$. Next, define a Boolean formula[1] that encodes Eq. 7

$$\bigwedge_{k \in \mathcal{A}} \bigwedge_{i \in \mathcal{S}} (x_a \equiv k \wedge x_s \equiv i) \rightarrow x_r \equiv k|\mathcal{S}| + i \tag{8}$$

with quantified variables $\exists x_a$ followed by $\text{Ɐ} x_s$ and $\text{Ɐ} x_r$ in order, where $x_s \in \{0, ..., |\mathcal{S}| - 1\}$ and $x_r \in \{0, 1, ..., |\mathcal{A}||\mathcal{S}| - 1\}$. Here $x \equiv k$ denotes $true$ when $x = k$ and $false$ otherwise. The distributions for the randomized variables are:

$$\Pr(x_s \equiv i) = b(i) \tag{9}$$
$$\Pr(x_r \equiv k|\mathcal{S}| + i) = r(i,k), \forall i, k \tag{10}$$

The first term in Eq. 8 guarantees that whenever an action, k, is taken, all clauses containing the term $x_a \not\equiv j$

[1]While this formula is not in conjunctive normal form (CNF) to ease the exposition, it can easily be converted in CNF.

for $j \neq k$ are removed since they have been satisfied and the term $x_a \not\equiv k$ is removed from the active clauses.

The remaining terms are used to perform the summation of randomized variables. The same reasoning is used to set $x_s \equiv i$. After selecting a state s from the initial belief, only one clause will be active (yet to be satisfied), a unit clause that implies the value of the reward variable based on a state-action combination.

Therefore, with only one literal remaining in one clause, $x_r \equiv k|\mathcal{S}| + i$, our goal is to make the probability of satisfying this clause equal to the reward of taking action k in state i. The optimal action in the original POMDP is recovered in SSAT as an assignment to $x_a \equiv k$ that maximizes the probability of satisfying all the clauses.

Consider general POMDPs with horizon $h > 1$. We can define the optimal value function as follows:

$$V_h(b) = \max_{a \in \mathcal{A}} \sum_s b(s)[r(s,a) + \sum_o \sum_{s'} \Omega^a_{s'o} T^a_{ss'} V_{h-1}(b^a_o)] \tag{11}$$

where an optimal policy maximizes the value function over a horizon h. Based on Eq. 11, we can reduce a POMDP problem with horizon h to SSAT in two steps: i) policy selection and ii) policy evaluation.

Introduce an alternating sequence of variables,

$$\exists x^1_a, \text{ꓤ} x^1_p, \text{ꓤ} x^1_o, \exists x^2_a, \text{ꓤ} x^2_p, \text{ꓤ} x^2_o, \cdots, \text{ꓤ} x^{h-1}_o, \exists x^h_a, \text{ꓤ} x^h_p,$$

for policy selection. The $\exists x^t_a$ variable corresponds to the action taken at time-step t in the original POMDP. The variable x^t_p is an auxiliary variable with domain $\{T, F\}$ and uniform distribution that indicates whether the process stops (F) and a reward is earned, or the process continues to the next time step (T). Each $\text{ꓤ} x^t_o$ has domain $\{0, ..., |\mathcal{O}| - 1\}$ and uniform distribution $\Pr(x^t_o \equiv z) = \frac{1}{|\mathcal{O}|}$. The observation distribution $\Pr(o|s', a)$ will be encoded later during the policy evaluation step. The uniform distribution for each x^t_o will change the scale of all probabilities by a factor of $1/|\mathcal{O}|$ at each time step and therefore we can recover the probability of satisfiability by multiplying by $|\mathcal{O}|^{h-1}$.

Next, policy evaluation computes the value of a policy by introducing the variables:

$$\text{ꓤ} x^t_s, \text{ꓤ} x^t_r \ \forall t \text{ such that } 1 \leq t \leq h \tag{12}$$

$$\text{ꓤ} x^t_\Omega, \text{ꓤ} x^t_T \ \forall t \text{ such that } 1 \leq t \leq h-1 \tag{13}$$

Those randomized variables can appear in any order as long as they are after the variables for policy selection. We will explain the semantics of those variables after introducing the Boolean formulas they appear in:

$$\bigwedge_{1 \leq t \leq h-1} x^t_p \equiv 0 \rightarrow \left(x^t_o \equiv 0 \wedge x^{t+1}_s \equiv 0\right) \tag{14}$$

$$\bigwedge_{1 \leq t \leq h-1} x^t_p \equiv 0 \rightarrow x^{t+1}_p \equiv 0 \tag{15}$$

$$x^h_p \equiv 0 \tag{16}$$

$$\bigwedge_{k \in \mathcal{A}} \bigwedge_{i \in \mathcal{S}} (x^1_p \equiv 0 \wedge x^1_a \equiv k \wedge x^1_s \equiv i) \rightarrow x^1_r \equiv k|\mathcal{S}| + i \tag{17}$$

$$\bigwedge_{2 \leq t \leq h} \bigwedge_{k \in \mathcal{A}} \bigwedge_{i \in \mathcal{S}} (x^{t-1}_p \equiv 1 \wedge x^t_p \equiv 0 \wedge x^t_a \equiv k \wedge x^t_s \equiv i) \rightarrow x^t_r \equiv k|\mathcal{S}| + i \tag{18}$$

$$\bigwedge_{1 \leq t \leq h-1} \bigwedge_{k \in \mathcal{A}} \bigwedge_{i \in \mathcal{S}} \bigwedge_{j \in \mathcal{S}} (x^t_p \equiv 1 \wedge x^t_a \equiv k \wedge x^t_s \equiv i \wedge x^{t+1}_s \equiv j) \rightarrow x^{t+1}_{T_{k,i}} \equiv j \tag{19}$$

$$\bigwedge_{1 \leq t \leq h-1} \bigwedge_{k \in \mathcal{A}} \bigwedge_{j \in \mathcal{S}} \bigwedge_{z \in \mathcal{O}} (x^t_p \equiv 1 \wedge x^t_a \equiv k \wedge x^{t+1}_s \equiv j \wedge x^t_o \equiv z) \rightarrow x^t_{\Omega_{k,j}} \equiv z \tag{20}$$

The formula in Eq. 15 ensures that once the process has stopped, it doesn't continue in the future and Eq. 14 guarantees that if the process stopped early, the observation and state auxiliary variables will all be assigned some arbitrary value as a normalization constant. x^t_s and x^t_o are used because they function as indicator variables and hence their distribution is uninformative. Since the horizon is finite, the formula in Eq. 16 ensures that the process is necessarily stopped at the horizon h.

The variable x^t_s encodes the state at time step t and has uniform distribution $\Pr(x^t_s \equiv i) = \frac{1}{|\mathcal{S}|} \ \forall i \in \mathcal{S}$. The variable x^t_r has domain $\{0, 1, ..., |\mathcal{S}||\mathcal{A}| - 1\}$ and has a distribution proportional to the rewards

$$\Pr(x^t_r \equiv k|\mathcal{S}| + i) = r(i,k), \ \forall k \in \mathcal{A}, i \in \mathcal{S} \tag{21}$$

that encodes the reward to be received for a particular action, k, and state, i, pair as a probabilistic value. The formula in Eq. 17 ensures that the process receives a reward at the first time step when $x^1_p \equiv F$, while Eq. 18 yields a reward at subsequent time steps when $x^{t-1}_p \equiv T$ changes to $x^t_p \equiv F$.

The variable $x^{t+1}_{T_{k,i}}$ has domain $\mathcal{S}$ and encodes the transition distribution after executing action k in state i

$$\Pr(x^{t+1}_{T_{k,i}} \equiv j) = \Pr(s_{t+1} = j | s_t = i, a_t = k) \tag{22}$$

The formula in Eq. 19 encodes this transition. Similarly, the variable $x^{t+1}_{\Omega_{k,j}}$ has domain $\mathcal{O}$ and encodes the observation distribution after executing action k and arriving in state j

$$\Pr(x^t_{\Omega_{k,j}} \equiv z) = \Pr(o_{t+1} = z | s_{t+1} = j, a_t = k) \tag{23}$$

The formula in Eq. 20 encodes this distribution. The following theorem confirms the equivalence of the SSAT problem obtained from a POMDP by the reduction.

Theorem 2. *In the reduction of POMDP to SSAT, there exists a satisfiable policy tree, ϕ, with probability $\Pr(\phi)$ iff there exists a POMDP policy, π, with value function $V^\pi = \Pr(\phi)$.*

See Appendix A for the proof.

4.3 Discussion

In the reduction of SSAT to POMDP, we showed that the number of clauses $|C|$ determines the number of POMDP states and the number of variables $|X|$ determines the planning horizon. This is surprising since the usual belief is that satisfiability solvers operate in a state space of size $2^{|X|}$, which is usually much larger than the size of the state spaces of flat POMDPs that are commonly tackled. In contrast, our reduction shows that $O(|C|)$ states are sufficient in the resulting POMDP. It is possible to consider different reductions that will map each joint assignment in satisfiability to a POMDP state. However, these reductions yield an exponential blow up in the number of states and therefore are clearly intractable. Alternatively, one can also construct reductions from satisfiability to factored POMDPs by associating Boolean variables to POMDP state variables. While these reductions do not yield an exponential blow up, they map satisfiability problems to an artificially more complex class of problems since factored POMDPs are EXP-hard.

In the reverse reduction of POMDP to SSAT, the number of variables and clauses is polynomial in the parameters of the original POMDP. The number of variables in the equivalent SSAT problem is 3 variables initially (action, current state, and reward) and $5 + |\mathcal{A}||\mathcal{S}| + |\mathcal{A}||\mathcal{O}|$ per time step for the remaining $h-1$ time steps. This gives a complexity of:

$$|X| = O\Big(h|\mathcal{A}|\big(|\mathcal{S}| + |\mathcal{O}|\big)\Big) \tag{24}$$

The number of clauses initially is $|\mathcal{A}||\mathcal{S}|$ and $1+|\mathcal{A}||\mathcal{S}|+|\mathcal{A}||\mathcal{S}|^2+|\mathcal{A}||\mathcal{S}||\mathcal{O}|$ per time step for the remaining $h-1$ time steps. This gives a complexity of:

$$|C| = O\Big(h|\mathcal{A}||\mathcal{S}|\big(|\mathcal{S}| + |\mathcal{O}|\big)\Big) \tag{25}$$

5 SSAT SOLVER

We describe our Stochastic SAT solver, SSAT-Prime, with many successful techniques incorporated from the SAT and #SAT literature including watch literals, component decomposition, and symmetry.

5.1 Solver Overview

SSAT Prime is structurally similar to the Davis–Putnam–Logemann–Loveland (DPLL) algorithm where given a formula F, first decompose the problem into a set of components, f_i, that are each solved separately as shown in Algorithm (1). The solution for each component is cached and reused if a similar component is detected (using symmetry) in the future. See code at https://github.com/rsalmon/prime.

Algorithm 1 Solve a SSAT problem by decomposition.

```
1: procedure SOLVE-PROBLEM(F)
2:     $\phi \leftarrow 1$
3:     for $f_i \in$ component-decomposition$(F)$ do
4:         if clauses$(f_i) == 0$ then
5:             go-to-next-component
6:         $c_i \leftarrow$ symmetry-encoding$(f_i)$
7:         if $\neg$cache-contains$(c_i)$ then
8:             $p \leftarrow$ solve-component$(f_i)$
9:             cache-save$(c_i, p)$
10:        $\phi \leftarrow \phi \cdot$ cache-value$(c_i)$
11:        if $\phi \equiv 0$ then
12:            return 0
13:    return $\phi$
```

To solve a sub-problem in Alg. 2, pick an unassigned variable from the lowest quantifier block and for each value, simplify the formula by unit propagation using the specified watch literal scheme. If a conflict is found, this partial assignment is no longer expanded. Otherwise, recursively solve the new problem. Finally, update the probability ϕ of satisfiability using a max for existential quantifiers and a sum for randomize quantifiers.

Algorithm 2 Compute solution for a single component.

```
1: procedure SOLVE-COMPONENT(f)
2:     $\phi \leftarrow 0$
3:     $x \leftarrow$ choose-variable$(f)$
4:     for $x_i \in values(x)$ do
5:         $\hat{p}_i, f' \leftarrow$ constraint-propagate$(f, x = x_i)$
6:         if $\neg$conflict$(f')$ then
7:             $p_i \leftarrow$ solve-problem$(f')$
8:             if $Q(x) \equiv \exists$ then
9:                 $\phi \leftarrow \max(\phi, \hat{p}_i \cdot p_i)$
10:            else if $Q(x) \equiv \text{Я}$ then
11:                $\phi \leftarrow \phi + \hat{p}_i \cdot p_i$
12:    return $\phi$
```

5.2 Improved Watch Literal Scheme

In satisfiability problems, the unit rule says that if a clause, c, has only two unassigned literals such that $c = (x_1 \equiv v_0 \vee x_2 \not\equiv v_3)$, then any assignment that falsifies a literal, say $x_2 \equiv v_3$, will leave $c = (x_1 \equiv v_0)$. A clause with only one unassigned literal is a unit clause. It is valid to assign $x_1 = v_0$ and to satisfy c since in the CNF representation all clauses must be satisfied to attain satisfiability. If the formula is unsatisfiable, then we will eventually reach a conflict.

More importantly, if any part of the assignment falsifies a unit clause, the current partial assignment has led to a conflict and all future assignments that use the current partial assignment are unsatisfiable. All unit rule assignments are propagated to other clauses until no more deductions can be made. In addition, for SSAT, when the variable being assigned is randomly quantified, we need to scale the probability of satisfiability by the probability of the variable taking that particular value.

The watch literal scheme is an efficient way to determine assignments from the unit rule [23]. We would like to minimize the number of clauses we check when assigning a variable and unassigning in backtracking. As explained above, it is only the last two unassigned literals that play a role in determining when a unit clause occurs. The idea is for each clause to always be tracking 2 unassigned literals such that when a new assignment, $x = v$, is made, we only visit clauses that are watching literals consistent with $x \neq v$.

A disadvantage of the watch literal scheme is that we may end up checking all literals in a clause when searching for a replacement watch literals even though the clause is already satisfied. This seems very wasteful and this could be a source of extended solution time for problems with a large number of literals per clause.

The idea for our improvement is a constant time statistic that can determine if a clause is satisfied. We can do this by using a stack, **satisfy**, to hold the current list of satisfied clauses in order and a field, **clause.satisfied**, for each clause that indexes the position in the stack that the particular clause is satisfied. Whenever a clause, c, is known to be satisfied, we can perform the updates:

$$\text{clauses[c].satisfied = stack-size(satisfy)} \tag{26}$$

$$\text{stack-push(satisfy, c)} \tag{27}$$

In Algorithm 3, we define **is-satisfied** that determines if a clause is satisfied in constant time. For a clause to be satisfied, it must satisfy two conditions: (1) the index of the stack corresponding to the clause must point to itself and (2) the clause's index into the stack must be at most the stack size.

Algorithm 3 Check if clause is satisfied in constant time.

```
1: procedure IS-SATISFIED(satisfy, c)
2:     s = clauses[c].satisfied
3:     return stack-index(satisfy, s) $\equiv$ c $\wedge$ s <stack-size(satisfy)
```

To show the rule is consistent and complete, first, if the clause at position c is satisfied, then it is added to the satisfy stack previously and its position in the stack will be at least less than or equal to the current size. **clause.satisfied** will point to that location in the stack which will contain c. Otherwise, if the clause at position c is not satisfied then **clause.satisfied** does not point to a valid position in the stack or the value at that position is different from c since there was never a reason to assign it to c. During backtracking, all that is required is to update the size of the stack in constant time.

5.3 Component Decomposition and Symmetry

Component decomposition was introduced by [2] in the context of #SAT for model counting. Given a graph G, the idea is to look at the constraint graph and find maximal connected subgraphs, called components. The constraint graph is derived by representing each variable $x \in X$ as a vertex and each pair of vertices x, y has an edge if they appear in an active clause together.

Consider a constraint graph, G, using variables X with a complete set of components $G_1, G_2, ..., G_{|G|}$ over variable subsets $X_1, X_2, ..., X_{|G|}$ that share no elements. The probability of satisfiability is $\Pr(X) = \prod_i^{|G|} \Pr(X_i)$, which means that the complexity is not determined by G, but the largest subgraph G_i. The idea should work well on problems that are highly decomposable and are made up of highly reusable subproblems. With component decomposition our worst case complexity for searching through all solutions is reduced from $2^{O(n)}$ to $2^{O(w)}$ for a binary variable problem with n variables and a tree width of w [13, 8].

Symmetry in SSAT problems is represented by a subset of variables, $X_\pi \subseteq X$, that forms a solution with satisfying probability $\Pr(X_\pi)$, but there exist further solutions that are permutations of the variables $Y_\pi \in \sigma(X_\pi)$ such that $\Pr(Y_\pi) \equiv \Pr(X_\pi)$ for all Y_π. Symmetries are an issue because they artificially increase the size of the search space with extra solutions that are effectively the same, but need to be enumerated independently.

Usually, breaking symmetries requires additional constraints to enforce only one assignment from each equivalence class. This can be done by sorting the solutions according to some ordering. However, in SSAT, we are still required to enumerate all the solutions to calculate the probability of satisfiability accurately so instead we focus on symmetric components as subproblems. Previous work in SAT has been on static symmetry breaking where the procedure is only applied as a preprocessing step in the solver. We focus on dynamic or conditional symmetries that occur during the execution of the solver.

In [13] they used graph canonization on CSPs to find a canonical labeling of a graph that is invariant to symmetries in the variables, values or some mixture. This reduces symmetry detection to graph isomorphism where we are tasked with determining if two finite graphs are isomorphic. Graph isomorphism is known to be in NP, but it is unknown whether it is also NP-complete.

Benchmark	Problem	Unit-Rule	UR-Improved	Speedup	Satisfied
RANDOM	fail-learn1	2.79	**2.73**	1.02	64.68%
	pure1	2.91	**2.85**	1.02	64.92%
	big1	72.03	**63.88**	1.13	99.24%
	big2	8.36	**6.69**	1.25	85.38%
POMDP	tiger.95_H10	4.79	**4.67**	1.03	88.55%
	ejs7_H10	64.35	**63.56**	1.01	81.08%
	query.s4_H2	113.56	**21.18**	5.36	99.99%
	aloha.10_H3	13.41	**6.66**	2.01	99.94%
INFERENCE	mastermind_04_08	27.80	**27.69**	1.00	64.88%
	fs-29.uai	12.33	**12.06**	1.02	87.90%

Table 2: Improvement in time (sec) to the watch literal rule. The suffix HN in the name of each POMDP problem indicates the planning horizon (i.e., H10 indicates a horizon of 10 steps).

Given a subproblem, $S = (X, C)$, that is constrained by clauses C using variables X, we can define a coloured graph, G=(V, E), such that we have

1. a vertex for each clause with colour 0
2. a vertex for each variable with colour corresponding to quantifier level
3. a vertex for each existential literal with a shared unique colour
4. a vertex for each randomized literal where each literal share the same colour iff they share the same probability and the colours are unique
5. a directed edge connecting each clause vertex to all the literal vertices it contains
6. a directed edge connecting each variable vertex to its associated literal vertices

After the graph is set up, call a graph library to perform canonization. In the relabeling, vertices that share a similar structure are exchangeable and a natural ordering is found. Note that only vertices of the same type can be exchanged since vertices can only be swapped iff they share the same colour and degree. This guarantees that a clause and a variable vertex are never swapped. The relabeling of the graph vertices is used for future identification of the component and storage in the cache.

In addition, we propose Component Projection, which is a simplification of graph canonization that only does the mapping to a coloured graph without graph relabeling. The most expensive part of the encoding, the canonization, is avoided and most subproblems would still be a match since the act of relabeling vertices from 1 to $|X|$ provides most of the expressive power. That is, perfect graph canonization is not that helpful for most problems since the restriction on quantifier ordering limits the order of variables in a subproblem.

6 EXPERIMENTS

In this section, we show some experiments that validate key features of the SSAT-Prime solver. All experiments were conducted on an Intel i5 at 3.5GHz with 4GB of available RAM. In each scenario, the solvers had $1,000$ seconds to solve each problem before timing out. All POMDP benchmarks were converted to SSAT with the reduction technique described in Section 4.1.

6.1 Unit Rule

Our proposed improvement to the watch literal rule is by keeping a lazy structure that allows us to determine in constant time if a clause is satisfied. In Table 2, we show the results of both unit rule algorithms on a variety of problems. The properties, statistics and origin of those problems are described in Table 1 in Appendix B.UR-Improved is the new algorithm and Unit-Rule is the original implementation. Our approach yielded an improvement in time (seconds) across all benchmarks. Moreover, improvements in running time (of 1.13x to 5.36x are achieved for the problems with longer clauses such as big1, big2, query and aloha).

The last column indicates the percentage of all clauses visited by the unit rule that were already satisfied and hence a waste of effort to search. The problems that showed the most improvement also had most of the visited clauses already satisfied over 99% of the time. This makes sense since longer clauses are more likely to be satisfied by at least one of the many literals when searching for a replacement watch literal.

6.2 Symmetry

We compare the effects of using symmetry. First, consider Component Basic (CB) where a component is recorded as a set of unassigned literals for each clause. Next, we explore using Component Canonical (CC) that encodes each component into a coloured graph and finds a canonical relabeling of the variables that is invariant to any permutation of variables or clauses. Finally, Component Project (CP) encodes a component into a graph, but relabels the variables sequentially. The results are in Table 3 where duration is in seconds, #C indicates the number of components generated and the last column indicates the percentage of reused components. The benchmarks are the same as for the previous experiment.

Unfortunately, CC's performance was the worst in a majority of the benchmarks (9 out of 10). However, in 7 cases, CP performed the best with up to a 10x speed improvement. In general, we found the results hold across most problems from the POMDP benchmarks. Note that there is an important improvement over CB only when the number of components is significantly reduced. This means that the extra time on the other problems is pure overhead and maybe there is room for improvement here.

Problem	Symmetry	Duration	#C	Reuse %
fail-learn1	CB	**2.70**	457,209	56.89%
	CC	9.16	471,469	56.99%
	CP	2.72	442,921	56.74%
pure1	CB	**2.72**	471,269	55.34%
	CC	9.72	480,377	55.27%
	CP	2.85	473,939	55.37%
big1	CB	64.34	13,249,524	61.80%
	CC	171.44	13,801,715	62.21%
	CP	**63.78**	13,549,226	62.14%
big2	CB	6.96	955,347	26.14%
	CC	18.77	955,567	26.42%
	CP	**6.68**	947,171	27.61%
tiger.95_H10	CB	X	133,713,296	84.13%
	CC	20.86	1,111,885	89.38%
	CP	**4.69**	1,111,885	89.38%
ejs7_H10	CB	**56.99**	7,748,117	81.95%
	CC	220.12	7,748,117	81.95%
	CP	63.87	7,747,733	81.95%
query.s4_H2	CB	24.47	2,264,792	99.98%
	CC	35.70	2,264,792	99.98%
	CP	**21.04**	2,264,792	99.98%
aloha.10_H3	CB	68.01	514,004	97.24%
	CC	18.69	57,828	97.32%
	CP	**6.67**	57,828	97.32%
mastermind_04_08	CB	29.98	69,252	38.35%
	CC	155.05	69,252	38.35%
	CP	**27.67**	69,220	38.43%
fs-29	CB	13.54	254,205	99.42%
	CC	258.56	254,205	99.47%
	CP	**12.01**	254,201	99.47%

Table 3: Results for improvement in symmetry by Canonical and Projection relabeling.

Therefore, the cost of CC was too high, especially on the RANDOM and INFERENCE benchmarks.

6.3 POMDP Encoding

In this section, we compare exact POMDP and SSAT solvers on finite horizon POMDPs converted to SSAT. For the reverse direction, from SSAT to POMDP, the POMDP solvers had numerical issues and were not able to solve the problems reliably. The set of 13 POMDP problems were selected from Cassandra's repository [6] such that they had more than 10 states. Our goal is to show that it is feasible to solve POMDP problems by encoding them into equivalent SSAT problems and finding optimal policies with SSAT solvers.

Our Prime solver is compared against two exact finite horizon POMDP solvers, Walraven [34] and Incremental Prune [5], and an exact SSAT solver ZANDER [22]. The results can be seen in Table 4 where the number of actions, states, and observations for each problem are listed. The H columns indicate the horizon reached within the 1000-second cutoff and the columns T indicate the time (seconds) needed for the horizon reported. An X indicates that the solver was not able to find an optimal policy for any horizon greater than 0. For each problem, the best score is bolded and the tally of best

				Walraven		PRIME		PRUNE		ZANDER	
problem	$\|A\|$	$\|S\|$	$\|O\|$	H	T	H	T	H	T	H	T
4x3.95	4	11	6	10	266.60	5	19.73	**10**	**161.79**	0	X
4x5x2.95	4	39	4	**20**	**127.58**	4	128.76	0	X	0	X
aloha.10	9	30	3	3	2.06	**4**	**198.73**	3	0.73	0	X
aloha.30	29	90	3	**2**	**21.88**	2	77.06	0	X	0	X
hallway	5	92	17	**3**	**41.93**	2	7.11	3	61.41	0	X
hallway2	5	92	17	2	0.08	2	22.60	**2**	**0.01**	0	X
learning.c2	8	12	3	2	0.53	**4**	**102.86**	2	0.19	0	X
learning.c3	12	24	3	2	7.19	**3**	**43.22**	2	29.23	0	X
learning.c4	16	48	3	2	115.15	**2**	**6.54**	1	0.00	0	X
milos-aaai97	6	20	8	2	1.33	**4**	**140.35**	2	0.38	0	X
query.s3	3	27	3	3	7.05	**4**	**133.64**	3	14.61	0	X
query.s4	4	81	3	2	0.39	**3**	**753.47**	2	0.10	0	X
tiger-grid	5	36	17	2	82.15	**3**	**281.13**	2	205.44	0	X

Table 4: Benchmark of POMDP problems encoded to SSAT problems by PRIME versus using native POMDP solvers Walraven and Prune.

scores for each solver was Walraven (3), Prime (8), Prune (2) and Zander (0). Overall, Prime was the better solver by finding an optimal policy for a longer horizon or within less time.

Zander was quite poor and this can be explained by the fact that features such as variable ordering do not help on POMDP problems since you are restricted by the natural quantifier ordering and pure literals can only be used on existential variables that correspond to actions in the encoding. Walraven and Prune performed similarly. They both incrementally add α-vectors to a solution set and prune dominated vectors. These methods perform well on problems where the optimal value function can be compactly represented by a small number of α-vectors. In contrast, Prime searches the set of policy trees directly in the encoded problem and its efficiency depends on the presence of symmetric subproblems that can be reused.

7 CONCLUSION

In summary, we showed how to encode SSAT problems as flat POMDPs and vice-versa without any exponential blow up. These constructive reductions permit for the first time an empirical comparison of exact SSAT and POMDP solvers on common benchmarks. We also proposed a new SSAT solver called Prime that incorporates watch literals, component decomposition and symmetry detection from the SAT and #SAT literature. SSAT-Prime is competitive in comparison to native POMDP solvers on POMDP benchmarks.

In the future, we would like to design a solver that leverages techniques from both SSAT and POMDP solvers. Exact POMDP solvers often focus on pruning α-vectors in order to obtain compact representations of optimal value functions. In contrast, SSAT solvers focus on pruning and decomposing policy trees to reduce the search time to compute an optimal policy. Since these techniques are orthogonal, it should be possible to combine them into a new technique with improved efficiency.

References

[1] Ignasi Abío and Peter J Stuckey. Encoding linear constraints into SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 75–91. Springer, 2014.

[2] Roberto J Bayardo Jr and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[3] Blai Bonet. Conformant plans and beyond: Principles and complexity. *Artificial Intelligence*, 174:245–269, 2010.

[4] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–204, 1994.

[5] Anthony Cassandra, Michael L Littman, and Nevin L Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 54–61. Morgan Kaufmann Publishers Inc., 1997.

[6] Anthony R. Cassandra. The POMDP page, 2003-2017. [Online; accessed 25-October-2017].

[7] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[8] Adnan Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

[9] C. P. Gomes, B. Selman, K. McAloon, and C. Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. *In 4th Int. Conf. Art. Intel. Planning Syst.*, pages 208–213, 1998.

[10] Marijn J. H. Heule, Matti Juhani Järvisalo, and Martin Suda. Proceedings of SAT competition 2018; solver and benchmark descriptions. In *SAT Competition 2018*. University of Helsinki, Department of Computer Science, 2018.

[11] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[12] Henry Kautz and Bart Selman. SATPLAN04: Planning as satisfiability. *Working Notes on the Fifth International Planning Competition (IPC-2006)*, pages 45–46, 2006.

[13] Matthew Kitching and Fahiem Bacchus. Symmetric component caching. In *IJCAI*, pages 118–124, 2007.

[14] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*, volume 2008. Zurich, Switzerland, 2008.

[15] Wee S Lee, Nan Rong, and Daniel J Hsu. What makes some POMDP problems easy to approximate? In *Advances in neural information processing systems*, pages 689–696, 2007.

[16] Michael L Littman. Probabilistic propositional planning: Representations and complexity. In *AAAI/IAAI*, pages 748–754, 1997.

[17] Michael L Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9(1):1–36, 1998.

[18] Michael L Littman, Stephen M Majercik, and Toniann Pitassi. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[19] Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. Nonapproximability results for partially observable Markov decision processes. *J. Artif. Intell. Res.(JAIR)*, 14:83–103, 2001.

[20] Stephen M. Majercik. APPSSAT: Approximate probabilistic planning using stochastic satisfiability. *International Journal of Approximate Reasoning*, 45(2):402–419, 2007.

[21] Stephen M Majercik and Byron Boots. DC-SSAT: a divide-and-conquer approach to solving stochastic satisfiability problems efficiently. In *Proceedings of the Naional Conference on Artificial Intelligence*, volume 20, page 416. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[22] Stephen M Majercik and Michael L Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1):119–162, 2003.

[23] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[24] Christos Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[25] Christos H Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[26] Pascal Poupart, Kee-Eung Kim, and Dongho Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

[27] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[28] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12):1031–1080, 2006.

[29] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state Markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952. ACM, 2006.

[30] Marc Toussaint, Amos Storkey, and Stefan Harmeling. Expectation-maximization methods for solving (PO)MDPs and optimal control problems. *Inference and Learning in Dynamic Models. Cambridge University Press: Cambridge, England*, 2010.

[31] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *European Workshop on Logics in Artificial Intelligence*. Springer Berlin Heidelberg, 2002.

[32] M. N. Velev and R. E. Bryant. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *Journal of Symbolic Computing*, 35(2):73–106, 2003.

[33] Nikos Vlassis, Michael L Littman, and David Barber. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Transactions on Computation Theory (TOCT)*, 4(4):12, 2012.

[34] Erwin Walraven and Matthijs TJ Spaan. Accelerated vector pruning for optimal POMDP solvers. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.