On the relationship between satisfiability and partially observable Markov decision processes

by

Ricardo Salmon

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

© Ricardo Salmon 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor(s):	Pascal Poupart Professor, Dept. of Computer Science, University of Waterloo
Internal Member:	Peter van Beek Professor, Dept. of Computer Science, University of Waterloo
	Jesse Hoey Professor, Dept. of Computer Science, University of Waterloo
Internal-External Member:	Mark Crowley Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
External Examiner:	Fangju Wang Professor, Dept. of Computer Science, University of Guelph

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Stochastic satisfiability (SSAT), Quantified Boolean Satisfiability (QBF) and decision-theoretic planning in finite horizon partially observable Markov decision processes (POMDPs) are all PSPACE-Complete problems. Since they are all complete for the same complexity class, I show how to convert them into one another in polynomial time and space. I discuss various properties of each encoding and how they get translated into equivalent constructs in the other encodings. An important lesson of these reductions is that the states in SSAT and flat POMDPs do not match. Therefore, comparing the scalability of satisfiability and flat POMDP solvers based on the size of the state spaces they can tackle is misleading.

A new SSAT solver called SSAT-Prime is proposed and implemented. It includes improvements to watch literals, component caching and detecting symmetries with upper and lower bounds under certain conditions. SSAT-Prime is compared against a state of the art solver for probabilistic inference and a native POMDP solver on challenging benchmarks.

Acknowledgements

I would like to personally thank my supervisor, Dr. Pascal Poupart, for his guidance and support else this thesis would not be possible. We had many intense meetings fleshing out ideas on the whiteboard that were instrumental in narrowing the scope of my work. Also, special recognition to my committee members Dr. Peter van Beek, Dr. Jesse Hoey, Dr. Mark Crowley, and Dr. Fangju Wang for their time.

I would also like to show appreciation to all the friends I have made during my time in Waterloo, including members of the Hopeless Experts soccer team and the Computer Science GSA. A special thank you to Cristina and Jimmy that have kept me grounded and motivated through the good and bad times. Thanks to my brother, Shane, my mother, Sandra, and close family for their support in my journey here.

Finally, I would like to thank the following organizations for providing financial support. These include the University of Waterloo, Cheriton School of Computer Science, Canadian Natural Sciences and Engineering Council (NSERC), Ontario Graduate Scholarship (OGS), Mitacs Accelerate, Kik Interactive, Hockeytech, and Vector Institute.

Dedication

I dedicate this dissertation to Wayne Erdman, my high school teacher, for his encouragement. He was a tremendous teacher and mentor who inspired my pursuit of Mathematics, Computer Science, and higher education.

Table of Contents

List of Figures

List of Tables

Chapter 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 [66]. State of the art solvers for flat POMDPs [43, 71] 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 [45]). 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 [53] 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 [4]. 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 as satisfiability [37, 39, 78, 77], 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 [50, 59]. In fact, SSAT is PSPACE-Complete [67], which means that SSAT and flat POMDPs can express the same space of planning problems. State of the art solvers such as Zander [59], DC-SSAT [57] and APPSAT [56] 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.

1.1 Contributions

I take a first step in this direction by describing constructive reductions from satisfiability problems (SAT, QBF and SSAT) to flat POMDP and from flat POMDP to SSAT. A full list of the encodings are below:

- Satisfiability ⇒ POMDP
    1. SAT ⇒ POMDP
    2. QBF ⇒ POMDP
    3. SSAT ⇒ POMDP
- POMDP ⇒ SSAT
- Probabilistic Inference ⇒ SSAT

The first half of this thesis is theoretical in nature. It opens the door to future cross-fertilization by explaining how POMDP solvers could be run on SSAT problems and how SSAT solvers could be run on flat POMDP problems. An important lesson of this work is to show that states in satisfiability problems (SAT, QBF and SSAT) do not correspond to states in flat POMDPs.

The reductions that I present 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. Hence, the common belief that satisfiability solvers scale much better than flat POMDP solvers based on a comparison of the size of their respective state spaces is erroneous.

In the second half, I build a solver, SSAT-Prime, for SSAT problems by extending techniques from various satisfiability solvers and generalizing them to the stochastic case:

- Watch Literals
- Component Decomposition
- Component Caching
- Symmetry Detection

- Upper Bounds

Furthermore, I verified the correctness of the encodings using my solver and tested the solver on encoded POMDP and Inference problems.

1.2 Outline

The thesis is structured as follows. Chapter 2 provides the necessary background on POMDPs, the different Satisfiability problems, and Probabilistic Inference that will be used throughout. Chapter 3 covers related works to encode restricted forms of POMDP as inference and model counting techniques for satisfiability. Later, I look at some of the problems and some available solvers. Chapter 4 describes how to encode satisfiability problems as POMDPs while Chapter 5 explains how to convert POMDP and Inference problems into SSAT and discusses the complexity of the reductions. Chapter 6 explains in detail the different advances of our SSAT solver and Chapter 7 shows some benchmark results using the encoding on some Inference, POMDP and random problems. Finally, Chapter 8 concludes and discusses future work.

Chapter 2

Background

In this chapter, we briefly review POMDPs, Boolean satisfiability and probabilistic inference. Boolean satisfiability solvers have improved tremendously in the last 50 years, which encouraged their use in numerous application domains including planning [37], scheduling [31] and hardware verification [92]. For many applications it is now reasonable to encode the original problem in SAT, find a solution, and decode the solution. Frequent SAT competitions have led to many clever improvements. In fact, modern SAT solvers are now able to solve instances with tens of thousands of variables and million of constraints.

2.1 POMDP

Partially Observable Markov Decision Processes (POMDPs) provide a principled mathematical framework for planning under uncertainty. Formally, it is specified by a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T, \Omega, R, b_0, \gamma, h)$, 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, $\gamma \in [0, 1]$ is the discount factor and h is the planning horizon. In this work, 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.

In each time step, the agent is in some state $s \in \mathcal{S}$ and takes an action $a \in \mathcal{A}$ that moves the agent to a new state s' according to the transition distribution $T(s', a, s) = \Pr(s'|s, a)$. Since the state of the agent is not directly observable, instead, it receives an observation $o \in \mathcal{O}$ according to the observation distribution $\Omega(o, s', a) = \Pr(o|s', a)$. The agent receives a reward $R(s, a)$ after executing action a in state s. The reward function

captures preferences over states and actions for the agent. The objective of the agent is to maximize the expected sum of rewards received by choosing good actions.

The discount factor γ can be seen as controlling the importance of rewards received in the future ($\gamma \approx 1$) versus rewards received immediately ($\gamma = 0$).

A POMDP solution is a policy. A policy, π, is usually a mapping from states to actions. However, since we are uncertain regarding the current state, we will use a sufficient statistic known as the belief [2]. A belief b is a distribution over states given all previous pairs of observations received and actions taken. An optimal policy, π^*, is a mapping from beliefs to actions that maximizes the long term rewards. We can define b_0 to be the starting distribution over states before any action is taken at time step $t = 0$.

Given an observation and action, we can derive the next belief state as in Eq. 2.1 by using the observation distribution and state transition distribution. In fact, the belief state is a sufficient statistic to derive optimal policies [2].

$$b_o^a(s') \propto \sum_s \Pr(s'|s,a) \Pr(o|s',a) b(s) \tag{2.1}$$

The value $V^\pi(b)$ of a policy π can be specified as the expected total future reward received by executing the policy starting from some belief state b.

2.1.1 Value Function

A policy, π, has a corresponding value function that is the expected sum of rewards starting from an initial belief b_0:

$$V^\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t E_\pi[R(b_t, \pi(b_t))], \ \forall b_0$$

We can rewrite the value function recursively as:

$$V^\pi(b) = R(b, \pi(b)) + \gamma \sum_o Pr(o|\pi(b), b) V^\pi(b_o^a) \ \forall b$$

where the value of starting in belief b and following policy π is the immediate reward of taking action $\pi(b)$ in b plus the sum of the value of all successor states weighted by their

likelihood. The probability of an observation given an action a and belief b is:

$$Pr(o|a,b) = \sum_{s'} \Omega(s',a,o) \sum_{s} T(s,a,s')b_s$$

An optimal value function, V^*, is one which entails the highest reward over all policies starting from some belief b and is unique [7] while satisfying Bellman's equation:

$$V^{\pi^*}(b) = \max_a R(b,a) + \gamma \sum_o Pr(o|a,b)V^{\pi^*}(b_o^a) \ \forall b$$

2.1.2 Value Iteration

It is not clear based on the form of the optimal value function how polices should be represented. One common approach is to use a policy tree where nodes are belief states, node labels are actions, edges are possible observations and the starting belief would be the root such that the value of policy tree p starting in belief b is

$$V^p(b) = \sum_{s \in \mathcal{S}} b(s)V^p(s)$$

A consequence of the policy tree representation is that value functions are linear with respect to beliefs. We show linearity by considering a base case of one node as the root (horizon = 1). It's value function will be $V^p_{t=1}(b) = \sum b(s)R(s,a(p))$. In the general case, if we have a linear value function at horizon h then a horizon $h+1$ value function will also be a linear combination of $|O|$ horizon h value functions:

$$V^p_{t=h+1}(b) = \sum_s b(s)[R(s,a(p)) + \gamma \sum_o Pr(o|s,a(p))V^{o(p)}_{t=h}(b_o^{a(p)})]$$

Let $\underline{\Gamma}$ be a set of policy trees each corresponding to a linear function in b. The optimal value function will be the upper surface of the linear functions in $\underline{\Gamma}$. Hence, the value function will be piecewise-linear and convex.

$$V_t(b) = \max_{p \in \underline{\Gamma}} b \cdot \alpha_p$$

where α_p corresponds to the linear value function of policy p and is a vector of dimension $|S|$.

The Value Iteration algorithm works by computing the optimal value functions for an increasing horizon t. Let's start with a default policy tree at $t = 0$ of value zero. Assume we're at time step t, to build a $t + 1$ policy we consider taking all actions and receiving each observation for all previous policy trees. The value is updated based on the immediate reward received.

Unfortunately, the number of policy trees grows exponentially. Therefore previous methods have tried to find novel and clever ways of pruning useless α-vectors. We say an α-vector is useful if it is optimal for at least some nonzero region in the belief space, otherwise it is useless and can be removed without affecting the value function. Related algorithms include Sondik's one-pass and two-pass algorithm [16], Cheng's linear support algorithm [19], Witness algorithm [47], and Zhang and Liu's incremental-pruning algorithm [98].

There are other techniques for solving POMDPs. In particular Policy Iteration [33] which optimizes a policy directly instead of a value function, forward search [83] which instead does a bounded online look-ahead while executing a policy and finite controllers [70].

2.1.3 Approximations

In the next sections, we consider some approximations based on value iteration that bound the value function from below and the fast informed bound that bounds the value function from above.

Point-based Value Iteration

A major contribution in solving POMDPs was the introduction of the Point-based Value Iteration (PBVI) algorithm [69, 71]. Point-based algorithms have been particularly successful in computing approximate solutions to large POMDPs that were considered intractable.

The key insight of PBVI was to bound the size of the value function by only considering α-vectors that are optimal at some belief points. It was first suggested to use the set of regular grid points as the belief set, but that led to optimizing belief points that might not be reachable from our initial belief. We maintain a set of beliefs that are reachable from the initial belief and only keep optimal α-vectors for each belief. Assuming we choose actions and observations appropriately for reachable beliefs, a compact update is:

$$\text{backup}(\underline{V}, b) = \underset{\alpha_b^a}{\text{argmax}}\, b \cdot \alpha_b^a \tag{2.2}$$

where

$$\alpha_b^a = r^a + \gamma \sum_o \underset{\alpha_o^a \in \underline{V}_o^a}{\operatorname{argmax}} b \cdot \alpha_o^a \ \forall a, b \tag{2.3}$$

$$\alpha_o^a = \gamma \sum_{s'} \Omega(s', a, o) T(s, a, s') \alpha_{s'}, \forall o, \forall \alpha \in \underline{V} \tag{2.4}$$

A generic PBVI algorithm can be found in Algorithm 1 and provides a lower bound estimate of the optimal value function.

Algorithm 1 Point-based Value Iteration

procedure PBVI($\mathcal{P}$)
  $\underline{\alpha}_0^a(s) \leftarrow \min_{s'} \frac{R(s',a)}{1-\gamma} \forall a, s$
  $\underline{\Gamma} = \{\underline{\alpha}_0^a | a \in \mathcal{A}\}$
  $B = \{b_0\}$
  repeat
    for each $b \in \mathcal{B}$ **do**
      $b' \leftarrow \operatorname{argmax}_{a,o} |b_o^a - \mathcal{B}|$
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{b'\}$
    end for
    $\Gamma' \leftarrow \{\}$
    for each $b \in \mathcal{B}$ **do**
      $\alpha_o^a = \gamma \sum_{s'} \Omega(s', a, o) T(s, a, s') \alpha_{s'}, \forall o, \forall \alpha \in \underline{\Gamma}$
      $\alpha_b^a = r^a + \gamma \sum_o \operatorname{argmax}_{\alpha_o^a \in \underline{\Gamma}_o^a} b \cdot \alpha_o^a$
      $\alpha_b \leftarrow \operatorname{argmax}_{\alpha_b^a} b \cdot \alpha_b^a$
      $\Gamma' \leftarrow \Gamma' \cup \{\alpha_b\}$
    end for
    $\underline{\Gamma} \leftarrow \Gamma'$
  until convergence
  return Γ'
end procedure

Observable MDP

One of the simplest approaches to approximate a POMDP is to consider the MDP model [52]. A solution to the MDP will give you the value of corners of the simplex in the original

POMDP space and interior beliefs can be represented as linear combinations of the corners.

$$\overline{V} = \sum_{s \in \mathcal{S}} b(s) V^*_{MDP(s)}$$

In fact, the MDP solution bounds the POMDP value function from above. It makes sense by noting that we are assuming the MDP is a fully observable version of the POMDP. This corresponds to gaining additional information. In general, you should have higher rewards with more information. An advantage of the approximation is the speed to calculate and the fact that it can be used in more complex approaches.

Fast Informed Bound

A draw back of the MDP approximation is the assumption of full observability. By incorporating partial observability to some degree in the update rule, we obtain the fast informed bound (FIB) [34].

We can derive the FIB from the exact update rule by moving the sum over states outside the max of Bellman's equation. Note that we will be maximizing the α-vectors for each state in contrast to finding a state that maximizes the α-vectors.

$$\overline{V}_{t+1}(b) = \max_{a \in \mathcal{A}} \{ \sum_{s \in \mathcal{S}} b(s) R(s,a) + \gamma \sum_{o \in \mathcal{O}} \max_{\alpha_i \in \overline{\Gamma}_i} \sum_{s' \in \mathcal{S}} \sum_{s \in \mathcal{S}} Pr(s', o|a, s) b(s) \alpha_i(s') \} \tag{2.5}$$

$$\leq \max_{a \in \mathcal{A}} \{ \sum_{s \in \mathcal{S}} b(s) R(s,a) + \gamma \sum_{o \in \mathcal{O}} \sum_{s \in \mathcal{S}} \max_{\alpha_i \in \overline{\Gamma}_i} \sum_{s' \in \mathcal{S}} Pr(s', o|a, s) b(s) \alpha_i(s') \} \tag{2.6}$$

$$= \max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b(s) \{ R(s,a) + \gamma \sum_{o \in \mathcal{O}} \max_{\alpha_i \in \overline{\Gamma}_i} \sum_{s' \in \mathcal{S}} Pr(s', o|a, s) b(s) \alpha_i(s') \} \tag{2.7}$$

As with the MDP approximation, the value function is piecewise linear and convex with $|A|$ α-vectors (one for each action). The time complexity of the FIB update is $O(|\mathcal{A}||\mathcal{S}|^2|\mathcal{O}||\overline{\Gamma}_t|)$ up to time horizon t. An implementation is shown in Algorithm 2 based on [71].

Algorithm 2 Fast Informed Bound algorithm

procedure FIB($\mathcal{P}$)
  $\alpha_0^a(s) \leftarrow \max_{s'} \frac{R(s',a)}{1-\gamma} \forall a, s$
  repeat
    $\alpha_{t+1}^a \leftarrow R(s,a) + \gamma \sum_{o \in \mathcal{O}} \max_{\alpha_i^a \in \overline{\Gamma}_i} \sum_{s' \in \mathcal{S}} Pr(s', o|a, s) \alpha_i^a(s')$
  until convergence
  return α^a, $\forall a \in \mathcal{A}$
end procedure

2.2 Satisfiability

2.2.1 Boolean Satisfiability

The Boolean satisfiability problem or SAT is to determine if it is possible to find a joint variable assignment to a Boolean formula that evaluates to *true*. Consider a formula F and a set of Boolean variables X. An example formula is shown below:

$$F = (x_3 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_5) \tag{2.8}$$

SAT is the first known NP-complete problem [20]. A sub-problem of SAT is restricting the formulas to be in conjunctive normal form (CNF). The CNF representation is a conjunction of clauses where a clause is a disjunction of literals and a literal is a variable or its negation. Without loss of generality, it is sufficient to consider formulas composed of only AND, OR, NOT operators. Furthermore, if we limit the number of literals in the disjunction to three this gives us 3SAT and an example is shown in Eq 2.8 for F. Unfortunately, even 3SAT is in the same complexity class as SAT (in fact, any SAT problem can be converted in polynomial time and space to 3SAT). In the next sections, we cover different approaches to solving SAT.

SAT solvers can be classified into complete methods and incomplete methods. Given a formula F, the complete methods, as their name implies, will return a satisfying set of variable assignments or will prove that the formula F is unsatisfiable. Most complete methods are based on the popular Davis-Putnam-Logemann-Loveland (DPLL) algorithm [27, 26]. The DPLL algorithm is an exhaustive branching procedure that is able to efficiently prune parts of the search space when clauses become unsatisfiable.

In contrast, incomplete methods often do a stochastic local search and provide no guarantee. Either a solution is returned or a time limit is reached and the process terminates.

In spite of the short comings, these methods are able to scale to much larger problems than complete methods and have been the driving force for large problems. Two key methods that enjoy success in the literature are GSAT [82] and Walksat [61]. In the remaining chapters we will focus on complete methods and in particular DPLL inspired techniques.

The key features of modern DPLL solvers are efficient unit propagation, clause learning, non-chronological back-jumping, variable/value selection heuristic and randomized restarts.

Unit Propagation

In SAT, a unit clause is where there is an unsatisfied clause with one unassigned literal. The unit rule says that since we are trying to satisfy all clauses, it is valid to assume such a clause will be satisfied and to make the necessary assignment to satisfy the clause. All clauses have to be satisfied and therefore it should not hurt to work under this assumption. The act of performing such assignments might lead to further unit clauses and propagating all such assignments is unit propagation. If the assumption was wrong, then a conflict will eventually occur and we will have to backtrack.

Performing unit propagation efficiently is a challenging task. A basic method is to track the number of unassigned literals in each clause and when it is one, we know it is a unit clause. Although, this technique and other similar techniques solve the problem, they tend to be too costly on larger problems, especially when the number of comparisons needed grows and there is extra work to backtrack. For most SAT problems, the solver spends up to 90% of the time on unit propagation. It was not until the 2-scheme watch literals was introduced in the solver zChaff [64] that a substantial breakthrough was made.

Clause Learning

The clause learning approach can be argued as the most important technique in modern DPLL solvers that gets us exponential speedups. When a conflict is encountered we concisely learn a new clause that encodes the reason for the conflict and as a result, future branches in the search space that include the original conflict can be avoided earlier.

The outline of a DPLL implementation is shown in Algorithm 3. Given a formula, F, we first check if the set of literals can be reduced to $true$ or $false$, otherwise we perform unit propagation and assign pure literals. Pure literals are literals in F that only take one form. A pure literal can easily be removed by setting it to a value that satisfies all the clauses that contain it. Next, a new literal is chosen and we recurse on both possible values and return its disjunction.

Algorithm 3 DPLL algorithm

```
1:  procedure DPLL(F)
2:      if F is consistent set of literals then
3:          return true
4:      end if
5:      if F contains an empty clause then
6:          return false
7:      end if
8:      for all unit clause l in F do
9:          F ← unit-propagation(l, F)
10:     end for
11:     for all literal l that occurs pure in F do
12:         F ← pure-literal-assign(l, F)
13:     end for
14:     l ← choose-literal(F)
15:     return DPLL(F ∧ l) ∨ DPLL(F ∧ ¬l)
16: end procedure
```

2.2.2 Quantified Boolean Formula

A general extension of SAT is the satisfiability of quantified Boolean formulas (QBF) where Universal quantifiers are introduced over variables. Each variable has a domain consisting of the set $\{true, false\}$. The universal and existential quantifiers are defined below:

$$\exists x \; F(x) = F(x = true) \vee F(x = false)$$

$$\forall x \; F(x) = F(x = true) \wedge F(x = false)$$

where F is a Boolean formula with a free variable x. It is also the case that all quantified Boolean formulas can be written in prenex normal form as below:

$$Q_1 X_1 Q_2 X_2 ... Q_D X_{|X|} \;\; F(X)$$

where $X_i \subseteq X$, $X = \bigcup_i X_i$ and all variables in the set X_i have the same quantifier type. In particular, the index i of each quantifier set corresponds to the quantifier level. The quantifier level restricts a variable in a higher level to be assigned before a variable in a lower level and the order of variables in the same level are interchangeable.

QBF is PSPACE-complete, but it shares many similarities with SAT and it is usually the case that similar techniques are used to tackle both types of satisfiability problems, including clause learning, backtracking and unit propagation.

2.2.3 Stochastic Satisfiability

Stochastic satisfiability (SSAT) was first proposed by [67] as a generalization of Boolean satisfiability where each variable x_i is either existentially quantified $\exists$ or randomly quantified $\mathbb{R}$ as below:

$$\exists x_1 \mathbb{R} x_2 \exists x_3 ... \mathbb{R} x_n F(x_1, x_2, ..., x_n) \tag{2.9}$$

Stochastic Satisfiability is closely related to quantified Boolean satisfiability (QBF) in the sense that both are PSPACE-complete, but they differ in the choice of quantifiers since randomization quantifiers are used in SSAT, while universal quantifiers are used in QBF. In SSAT, the goal is to find an assignment of values to the existentially quantified variables that maximizes the probability that a Boolean formula $F(x_1, x_2, ..., x_n)$ is satisfied.

The probability of satisfiability for a true formula is 1 and in the case of unsatisfiable formula evaluates to 0. Therefore the computation for each quantifier type is now:

$$\exists x \ F(x) = \max\left(F(x = true), F(x = false)\right)$$

$$\mathbb{R} x \ F(x) = \Pr(x = true) F(x = true) + \Pr(x = false) F(x = false)$$

The probability depends on the distribution of the randomized variables. In this thesis, I consider discrete variables x_i that can take two or more values. Hence, the SSAT problem in Eq. 2.9 corresponds to the following optimization problem:

$$\max_{x_1} \sum_{x_2} \Pr(x_2) \max_{x_3} ... \sum_{x_n} \Pr(x_n) \delta(F(x_1, x_2, ..., x_n)) \tag{2.10}$$

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

SSAT can be used to encode planning problems with uncertainty. The existentially quantified variables correspond to actions while the randomized variables correspond to uncertain environmental variables whose values are decided by nature. The formula F encodes the goal of the planning problem and the dynamics of the environment, including action effects [59]. The marginal distributions for each randomized variable quantify the uncertainty. SSAT solvers find values (corresponding to the actions) for the existential

variables that maximize the probability of reaching the goal while respecting the dynamics of the environment by satisfying formula F.

2.3 Probabilistic Inference

Probability theory was originally developed to analyze games, but it has since evolved and it is now grounded in a rigorous set of axioms. Probabilistic models are the corner stone of many decision making systems across a variety of fields and inference allows us to quantify the uncertainty of these models in a principled way. In the next section, I discuss Bayesian networks and the different inference problems. The scope of my work is limited to discrete variables.

2.3.1 Bayesian Network

A Bayesian Network (BN) [42] is a graphical model that captures probabilistic relations between a set of random variables. The representation used is a directed acyclic graph (DAG) such that each node is a random variable and an edge implies a conditional dependence between variables while the lack of an edge implies conditional independence. In this model, a random variable can be observed or latent (i.e., unobserved or hidden).

Let $X = \{x_1, ..., x_{|X|}\}$ represent a finite set of random variables and E is the set of directed edges in the Bayesian network. A random variable x_j is a parent of x_i if $(x_i, x_j) \in E$ and all the parents of x_i are $\text{parents}(x_i) = \{x_j | (x_i, x_j) \in E\}$. An advantage of using a BN is that the joint distribution for the set of random variables X can be factorized as a product of conditional distributions:

$$\Pr(X) = \prod_{x_i \in X} \Pr(x_i | \text{ parents}(x_i))$$

The conditional distributions can be represented as a table for discrete random variables. A Bayesian Network is a general representation that many approaches use to compactly represent multivariate distributions, express conditional independences and facilitate inference.

2.3.2 Inference Problems

There are usually three problems of interest when using Bayesian Networks.

- Infer the value(s) of some variable(s) given some evidence.
- Learn the parameters of the conditional distributions in the network.
- Identify the most likely structure of the network based on some data.

Our focus will be on the first kind of problem: inference. It is known that inference is in PP [15, 44]. In probabilistic inference we are interested in answers to queries of the form:

$$\Pr(\theta|y) = \sum_{h \in H} \Pr(\theta, h|y)$$

where a set of variables H has been marginalized.

Complexity classes are a way to capture the difficulty of a set of problems in a general way. They are usually defined with respect to a computational model that is able to recognize such a language. In particular, the complexity classes that interest us can be described by some restricted Turing machines.

Gill [30] defined the class Probabilistic Polynomial time, PP, as the decision problems solvable by probabilistic Turing machines where:

- more than $\frac{1}{2}$ of computation paths accept if the answer is *yes*.
- at most $\frac{1}{2}$ of computation paths accept if the answer is *no*.

2.3.3 Maximum a Posteriori (MAP)

Another problem related to inference is Maximum a Posteriori (MAP) estimation. This type of problem usually comes from attempting to find an optimal point estimate, the mode, of some parameter given it's posterior distribution. In statistics the mode is the most frequent value in the data. In fact, if we consider a likelihood function or conditional distribution, $\Pr(y|\theta)$, for y given a parameter θ and a prior, $\Pr(\theta)$, on θ, we can compute the posterior distribution for θ given y according to Bayes theorem:

$$\Pr(\theta|y) \propto \Pr(y|\theta)Pr(\theta)$$

Normally, we are interested in a point estimate. Therefore the MAP problem is one of maximization:

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \Pr(\theta|x)$$

We can also relate the MAP solution to the Maximum Likelihood solution, θ_{ML}, by noting that MAP is a regularized version of ML. This is of interest to us because MAP is in the complexity class NP which is related to PP by NP $\subseteq$ PP.

2.3.4 Marginal MAP

Finally, marginal MAP is MAP where in addition we need to marginalize the hidden variables in the set H.

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \sum_{h \in H} \Pr(\theta | h, y)$$

We note that marginal MAP is in the complexity class NP^{PP} [68] where NP $\subseteq$ PP $\subseteq$ NP^{PP}.

2.3.5 Inference Algorithms

In the next sections, we will review current approaches for performing inference on a Bayesian Network. We will start by describing Enumeration and Variable Elimination for exact inference. In the next section, we review approximate techniques that are stochastic and in particular Gibbs sampling. Finally, we will review approximate methods that are deterministic based on Variational Bayes.

Exact Inference

The naive approach, inference by enumeration, uses a table representation of the full joint distribution. Therefore, for each row there is a corresponding combination of variable assignments and inference would amount to summing the proportion of assignments consistent with the query. Variable Elimination is a dynamic programming approach that improves upon Enumeration by caching repeated computations in sub-expressions for reuse and avoiding irrelevant variables. The key idea is to push sums further inside the joint equation whenever possible. Unfortunately, the worst time and space complexity of Enumeration and Variable Elimination is exponential in the number of variables [79]. Inference techniques based on weighted model counting are among the best exact inference techniques since they can exploit several types of structure including context specific independence and sparsity [46].

Approximate Inference - Sampling

In many large real-world problems, inference becomes intractable and approximate methods are sought after. Approximate methods achieve a compromise by trading off computation time for accuracy. These approximate techniques are ideal when the network is too large for exact methods.

Monte Carlo (MC) methods [65] are a class of stochastic algorithms that allows one to estimate key statistics from a multivariate distribution by gathering samplings from the joint distribution. In particular, we discuss Gibbs sampling [29, 51] that is a special case of MC methods.

Gibbs sampling allows us to sample from a multivariate distribution by considering the conditional distribution of each variable given the rest. Given a distribution of random variables $X = \{x_1, ..., x_{|X|}\}$ that forms a joint distribution $\Pr(x_1, ..., x_{|X|})$, we gather N samples as follows:

- arbitrarily set initial values $V^{(0)} = \langle v_1^{(0)}, ..., v_{|X|}^{(0)} \rangle$
- for sample $n = \{1, ..., N\}$
  - for each variable $x_i \in \{x_1, ..., x_{|X|}\}$
    - sample x_i from conditional distribution: $\Pr(x_i | x_1 = v_1^{(n)}, ..., x_{i-1} = v_{i-1}^{(n)}, x_{i+1} = v_{i+1}^{(n-1)}, ..., x_{|X|} = v_{|X|}^{(n-1)})$
    - $V^{(n)} = \langle v_1^{(n)}, ..., v_{i-1}^{(n)}, v_i^{(n)}, v_{i+1}^{(n-1)}, ..., v_{|X|}^{(n-1)} \rangle$
  - record new sample: $V^{(n)} = \langle v_1^{(n)}, ..., v_{|X|}^{(n)} \rangle$

Approximate Inference - Variational Bayes

Variational Bayes (VB) [54, 11] algorithms are a family of approximate inference methods that aim to find an analytical form of the posterior distribution of θ given data y. The motivation for VB is a method that is deterministic in contrast to MC methods.

In Variational Bayes, we are interested in the posterior distribution, $\Pr(\theta|y)$ over a set of hidden or latent variables $\theta = \langle \theta_1, ..., \theta_{|\theta|} \rangle$ given some data y. A distribution $Q(\theta)$, called the variational distribution is proposed as an approximation to $\Pr(\theta|y)$ such that $Q(\theta)$ has a simpler form (i.e., analytically tractable) than $\Pr(\theta|y)$ and expressive enough to be similar to the true posterior. The concept of similarity can be captured by a distance function,

in particular we minimize the Kullback-Leibler (KL) divergence or relative entropy. The KL-divergence is:

$$KL(Q(\theta)||Pr(\theta|y)) = \sum_{\theta} Q(\theta) \log \frac{Q(\theta)}{Pr(\theta|y)}$$

It turns out that minimizing the reverse form of the KL-divergence, $KL(Pr(\theta|y)||Q(\theta))$, leads to another algorithm Expectation Propagation [63]. Variational Bayes is often faster than MC methods for comparable accuracy, however the draw back is that deriving the equations can be tedious and requires lots of work even for modestly complex models.

2.3.6 Discussion

While tremendous progress has been made in recent years to develop approximate inference techniques that scale to large problems. Most of the current methods usually provide no performance guarantees on the quality of the solution for queries. Variational Bayes techniques typically improve a lower bound estimate, but are prone to getting stuck in local optima. They can return answers that are arbitrarily far from the correct one and it doesn't matter whether we give them more time, they will remain stuck. While there are variational methods that provide an upper bound [94], the solution quality is not competitive. The best methods are by far those which do not provide any guarantees.

Similarly, Monte Carlo techniques may get stuck in a mode for a while and even though they will converge in the limit, it is never clear when a run has converged. As a result, existing approximate inference techniques are often sufficient for research purposes, but their lack of performance guarantees make them poor candidates for industrial grade tasks. Stochastic methods can be used to derive confidence bounds on key statistics, but are usually probabilistic in nature unless hard assumptions are made on the distribution and the form of the latent variables.

There is usually a time consuming and error prone manual step to derive the necessary equations for variational techniques. MC methods are usually easier to use, but in practice there are many fine tunings required to get good results such as how to detect that the chain has converged, how many samples should be used for burn-ins, and the strength of auto-correlation between samples. To overcome these issues we propose an anytime algorithm that comes with performance guarantees in the form of lower and upper bounds while being easier to use.

2.4 Summary

In this chapter we reviewed three research areas, which are POMDP, Satisfiability and Inference. In POMDPs, computations based on Value Iteration are usually intractable. Therefore research is now focusing on approximation methods. The most common algorithm, Point Based Value Iteration, is an approximation that refines a lower bound to the optimal value function. In contrast, computing the upper bound based on the Fast Informed Bound ties in with the value of information and MDPs. The tractability limit of exact flat POMDP solvers is usually around tens to thousands of states.

Boolean satisfiability is a well studied problem that has matured with numerous solution methods. In particular, exact solvers can often solve problems on the order of hundreds of thousands of variables and millions of clauses. In QBF problems, we see a drop to tens of thousands of variables and hundreds to thousands for SSAT. Empirically, SSAT problems seem to be more difficult on average and this might be related to the fact that there are no equivalent rules to reduce randomly quantified variables.

Chapter 3

Related Work

In this chapter we cover related approaches for encoding POMDPs to inference, inference into satisfiability models such as stochastic SAT and weighting model counting of #SAT, and various probabilistic planning solvers.

3.1 POMDP Encoding

In the POMDP literature, several approaches have been proposed to optimize POMDP policies by probabilistic inference [88, 89] and they have been expanded to continuous [35] and hierarchical [87] domains. It was shown by [89] how to encode a POMDP into a mixture of dynamic Bayesian network using Expectation-Maximization for inference. The optimal policy is derived by transforming the problem from maximizing expected future rewards into likelihood maximization using mixture models. Later, [40] reduced the encoding to a single Dynamic Bayesian Network from a mixture.

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 $\text{NP}^{\#P}$ for marginal-MAP inference). For instance, [93] explained 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.

There has been related work by [90] in reducing conditional planning of polynomial

length to QBF and [12] for partially observable plans which are linear. [14] showed that 3SAT and hence n-SAT can be polynomially reduced to PLANSAT.

3.2 Model Counting

Model counting [32] or #SAT is an extension of SAT that asks how many variable assignments satisfy the formula. In contrast, SAT is a decision problem where as #SAT is a counting problem in the class #P-complete. Surprisingly, model counting is hard even for [91] 2SAT! Exact solvers are able to handle hundreds of variables while approximate solvers can handle up to thousands of variables [81]. If efficient solution algorithms can be derived, the main applications would be contingency planning and probabilistic reasoning. The focus of this section will be on mapping probabilistic inference to weighted model counting using exact methods.

Exact model counting methods are usually based on systematic DPLL inspired search or knowledge compilation [18]. Solvers based on searching usually use techniques from the SAT literature for pruning and to more efficiently explore the search space. In contrast, for knowledge compilation, the CNF formula is usually converted into another logical form that allows solutions to be counted in polynomial time. The most common of these forms include binary decision diagram (BDD) and deterministic decomposable negation normal form (d-DNNF). Some exact model counters are Relsat [5], CDP [10], Cachet [80], sharpSAT [86], c2d [25], and more recently ACE [1].

To solve basic model counting using search, explore all branches and whenever a satisfiable partial assignment is encountered that assigns m of n total variables we deduce that there are 2^{n-m} solutions by exhaustively enumerating the remaining $n-m$ variables. Normally, DPLL solvers are not able to detect when a solution is found except when all the variables are assigned. But the solver can be augmented with an extra data-structure to track when all clauses have been satisfied.

3.2.1 Encoding

In weighted model counting, we review a way to encode random variables and their conditional probability tables into a CNF. First, assume a Bayesian network contains a set of n variables X such that $X_i \in X$ with finite domain $X_i = \{0, 1, ..., |X_i|\}$. See Section

[1] http://reasoning.cs.ucla.edu/ace

2.3.1 for more information on inference in Bayesian networks. The joint distribution can be represented as a product of conditional distributions:

$$\Pr(x_1, x_2, ..., x_{|X|}) = \prod_n \Pr(x_n | parent(x_n)) \tag{3.1}$$

where $parent(x_n)$ is the set of assigned variables that are parents of x_n.

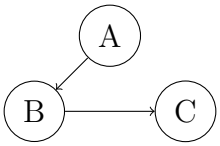


Figure 3.1: A 3-variable Bayesian network.

Table 3.1: Conditional distribution for random variables A, B, and C from Fig 3.1

a	$\Pr(A)$
0	0.4
1	0.6

a	b	$\Pr(B\|A)$
0	0	0.2
0	1	0.8
1	0	0.7
1	1	0.3

b	c	$\Pr(C\|B)$
0	0	0.0
0	1	0.0
0	2	1.0
1	0	0.2
1	1	0.6
1	2	0.2

Going forward, we refer to Figure 3.1 as an example Bayesian network with 3 variables. In WMC, given a theory of propositional logic one needs to assign a weight $W(l)$ to each literal l such that an assignment to all the variables corresponds to a model m that has weight:

$$W(m) = \prod_{l \in m} W(l) \tag{3.2}$$

and is proportional to the probability $\Pr(m)$.

Based on the work of [24], to transform a Bayesian network into CNF requires clauses (1) encoding the variable values and (2) the parameters of the conditional distribution. Let β represent indicator variables for each variable in the Bayesian network and λ be

parameters for each probability value in the CPTs. Each variable x that has domain values $\{0, 1, ..., |x| - 1\}$ is represented by the following clauses:

$$\bigvee_{v=0}^{|x|-1} \beta_{x_v}, \qquad \beta_{x_v} \Rightarrow \neg\beta_{x_{v'}}, v \neq v' \tag{3.3}$$

the parameters of the distribution $\Pr(x|y_1, ..., y_m)$ induce the following clauses:

$$\beta_x \wedge \beta_{y_1} \wedge ... \wedge \beta_{y_m} \iff \lambda_{x|y_1,...,y_m} \tag{3.4}$$

where in Eq. (3.3) it is guaranteed that at least one of the values in the domain of x is true and in the second term at most one literal in each clause is true. Taken together, exactly one value will be active. In Eq. (3.4), if the indicators for each variable are active, then the corresponding parameter must be active. Since the relation is an equivalence the reverse implication is also true. That is, if a parameter with certain indicator variables is true, then each individual variable indicator must be true.

The network in Fig 3.1 can be encoded as a CNF formula as shown below. First, according to rule (3.3), we can encode the indicator variables into the following clauses:

$$\beta_{a_0} \vee \beta_{a_1}, \neg\beta_{a_0} \vee \neg\beta_{a_1} \tag{3.5}$$

$$\beta_{b_0} \vee \beta_{b_1}, \neg\beta_{b_0} \vee \neg\beta_{b_1} \tag{3.6}$$

$$\beta_{c_0} \vee \beta_{c_1} \vee \beta_{c_2}, \neg\beta_{c_0}\neg \vee \beta_{c_1}, \neg\beta_{c_0} \vee \neg\beta_{c_2} \tag{3.7}$$

and the parameters generate the clauses in Table 3.2:

The weights for the indicator literals (both positive and negated) and the negated literal weights of all the parameters are always 1. The remaining weights are shown below in Table 3.3.

3.2.2 Local Structure

A consequence of doing the transformation to CNF is the exploitation of structure not present in the Bayesian networks such as context-specific independence (CSI). This includes determinism, equal parameters, and evidence.

In the CNF representation, determinism can be exploited by noting that whenever a parameter, $\lambda_{x|y_1,...,y_m}$, is 1, this implies that any clause that contains such a variable can be removed. Furthermore, the other parameters from the same CPT will be 0 since

Table 3.2: Parameter generated clauses for encoding Fig 3.1.

$$\beta_{a_0} \Rightarrow \lambda_{a_0}$$
$$\lambda_{a_0} \Rightarrow \beta_{a_0}$$
$$\beta_{a_1} \Rightarrow \lambda_{a_1}$$
$$\lambda_{a_1} \Rightarrow \beta_{a_1}$$

$$\beta_{c_1} \wedge \beta_{b_1} \Rightarrow \lambda_{c_1|b_1}$$
$$\lambda_{c_1|b_1} \Rightarrow \beta_{c_1}$$
$$\lambda_{c_1|b_1} \Rightarrow \beta_{b_1}$$
$$\beta_{c_2} \wedge \beta_{b_1} \Rightarrow \lambda_{c_2|b_1}$$
$$\lambda_{c_2|b_1} \Rightarrow \beta_{c_2}$$
$$\lambda_{c_2|b_1} \Rightarrow \beta_{b_1}$$

$$\beta_{b_0} \wedge \beta_{a_0} \Rightarrow \lambda_{b_0|a_0}$$
$$\lambda_{b_0|a_0} \Rightarrow \beta_{b_0}$$
$$\lambda_{b_0|a_0} \Rightarrow \beta_{a_0}$$
$$\beta_{b_1} \wedge \beta_{a_0} \Rightarrow \lambda_{b_1|a_0}$$
$$\lambda_{b_1|a_0} \Rightarrow \beta_{b_1}$$
$$\lambda_{b_1|a_0} \Rightarrow \beta_{a_0}$$
$$\beta_{b_0} \wedge \beta_{a_1} \Rightarrow \lambda_{b_0|a_1}$$
$$\lambda_{b_0|a_1} \Rightarrow \beta_{b_0}$$
$$\lambda_{b_0|a_1} \Rightarrow \beta_{a_1}$$
$$\beta_{b_1} \wedge \beta_{a_1} \Rightarrow \lambda_{b_1|a_1}$$
$$\lambda_{b_1|a_1} \Rightarrow \beta_{b_1}$$
$$\lambda_{b_1|a_1} \Rightarrow \beta_{a_1}$$

$$\beta_{c_0} \wedge \beta_{b_0} \Rightarrow \lambda_{c_0|b_0}$$
$$\lambda_{c_0|b_0} \Rightarrow \beta_{c_0}$$
$$\lambda_{c_0|b_0} \Rightarrow \beta_{b_0}$$
$$\beta_{c_1} \wedge \beta_{b_0} \Rightarrow \lambda_{c_1|b_0}$$
$$\lambda_{c_1|b_0} \Rightarrow \beta_{c_1}$$
$$\lambda_{c_1|b_0} \Rightarrow \beta_{b_0}$$
$$\beta_{c_2} \wedge \beta_{b_0} \Rightarrow \lambda_{c_2|b_0}$$
$$\lambda_{c_2|b_0} \Rightarrow \beta_{c_2}$$
$$\lambda_{c_2|b_0} \Rightarrow \beta_{b_0}$$
$$\beta_{c_0} \wedge \beta_{b_1} \Rightarrow \lambda_{c_0|b_1}$$
$$\lambda_{c_0|b_1} \Rightarrow \beta_{c_0}$$
$$\lambda_{c_0|b_1} \Rightarrow \beta_{b_1}$$

Table 3.3: Weights for the CNF encoding of Fig 3.1.

$$W(\lambda_{a_0}) = 0.4$$
$$W(\lambda_{a_1}) = 0.6$$
$$W(\lambda_{b_0|a_0}) = 0.2$$
$$W(\lambda_{b_1|a_0}) = 0.8$$

$$W(\lambda_{b_0|a_1}) = 0.7$$
$$W(\lambda_{b_1|a_1}) = 0.3$$
$$W(\lambda_{c_0|b_0}) = 0.0$$
$$W(\lambda_{c_1|b_0}) = 0.0$$

$$W(\lambda_{c_2|b_0}) = 1.0$$
$$W(\lambda_{c_0|b_1}) = 0.2$$
$$W(\lambda_{c_1|b_1}) = 0.6$$
$$W(\lambda_{c_2|b_1}) = 0.2$$

probabilities add up to 1 and these parameters can be removed from any clause they are included in. This will have no effect on the number of solutions, but this will reduce the size of the CNF and the search space the solver has to explore. In fact, many networks in the industry are highly deterministic and would greatly benefit.

Equal parameters occur when many parameters in the same CPT have equal values. For instance, in Table 3.1, the CPT for variable C has a repeated value of 0.2 for $\lambda_{c_0|b_1}$ and $\lambda_{c_2|b_1}$. We can merge the variables $\lambda_{c_0|b_1}$ and $\lambda_{c_2|b_1}$ into a single new variable $\lambda_{c_{0,2}|b_1}$ since only one parameter variable can be active in a particular CPT. If we replace the clauses that include $\lambda_{c_0|b_1}$ and $\lambda_{c_2|b_1}$, we obtain:

$$\begin{aligned} \beta_{c_0} \wedge \beta_{b_1} &\iff \lambda_{c_{0,2}|b_1} \\ \beta_{c_2} \wedge \beta_{b_1} &\iff \lambda_{c_{0,2}|b_1} \end{aligned} \tag{3.8}$$

Unfortunately, when the equivalence is expanded in both directions these lead to inconsistent indicator variables being implied by the merged parameter variable as shown below for c_0 and c_2:

$$\beta_{c_0} \wedge \beta_{b_1} \Rightarrow \lambda_{c_{0,2}|b_1}, \lambda_{c_{0,2}|b_1} \Rightarrow \beta_{c_0}, \lambda_{c_{0,2}|b_1} \Rightarrow \beta_{b_1} \tag{3.9}$$

$$\beta_{c_2} \wedge \beta_{b_1} \Rightarrow \lambda_{c_{0,2}|b_1}, \lambda_{c_{0,2}|b_1} \Rightarrow \beta_{c_2}, \lambda_{c_{0,2}|b_1} \Rightarrow \beta_{b_1} \tag{3.10}$$

If the equivalence is replaced with implication in (3.8), that will remove conflicting assignments, but increases the number of models that are consistent with the original joint assignment. It turns out that these extra models all assign more variables to true than in the original joint distribution. A process call minimization can be used to filter out the extra models by placing an upper limit on the number of variables that are assigned the value true.

Evidence is usually in the form of a conjunction of variables assigned over a disjunction of values. Therefore, evidence can be incorporated into the CNF encoding without extra complications as additional clauses. Directly encoding evidence into the WMC encoding usually reduces significantly the original problem space since many variables are assigned fixed values. Evidence often makes tractable some problems that would otherwise be intractable.

The trade off for solvers that compile to a different logical form would be that the transformation is query specific and it would need to be recomputed for new evidence. However, queries such as finding the marginal of each variable conditioned on evidence would still be computationally efficient.

There are many more improvements in the literature that is beneficial to a #SAT solver that may require modification of the solver (eclause [18]) or is domain specific (noisy-or/max relations [96]) that we do not consider here. An eclause is a clause with the additional constraint that exactly one of its literal must be satisfied. This can be used to enforce finite domain values over multiple boolean variables. See [18, 46, 9] for further readings.

3.3 Probabilistic Planning Solvers

Probabilistic planning is defined by a set of states, actions, initial state and goal states. Here, actions map states to next states stochastically. The solution is a sequence of actions from the initial state that is able to reach a goal state with high probability. For instance, in propositional planning, states are described by the joint assignment of propositional variables.

In the case of deterministic planning, the solver SATPLAN [38] was able to encode planning problems into satisfiability and the solution was decoded back to the original planning domain. This proved to be successful and analogous to other problems that have a reduction to SAT. Intuitively, probabilistic planning, corresponding to the SAT extension called E-MAJSAT, is at least as hard as classical planning and is in the complexity class $\mathrm{NP}^{\mathrm{PP}}$-complete.

One of the earlier solvers, MAXPLAN [58], encodes planning problems into an E-MAJSAT problem. An E-MAJSAT problem is a Boolean formula quantified by a set of variables such that the first block is a set of existential variables followed by a block of randomized variables. This approach was tested on stochastic propositional planning problems and it performed competitively against other non-satisfiability solvers. The main contributions were a LRU cache to store sub-formulas for reuse and a smart cache based off the estimated difficulty of a subproblem.

3.3.1 Zander

Later the solvers Zander and C-MAXPLAN were introduced by [59] to solve contingent planning under uncertainty. Contingent plans are those which depend on observable variables during execution. Unlike MAXPLAN and C-MAXPLAN, Zander encodes probabilistic plans in Stochastic SAT. In fact, Zander was the first true stochastic SAT solver to incorporate techniques from satisfiability. Overall, the encoding of ZANDER is more compact than that of C-MAXPLAN, but it is also responsible for the resulting higher complexity class.

A partially observable probabilistic plan is specified by a set P of unique propositions that take on Boolean values. The states are configurations of propositions and an initial state is described by a decision tree for each proposition. The goal states consist of a partial set of all configurations where some propositions are satisfied. An action from the set A maps a state to a distribution over next states and only a subset of the propositions are observable. The solution is a maximal plan that recommends an action at each time step

conditioned on observable propositions where a maximal plan maximizes the probability of reaching a goal state.

Zander encoded plans as an alternating sequence of existential and randomized variables. Existential blocks correspond to action variables and randomized blocks to observations. A very general encoding for contingent plans is shown below in (3.11):

$$\overbrace{\exists x_{1,1},...,\exists x_{1,|A|}}^{\text{first action}}\overbrace{\forall y_{1,1},...,\forall y_{1,|O|}}^{\text{first observation}},...,\overbrace{\forall y_{|O|,1},...,\forall y_{|O|,|O|}}^{\text{last observation}}\overbrace{\exists x_{|A|,1},...,\exists x_{|A|,|A|}}^{\text{last action}}\overbrace{\forall z_1,...,\forall z_{|Z|}}^{\text{random outcomes}}\overbrace{\exists s_1,...,\exists s_{1,k_s}}^{\text{the states}} \tag{3.11}$$

Now future actions can depend on previous observations to allow contingency in plans. The final two blocks compute the effects of random outcomes and states consistent with the current set of actions and observations.

The contributions of Zander include a variable ordering heuristic, unit propagation, pure literal elimination, and thresholding. Although, results for the variable ordering heuristic were disappointing.

In follow up work, [55] described how non-chronological backtracking (NCB) can be incorporated into a SSAT solver. Early backtracking allows the solver to skip over computing the right branch after accumulating sufficient information during computation of the left branch such as a satisfiability or unsatisfiability which is called the reason. Their approach allowed for backtracking over many variables in a single jump if they were not contributing to the reason. The results looked promising on randomly generated problems, however on planning problems the overhead was significant and NCB was outperformed by the base model. They concluded that the structure of probabilistic planning problems does not offer many situations where NCB would be useful.

3.3.2 DC-SSAT

DC-SSAT [57], like Zander, is a stochastic SAT solver for probabilistic planning problems that is sound and complete. DC-SSAT works by decomposing the original problem into overlapping subproblems called viable partial assignments (VPA) and combining the solutions.

More specifically, DC-SSAT tackles SSAT encodings of completely observable probabilistic planning (COPP) problems. These types of problems have extra structure that can be exploited by the solver for much larger speedups. The requirements are that the first quantifier block must correspond to existential variables, among which the second half of the existential variables can only appear in clauses with other variables from the same existential block or an adjacent block of randomized variables.

Given a problem $\Phi = \Phi_1, \Phi_2, ..., \Phi_s$, we generate VPAs for each subproblem Φ_i that satisfies all the clauses in that subproblem. The subproblems can be created in $\mathcal{O}(|V|+|C|)$ that is linear in the number of variables and clauses. Next, a VPA for the current problem, Φ, is constructed by noting which set of VPAs together satisfy all the clauses. VPAs that share contradictory assignments of variables are discarded.

On benchmarks, DC-SSAT was able to consistently achieve 1-2 orders of magnitude improvements in both time and space in comparison to Zander. However, it is still an open question if the technique can be expanded to general SSAT problems efficiently. The issue is that the VPA for subproblem Φ_i may share variables with another VPA Φ_j such that $j > i + 1$. This rules out adjacent subproblems. Checking compatibility between subproblems further apart will need to keep track of many potentially useless subproblems that might undo any gains.

3.3.3 APPSAT

APPSSAT [56] is another solver based on Zander that tackles probabilistic contingent plans, but it is an anytime algorithm. In a probabilistic plan encoding to SSAT, many of the variables are randomly quantified. APPSAT seeks to determine the most likely instantiations of these variables that are consistent and form an approximate plan that is improved given additional time.

In APPSSAT, each observation variable (associated with a randomized quantifier) is assigned a new type called a branch variable, which does not have an associated probability distribution. Additionally, the full contingency of a plan is considered not just for the sampled path variables. The results show that APPSSAT is able to generate lower bounds on problem sizes beyond Zander's reach.

Chapter 4

Encoding Problems into POMDP

While the goal of this chapter is to explain how to reduce QBF and SSAT problems to POMDPs since they are all PSPACE-Complete, we start by explaining how to encode propositional satisfiability as POMDPs. This will ease the description of the QBF and SSAT conversions to POMDPs.

4.1 SAT $\Rightarrow$ POMDP

We describe a transformation from SAT to POMDP where a joint assignment of variables in SAT corresponds to a policy in the resulting POMDP encoding. A joint assignment is satisfiable if and only if the value of the corresponding policy is 1. The planning horizon in the POMDP encoding has a number of steps equal to 1 plus the number of variables in SAT. At each step, the POMDP assigns a truth value to a variable such that a joint assignment for all the variables is obtained at the end of the plan. In this section, we consider SAT problems where the Boolean formula F is defined by a set of clauses $C = \{c_1, ..., c_{|C|}\}$ where each c_i is a disjunction of literals from the set of variables $X = \{x_1, ..., x_{|X|}\}$. We describe below a (constructive) reduction that yields the following POMDP components:

- **State space** $\mathcal{S} = \{sat, c_1, c_2, ..., c_{|C|}\}$: 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.
- **Initial belief** b_0: Initially, none of the clauses are satisfied, hence we start with $b_0(sat) = 0$ and $b_0(c_i) = \frac{1}{|C|}\ \forall i$. Here we should not interpret the uniform distribution

over the c_i's literally. Instead, we will interpret a belief as denoting a set of clauses that remain to be satisfied. More precisely, whenever $b(c_i) > 0$, this means that clause c_i has yet to be satisfied. The precise probability $b(c_i)$ is not important. We will simply distinguish between $b(c_i) = 0$ (c_i is satisfied) and $b(c_i) > 0$ (c_i is not satisfied).

- **Action space** $\mathcal{A} = \{true, false\}$: At time step t, variable x_t is assigned either $true$ or $false$.

- **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 satisfiable state sat, if the current action assigns a truth value to variable x_{t+1} that satisfies clause c_i. Otherwise, the process remains in the current state.

$$Pr(s_{t+1}|s_t, a_t) \tag{4.1}$$
$$= \begin{cases} 1 & \text{if } s_t = c_i \text{ and } a_t \text{ satisfies } c_i \text{ and } s_{t+1} = sat \\ 1 & \text{if } s_t = c_i \text{ and } a_t \neg\text{satisfy } c_i \text{ and } s_{t+1} = c_i \\ 1 & \text{if } s_t = s_{t+1} = sat \\ 0 & \text{otherwise} \end{cases}$$

- **Reward function** $R(s_t, a_t)$: The reward function returns a non-zero reward only at the last time step where a reward of 1 is earned if the system is in the satisfiable state sat.

$$R(s_t, a_t) = \begin{cases} 1 & \text{if } t = |X| \text{ and } s_t = sat \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

  Note that the reward function does not depend on the action since this is not needed for the reduction.

- **Observation space** $\mathcal{O} = \{null\}$: There is a single $null$ observation, which means that this observation is uninformative and the process is effectively unobservable.

- **Observation distribution** $\Pr(o_{t+1}|a_t, s_{t+1})$: Since there is a single observation, the probability of that observation is always 1.

$$\Pr(o_{t+1} = null|a_t, s_{t+1}) = 1 \tag{4.3}$$

- **Horizon** $h = |X|$: There are $|X| + 1$ time steps (time step 0 to time step $|X|$).

- **Discount factor** $\gamma = 1$: Since the process has a finite horizon, we do not use any discount factor.

States	b_0	b_1	b_2	b_3	b_4	b_5
sat	0	1/3	2/3	2/3	1	1
$c_1 = \quad x_3 \vee \quad x_4 \vee \neg x_5$	1/3	1/3	1/3	1/3	0	0
$c_2 = \neg x_1 \vee \neg x_2 \vee \quad x_4$	1/3	0	0	0	0	0
$c_3 = \quad x_1 \vee \neg x_2 \vee \quad x_5$	1/3	1/3	0	0	0	0

Table 4.1: beliefs in SAT example after taking actions $x_1 = false$, $x_2 = false$, $x_3 = false$, $x_4 = true$ and $x_5 = true$.

To illustrate the POMDP reduction, we give an example in Table 4.1 for the SAT problem from Eq. 2.8 that shows how the belief is updated as we execute the sequence of actions $x_1 = false$, $x_2 = false$, $x_3 = false$, $x_4 = true$ and $x_5 = true$. Since the final belief b_5 has all its mass in the state sat, we conclude that this joint assignment is satisfiable.

The POMDP obtained by our reduction has $|C|+1$ states, 2 actions, 1 observation and a planning horizon of $|X|+1$ time steps. Note that the POMDP is not stationary since the transition and reward functions are not stationary. The transition function assigns a truth value to a different variable x_{t+1} at each time step t while the reward function produces a non-zero reward only at the last time step $|X|$. It is possible to obtain a stationary process by enlarging the state space, but this will complicate the reduction. Note also that the resulting POMDP is unobservable since the single *null* observation is uninformative. Unobservable POMDPs are NP-Complete [66] and therefore it makes sense that we can reduce SAT to unobservable POMDPs. The following theorem confirms that a SAT problem is satisfiable when there exists a POMDP policy with value 1.

Theorem 1. *In the above reduction from SAT to POMDP, there exists a policy with value 1 iff there exists a satisfiable joint assignment.*

Proof. Suppose there is a satisfiable joint assignment $\langle x_1 = a_0, x_2 = a_1, ..., x_{|X|} = a_{|X|-1}\rangle$, the transition function ensures that the final belief is $b_{|X|}(c_i) = 0 \ \forall i$ and $b_{|X|}(sat) = 1$ since all the clauses are satisfied. In turn, the reward function returns a value of 1 for this belief. Conversely, if we solve the POMDP and find a policy with value 1, it means that this policy selected actions that satisfied all the clauses and therefore the joint assignment corresponding to this policy is satisfiable. □

Example

Lets focus on the state representation by using Eq. 2.8 as an example. Shown in column two of Table 4.1 is the initial belief state the process could be in where each clause has equal probability of being unsatisfiable. In columns b_1 to b_5 is the belief state if we consider a policy, π^F that recommends the assignments $\neg x_1 \wedge x_3 \wedge \neg x_4 \wedge x_2 \wedge x_5$. Since the belief state has all its mass concentrated in the satisfiable state, *sat*, after all variables have been assigned in X, we can conclude that the original SAT problem is satisfiable and π^F is an optimal policy. An immediate reward of 1 is also gained from being in the satisfiable state.

In summary, we have shown how to reformulate a SAT problem as a POMDP where the size of the state space, $|C| + 1$, is linear in the number of clauses $|C|$.

4.2 QBF $\Rightarrow$ POMDP

We now extend the SAT reduction to POMDP to work with quantified Boolean formulas (QBF) of the form

$$Q_1 x_1 Q_2 x_2 ... Q_{|X|} x_{|X|} \; F(x_1, ..., x_{|X|})$$

Without loss of generality, we require the sequence of quantifiers to be of the form $\exists x_1, \forall x_2, \exists x_3, ..., \forall x_{|X|}$. If the sequence is non alternating, we can introduce extra quantified variables until we satisfy our condition without changing the semantics of the formula. In the worst case, we will add $|X| + 1$ new variables.

The reduction for QBF is similar to that of SAT. 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 universally quantified 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 universal quantifier. An existentially quantified variable is assigned a value by an action. We therefore relabel variables for simplicity to be x_t^a and x_t^o respectively.

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

- **State space** $\mathcal{S} = \{sat, c_1, c_2, ..., c_{|C|}\}$: This is the same as for SAT where state c_i indicates that the i^{th} clause has yet to be satisfied and state *sat* indicates that all clauses have been satisfied.
- **Initial belief** b_0: We start with a uniform initial belief, i.e., $b_0(sat) = \frac{1}{|C|+1}$ and $b_0(c_i) = \frac{1}{|C|+1}$ $\forall i$. For the QBF reduction, it is important to start with $b_0(sat) > 0$,

otherwise some observations might have 0 probability. Nevertheless, as in the SAT reduction, the precise probability of each state is not important. What matters is whether $b(c_i) = 0$ (c_i is satisfied) or $b(c_i) > 0$ (c_i is not satisfied).

- **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 the same as for SAT. When in state $s_t = c_i$, the process will transition deterministically to the satisfiable state sat if the current action satisfies clause c_i. Otherwise, the process remains in the current state.

$$
\begin{aligned}
&Pr(s_{t+1}|s_t, a_t) \\
&= \begin{cases} 1 & \text{if } s_t = c_i \text{ and } a_t \text{ satisfies } c_i \text{ and } s_{t+1} = sat \\ 1 & \text{if } s_t = c_i \text{ and } a_t \neg\text{satisfy } c_i \text{ and } s_{t+1} = c_i \\ 1 & \text{if } s_t = s_{t+1} = sat \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{4.4}
$$

- **Reward function** $R(s_t, a_t)$: The reward function is the same as for SAT. It returns a non-zero reward only at the last time step where a reward of 1 is earned if the system is in the satisfiable state sat.

$$
R(s_t, a_t) = \begin{cases} 1 & \text{if } t = |X| \text{ and } s_t = sat \\ 0 & \text{otherwise} \end{cases} \tag{4.5}
$$

- **Observation space $\mathcal{O} = \{true, false\}$**: Each observation is an assignment of $true$ or $false$ to the lowest unassigned universally quantified variable.

- **Observation distribution** $\Pr(o_{t+1}|a_t, s_{t+1})$: The observation distribution is setup in a way to ensure that $b(c_i)$ becomes 0 (at the next time step) when clause c_i is satisfied by the truth value assigned by the observation to 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 state. The stochastic distribution over the observation for the sat state is needed to

ensure that both truth values are considered for universally quantified variables.

$$Pr(o_{t+1}|a_t, s_{t+1}) \tag{4.6}$$
$$= \begin{cases} 0 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1} := o_{t+1}) \text{ satisfies } c_i \\ 1 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1} := o_{t+1}) \neg\text{satisfy } c_i \\ \frac{1}{2} & \text{otherwise (i.e., } s_{t+1} = sat) \end{cases}$$

Note that the observation distribution does not depend on the current action a_t. Note also that the observation probability for $s_{t+1} = sat$ does not have to be 1/2. Any probability between 0 and 1 is fine.

- **Horizon** $h = |X|/2$: Two variables (one existentially quantified and one universally quantified) 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 $|X|/2$).
- **Discount factor** $\gamma = 1$: Since the process has a finite horizon, we do not use any discount factor.

Consider again the Boolean formula in Eq. 2.8. Suppose that we quantify the 5 variables as follows: $\exists x_1, \forall x_2, \exists x_3, \forall x_4, \exists x_5$. Table 4.2 reports the intermediate (unnormalized) beliefs that are obtained when we process the sequence of actions and observations $a_0 = (x_1 := false), o_1 = (x_2 := false), a_1 = (x_3 := false), o_2 = (x_4 := true), a_2 = (x_5 := true)$. Let b^a_t denote the intermediate belief obtained after processing action a_{t-1}:

$$b^a(s_t) = \sum_{s_{t-1}} b(s_{t-1}) \Pr(s_t|s_{t-1}, a_{t-1}) \tag{4.7}$$

Similarly, let $\hat{b}^{ao}_t$ denote the unnormalized belief obtained after processing observation o_t

$$\hat{b}^{ao}(s_t) = b^a(s_t) \Pr(o_t|a_{t-1}, s_t) \tag{4.8}$$

This belief is unnormalized since we do not divide by $\Pr(o_t)$ to normalize it. Hence the sum of the probabilities for each belief is equal to the probability of the observations processed so far. In Table 4.2, $\hat{b}^a_3$ has all its mass in the *sat* state, which means that this joint assignment is satisfiable and the mass of 1/8 reflects the probability of o_1 and o_2. In QBF, we do not need to assign probabilities to observations, but this is an indirect way of making sure that all observations are considered for universally quantified variables. This also explains why the precise probabilities do not matter, as long as each observation has a non-zero probability.

States	b_0	b_1^a	$\hat{b}_1^{ao}$	$\hat{b}_2^a$	$\hat{b}_2^{ao}$	$\hat{b}_3^a$
sat	1/4	1/2	1/4	1/4	1/8	1/8
$c_1 = \quad x_3 \vee \quad x_4 \vee \neg x_5$	1/4	1/4	1/4	1/4	0	0
$c_2 = \neg x_1 \vee \neg x_2 \vee \ x_4$	1/4	0	0	0	0	0
$c_3 = \ x_1 \vee \neg x_2 \vee \ x_5$	1/4	1/4	0	0	0	0

Table 4.2: intermediate (unnormalized) beliefs in QBF example after processing $a_0 = (x_1 := false)$, $o_1 = (x_2 := false)$, $a_1 = (x_3 := false)$, $o_2 = (x_4 := true)$ and $a_3 = (x_5 := true)$.

Similar to the POMDP obtained from SAT, the one obtained from QBF has $|C| + 1$ states, 2 actions, 2 observations, a planning horizon of $|X|/2 + 1$ time steps and it is non-stationary. The main difference is the observation space, which has increased to 2 observations that are generally informative. The following theorem confirms that a QBF problem is satisfiable when there exists a POMDP policy with value 1.

Theorem 2. *The reduction from QBF to POMDP guarantees that optimal policies have value 1 iff the quantified Boolean formula is satisfiable.*

Proof. Assume there exists an optimal policy π that has value 1. We can construct an assignment tree in QBF that satisfies all clauses by assigning the variables in order of the POMDP time steps. An action a_t corresponds to the assignment of an existential variable and for observations we branch on both truth values. Since the value of the policy is 1, the reward of each branch must be equal to the probability of the observations of this branch and therefore the entire mass of the final belief must be in the *sat* state, which means that the entire assignment tree is satisfiable. Conversely, if we are given a satisfiable assignment tree we could construct an optimal policy with value 1 by taking actions based on the existential variable assignments. □

4.3 SSAT ⇒ POMDP

We now show a transformation from SSAT to POMDP. In our reduction, 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** $\mathcal{S} = \{sat, prob, c_1, c_2, ..., c_{|C|}\}$: The state space is similar to the one for SAT, with the addition of the special state *prob* whose probability will be 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., $b_0(sat) = b_0(prob) = b_0(c_i) = \frac{1}{|\mathcal{S}|}\ \forall i$.

- **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 similar to the one for SAT. In state $s_t = c_i$, the process will transition deterministically to the *sat* state if the current action satisfies clause c_i. Otherwise, the process remains in the current state.

$$Pr(s_{t+1}|s_t, a_t) \tag{4.9}$$
$$= \begin{cases} 1 & \text{if } s_t = c_i \text{ and } a_t \text{ satisfies } c_i \text{ and } s_{t+1} = sat \\ 1 & \text{if } s_t = c_i \text{ and } a_t \neg\text{satisfy } c_i \text{ and } s_{t+1} = c_i \\ 1 & \text{if } s_t = s_{t+1} = sat \\ 1 & \text{if } s_t = s_{t+1} = prob \\ 0 & \text{otherwise} \end{cases}$$

- **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* might be less than the mass in all the c_i's combined, the overall reward might be 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.

$$R(s_t, a_t) \tag{4.10}$$
$$= \begin{cases} |\mathcal{S}| & \text{if } t = \frac{|X|}{2} \text{ and } s_t = prob \text{ and } a_t = true \\ -|\mathcal{S}| & \text{if } t = \frac{|X|}{2} \text{ and } s_t = c_i \text{ and } a_t = true \\ 0 & \text{otherwise} \end{cases}$$

- **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 takes value $true$. The $prob$ state will effectively keep track of the probability of the observation sequence.

$$Pr(o_{t+1}|a_t, s_{t+1}) \tag{4.11}$$
$$= \begin{cases} 0 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1} := o_{t+1}) \text{ satisfies } c_i \\ 1 & \text{if } s_{t+1} = c_i \text{ and } (x^o_{t+1} := o_{t+1}) \neg\text{satisfy } c_i \\ \frac{1}{2} & s_{t+1} = sat \\ \lambda_t & \text{if } s_{t+1} = prob \text{ and } o_{t+1} = true \\ 1-\lambda_t & \text{if } s_{t+1} = prob \text{ and } o_{t+1} = false \end{cases}$$

- **Horizon** $h = |X|/2 + 1$: Two variables (one existentially quantified 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. 2.8. 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 4.3 reports the intermediate (unnormalized) beliefs that are obtained when we process the sequence of actions and observations $a_0 = (x_1 := false), o_1 = (x_2 := false), a_1 = (x_3 := false), o_2 = (x_4 := true), a_2 = (x_5 := true)$.

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
$c_1 = \quad x_3 \vee \quad x_4 \vee \neg x_5$	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 4.3: 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 $\Pr(o_2 = (x_4 := true)) = 6/7$.

Theorem 3. *The reduction from SSAT to POMDP guarantees that the value of any optimal POMDP policy is equal to the probability of satisfiability obtained by the best SSAT policy tree.*

Proof. Consider a POMDP policy π, which defines a policy tree. Each branch yields a final (unnormalized) belief with mass

$$\hat{b}^{\pi}_{o_{1:|X|/2}}(prob) = b_0(prob)\Pr(o_{1:|X|/2}|prob, \pi) \tag{4.12}$$

Based on the properties of the reward function, the expected reward of each branch is

$$R(\hat{b}^{\pi}_{o_{1:|X|/2}}) = \max_a \sum_s \hat{b}^{\pi}_{o_{1:|X|/2}}(s)R(s,a) \tag{4.13}$$

$$= \begin{cases} \Pr(o_{1:|X|/2}|prob, \pi) & \text{if branch is satisfying} \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

Hence the value of the best policy is

$$V^* = \max_{\pi} \sum_{o_{1:|X|/2}} R(\hat{b}^{\pi}_{o_{1:|X|/2}}) \tag{4.15}$$

$$= \max_{\pi} \sum_{o_{1:|X|/2} \text{ is satisfying}} \Pr(o_{1:|X|/2}|prob, \pi) \tag{4.16}$$

The above equation shows that the value of an optimal policy is equal to the highest expected probability of satisfying the Boolean formula of the corresponding SSAT problem. □

4.4 Discussion

In the reduction of satisfiability (SAT, QBF and 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.

Chapter 5

Encoding Problems into SSAT

In this chapter, we demonstrate through a constructive proof that it is possible to encode POMDPs into SSAT. Afterwards, we show a similar proof for encoding Inference into SSAT in Section 5.3.

Before continuing I would like to cover some of the notation used in this chapter. Given a set $X = \{x_0, x_1, \ldots, x_K\}$ for $x_i \in X$ we can list all elements as $x_0, x_1, \ldots, x_{|X|}$.

The notation below can be used to specify a conjunction of literals,

$$x_0 \wedge x_1 \wedge \ldots \wedge x_K \iff \bigwedge_{i=0}^{K} x_i \tag{5.1}$$

or a disjunction of literals.

$$x_0 \vee x_1 \vee \ldots \vee x_K \iff \bigvee_{i=0}^{K} x_i \tag{5.2}$$

Terms that contain an implication can be converted to a disjunction of terms as follows:

$$x \equiv k_x \rightarrow y \equiv k_y \iff x \not\equiv k_x \vee y \equiv k_y \tag{5.3}$$

5.1 POMDP $\Rightarrow$ 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 the set of Reals. However, rewards can be scaled and translated without changing the optimal policy. We define a new reward function

$$r(s,a) = \frac{R(s,a) - \min_{a',s'} R(s',a')}{\sum_{a,s}[R(s,a) - \min_{a',s'} R(s',a')]} \tag{5.4}$$

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:

$$a^* = \arg\max_{a \in \mathcal{A}} \sum_{s \in \mathcal{S}} b_0(s) r(s,a) \tag{5.5}$$

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. 5.5

$$\bigwedge_{k \in \mathcal{A}} \bigwedge_{i \in \mathcal{S}} (x_a \equiv k \wedge x_s \equiv i) \rightarrow x_r \equiv k|\mathcal{S}| + i \tag{5.6}$$

with quantified variables $\exists x_a$ followed by $\mathfrak{U} x_s$ and $\mathfrak{U} x_r$ in order, where $x_s \in \{0, ..., |\mathcal{S}| - 1\}$

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

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:

$$\Pr(x_s \equiv i) = b(i) \tag{5.7}$$

$$\Pr(x_r \equiv k|\mathcal{S}| + i) = r(i, k), \forall i, k \tag{5.8}$$

The first term in Eq. 5.6 guarantees that whenever an action, k, is taken, all clauses containing the term $x_a \not\equiv j$ for $j \neq k$ are removed since they have been satisfied and the term $x_a \not\equiv k$ is removed from the remaining 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:

$$V_h(b) = \max_{a \in \mathcal{A}} \sum_s b(s)[r(s, a) + \sum_o \sum_{s'} \Omega^a_{s'o} T^a_{ss'} V_{h-1}(b^a_o)] \tag{5.9}$$

where an optimal policy maximizes the value function over a horizon h. Based on Eq. 5.9, 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,

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

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 $\text{Ⱨ} x^t_o$ 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:

$$\mathfrak{U}x_s^t, \mathfrak{U}x_r^t \ \forall t \text{ such that } 1 \leq t \leq h \tag{5.10}$$

$$\mathfrak{U}x_\Omega^t, \mathfrak{U}x_T^t \ \forall t \text{ such that } 1 \leq t \leq h-1 \tag{5.11}$$

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:

$$\bigwedge_{1\leq t\leq h-1} x_p^t \equiv 0 \rightarrow \left(x_o^t \equiv 0 \wedge x_s^{t+1} \equiv 0\right) \tag{5.12}$$

$$\bigwedge_{1\leq t\leq h-1} x_p^t \equiv 0 \rightarrow x_p^{t+1} \equiv 0 \tag{5.13}$$

$$x_p^h \equiv 0 \tag{5.14}$$

$$\bigwedge_{k\in\mathcal{A}} \bigwedge_{i\in\mathcal{S}} (x_p^1 \equiv 0 \wedge x_a^1 \equiv k \wedge x_s^1 \equiv i) \rightarrow x_r^1 \equiv k|\mathcal{S}| + i \tag{5.15}$$

$$\bigwedge_{2\leq t\leq h} \bigwedge_{k\in\mathcal{A}} \bigwedge_{i\in\mathcal{S}} (x_p^{t-1} \equiv 1 \wedge x_p^t \equiv 0 \wedge x_a^t \equiv k \wedge x_s^t \equiv i) \rightarrow x_r^t \equiv k|\mathcal{S}| + i \tag{5.16}$$

$$\bigwedge_{1\leq t\leq h-1} \bigwedge_{k\in\mathcal{A}} \bigwedge_{i\in\mathcal{S}} \bigwedge_{j\in\mathcal{S}} (x_p^t \equiv 1 \wedge x_a^t \equiv k \wedge x_s^t \equiv i \wedge x_s^{t+1} \equiv j) \rightarrow x_{T_{k,i}}^{t+1} \equiv j \tag{5.17}$$

$$\bigwedge_{1\leq t\leq h-1} \bigwedge_{k\in\mathcal{A}} \bigwedge_{j\in\mathcal{S}} \bigwedge_{z\in\mathcal{O}} (x_p^t \equiv 1 \wedge x_a^t \equiv k \wedge x_s^{t+1} \equiv j \wedge x_o^t \equiv z) \rightarrow x_{\Omega_{k,j}}^t \equiv z \tag{5.18}$$

The formula in Eq. 5.13 ensures that once the process has stopped, it doesn't continue in the future and Eq. 5.12 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_s^t and x_o^t are used because they function as indicator variables and hence their distribution is uninformative. Since the horizon is finite, the formula in Eq. 5.14 ensures that the process is necessarily stopped at the horizon h.

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

$$\Pr(x_r^t \equiv k|\mathcal{S}| + i) = r(i, k), \quad \forall k \in \mathcal{A}, i \in \mathcal{S} \tag{5.19}$$

that encodes the reward to be received for a particular action, k, and state, i, pair as a probabilistic value. The formula in Eq. 5.15 ensures that the process receives a reward at the first time step when $x_p^1 \equiv F$, while Eq. 5.16 yields a reward at subsequent time steps when $x_p^{t-1} \equiv T$ changes to $x_p^t \equiv F$.

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

$$\Pr(x_{T_{k,i}}^{t+1} \equiv j) = \Pr(s_{t+1} = j | s_t = i, a_t = k) \tag{5.20}$$

The formula in Eq. 5.17 encodes this transition. Similarly, the variable $x_{\Omega_{k,j}}^{t+1}$ has domain $\mathcal{O}$ and encodes the observation distribution after executing action k and arriving in state j

$$\Pr(x_{\Omega_{k,j}}^{t} \equiv z) = \Pr(o_{t+1} = z | s_{t+1} = j, a_t = k) \tag{5.21}$$

The formula in Eq. 5.18 encodes this transition.

The following theorem confirms the equivalence of the SSAT problem obtained from a POMDP by the reduction.

Theorem 4. *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 $V^\pi = \Pr(\phi)$.*

Proof. Consider a base case policy tree of size 1. Let the policy tree be $\phi = \{x_a \equiv \hat{k}\}$ with clauses:

$$\bigwedge_{i \in \mathcal{S}} x_s \neq i \vee x_r \equiv \hat{k}|\mathcal{S}| + i \tag{5.22}$$

after simplifying Eq. 5.6 with the assignment $x_a \equiv \hat{k}$. The probability of satisfiability of (5.22) is equivalent to

$$\Pr(\phi) = \sum_i \Pr(x_s \equiv i) \Pr(x_r \equiv \hat{k}|\mathcal{S}| + i) = \sum_i b(i) r(i, \hat{k}) \tag{5.23}$$

by using the distributions from (5.7) and (5.8). However, (5.23) corresponds exactly to the policy that takes action $a_1 = \hat{k}$ and has a value of $V^\pi = \sum_i b(i) r(i, \hat{k})$.

For the general case, we give a proof by induction. Assume we have a policy tree ϕ_h, policy π_h, and we know $\Pr(\phi_h) = V^{\pi_h}$. Given ϕ_{h+1} and π_{h+1} show that $\Pr(\phi_{h+1}) = V^{\pi_{h+1}}$.

Since we are given the policy tree, all the actions are known. Therefore, if we simplify first by making the assignments in ϕ_{h+1}, then only the randomized variables will remain in the quantifier prefix. Any subset of variables can now be re-ordered freely. Based on the

number of randomize quantified variables we introduced for horizon h and $h+1$, encoding the probability of satisfiability is:

$$\Pr(\phi_{h+1}) = \sum_{v_1,\cdots,v_{h+1}}^{2} \sum_{z_1,\cdots,z_h}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{h+1}}^{|\mathcal{S}|} \prod_{l=1}^{h+1} \Pr(x_p^l = v_l, x_s^l = i, x_o^l = z_l, x_r^l)$$
$$\prod_{l=1}^{h} \Pr(x_\Omega^l, x_T^l | x_p^l = v_l, x_s^l = i, x_o^l = z_l) \tag{5.24}$$

To achieve the second line, the distribution for x_p is just a uniform distribution that can be factored out as 2^{-h}. However, each x_p is controlling the length of the process, so it naturally controls how many terms contribute to the total sum if we re-arrange by horizon and then simplify. Note that given values for x_p, x_o, x_s the other variables are forced by unit propagation to a specific value.

$$= 2^{-(h+1)} \sum_{\hat{h}=1}^{h+1} \sum_{z_1,\cdots,z_{\hat{h}-1}}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{\hat{h}}}^{|\mathcal{S}|} \prod_{l=1}^{\hat{h}} \Pr(x_s^l = i, x_o^l = z_l, x_r^l)$$
$$\prod_{l=1}^{\hat{h}-1} \Pr(x_\Omega^l, x_T^l | x_p^l = v_l, x_s^l = i, x_o^l = z_l) \tag{5.25}$$

Similarly, for the distribution x_o the constant, $|O|^{h-1}$, can be factored out in front and its value is used in the conditional distribution x_Ω.

$$= 2^{-(h+1)} |O|^{-h} \sum_{\hat{h}=1}^{h+1} \sum_{z_1,\cdots,z_{\hat{h}-1}}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{\hat{h}}}^{|\mathcal{S}|} \prod_{l=1}^{\hat{h}} \Pr(x_s^l = i, x_r^l) \prod_{l=1}^{\hat{h}-1} \Pr(x_\Omega^l, x_T^l | x_p^l = v_l, x_s^l = i, x_o^l = z_l) \tag{5.26}$$

the next variable x_s^l has uniform distribution for all $l > 1$ and the initial belief when $l = 1$. Therefore, we can simplify the equation by pulling out the constant factors again.

$$= 2^{-(h+1)}(|O| \cdot |S|)^{-h} \sum_{\hat{h}=1}^{h+1} \sum_{z_1,\cdots,z_{\hat{h}-1}}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{\hat{h}}}^{|\mathcal{S}|} \Pr(x_s^1 = i) \prod_{l=1}^{\hat{h}} \Pr(x_r^l)$$
$$\prod_{l=1}^{\hat{h}-1} \Pr(x_\Omega^l, x_T^l | x_p^l = v_l, x_s^l = i, x_o^l = z_l) \tag{5.27}$$

According to the distribution x_p, rewards x_r will only be given at the end of the process for each $\hat{h}$.

$$= 2^{-(h+1)}(|O| \cdot |S|)^{-h} \sum_{\hat{h}=1}^{h+1} \sum_{z_1,\cdots,z_{\hat{h}-1}}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{\hat{h}}}^{|\mathcal{S}|} \Pr(x_s^1 = i) \Pr(x_r^{\hat{h}})$$
$$\prod_{l=1}^{\hat{h}-1} \Pr(x_\Omega^l, x_T^l | x_p^l = v_l, x_s^l = i, x_o^l = z_l) \tag{5.28}$$

If we replace the distributions below with their definitions and replace constants with the proportional relation, we obtain

$$\propto \sum_{\hat{h}=1}^{h+1} \sum_{z_1,\cdots,z_{\hat{h}-1}}^{|\mathcal{O}|} \sum_{s_1,\cdots,s_{\hat{h}}}^{|\mathcal{S}|} b(s_1) \prod_{l=1}^{\hat{h}-1} \Omega_{s_{l+1},z_l}^{a_l} T_{s_l,s_{l+1}}^{a_l} r(s_{\hat{h}}, a_{\hat{h}}) \tag{5.29}$$

$$= \sum_{s_1}^{|\mathcal{S}|} b(s_1) \left(r(s_1, a_1) + \sum_{z_1}^{|\mathcal{O}|} \sum_{s_2}^{|\mathcal{S}|} \Omega_{s_2,z_1}^{a_1} T_{s_1,s_2}^{a_1} \left(r(s_2, a_2) + \sum_{z_2}^{|\mathcal{O}|} \sum_{s_3}^{|\mathcal{S}|} \Omega_{s_3,z_2}^{a_2} T_{s_2,s_3}^{a_2} \left(r(s_3, a_3) + ... \right) \right) \right) \tag{5.30}$$

$$= \sum_{s_1}^{|\mathcal{S}|} b(s_1) \left(r(s_1, a_1) + \sum_{z_1}^{|\mathcal{O}|} \sum_{s_2}^{|\mathcal{S}|} \Omega_{s_2,z_1}^{a_1} T_{s_1,s_2}^{a_1} \Pr(\phi_h) \right) \tag{5.31}$$

where $\Pr(\phi_h) = r(s, a) + \sum_{z}^{|\mathcal{O}|} \sum_{s'}^{|\mathcal{S}|} \Omega_{s',z}^{a} T_{s,s'}^{a} \Pr(\phi_{h-1})$

Now consider the reverse. Given a policy, π_{h+1}, with value function $V^{\pi_{h+1}}$ there exists

a satisfiable policy tree, ϕ_{h+1}, with satisfiability probability $\Pr(\phi_{h+1})$ such that $V^{\pi_{h+1}} = \Pr(\phi_{h+1})$. First, Bellman's equation for a $h+1$ horizon policy is:

$$V^{\pi_{h+1}} = \sum_{s} b^{h+1}(s) \left(r(s,a) + \sum_{o} \sum_{s'} \Omega^{a}_{s'o} T^{a}_{ss'} V^{\pi_h}(b^{a}_{o}) \right), \; a = \pi(b) \tag{5.32}$$

However, any $h+1$ horizon policy can be written as a linear combination of h horizon policies. Since we know $Pr(\phi_h) = V^{\pi}_{h}$ by the inductive step, we conclude, that (5.31) and (5.32) are equal. Therefore, the probability of satisfying a $h+1$ depth policy tree corresponds to the value function of a $h+1$ step policy. □

In Algorithm 4, a procedure is shown to encode a POMDP into a SSAT problem. The first step is to normalize the reward function into a valid probability distribution $r(s,a)$. The next step is to create 3 lists to hold the sequence of variables in order, the distribution for randomly quantified variables, and the dynamics of the POMDP encoded as a set of clauses.

Algorithm 4 Encode Finite Horizon POMDP into SSAT

procedure ENCODE-POMDP(POMDP)
  $(S, A, O, T, \Omega, R, b_0, H) = POMDP$
  for $a \in A$ **do**
    for $s \in S$ **do**
      $r(s,a) = \frac{R(s,a) - \min_{a',s'} R(s',a')}{\sum_{a,s}[R(s,a) - \min_{a',s'} R(s',a')]}$
    end for
  end for
  vars $\leftarrow$ list-init()
  distr $\leftarrow$ list-init()
  clauses $\leftarrow$ list-init()
  list-add(vars, $\exists x_a^1, |A|$)
  list-add(vars, $\exists x_p^1, 2$)
  list-add(distr, $\Pr(x_p^1 \equiv i) = \frac{1}{2}\ \forall i$)
  for h from 2 to H **do**
    list-add(vars, $\forall x_o^{h-1}, |O|$)
    list-add(distr, $\Pr(x_o^{h-1} \equiv z) = \frac{1}{|O|}\ \forall z$)
    list-add(vars, $\exists x_a^h, |A|$)
    list-add(vars, $\exists x_p^h, 2$)
    list-add(distr, $\Pr(x_p^h \equiv i) = \frac{1}{2}\ \forall i$)
  end for
  list-add(vars, $\forall x_s^1, |S|$)
  list-add(distr, $\Pr(x_s^1 \equiv i) = b_0(i)\ \forall i$)
  list-add(vars, $\forall x_r^1, |A||S|$)
  list-add(distr, $\Pr(x_r^1 \equiv k|\mathcal{S}| + i) = r(i,k)\ \forall k, i$)
  for h from 2 to H **do**
    list-add(vars, $\forall x_s^h$)
    list-add(distr, $\Pr(x_s^h \equiv i) = \frac{1}{|S|}\ \forall i$)
    list-add(vars, $\forall x_r^h$)
    list-add(distr, $\Pr(x_r^h \equiv k|\mathcal{S}| + i) = r(i,k)\ \forall k, i$)
    for $k \in A$ **do**
      for $i \in S$ **do**
        list-add(vars, $\forall x_{T_{k,i}}^{h-1}$)
        list-add(distr, $\Pr(x_{T_{k,i}}^{h-1} \equiv j) = T_{i,j}^k\ \forall j$)
        list-add(vars, $\forall x^{h-1} t_{\Omega_{k,i}}$)
        list-add(distr, $\Pr(x_{\Omega_{k,i}}^{h-1} \equiv z) = \Omega_{i,z}^k\ \forall z$)
      end for
    end for
  end for
  list-add(clauses, $x_p^H \equiv 0$)
  for $k \in A$ **do**
    for $i \in S$ **do**
      list-add(clauses, $(x_p^1 \equiv 0 \wedge x_a^1 \equiv k \wedge x_s^1 \equiv i) \rightarrow x_r^1 \equiv k|\mathcal{S}| + i$)
    end for
  end for
  for h from 1 to $H-1$ **do**
    list-add(clauses, $x_p^h \equiv 0 \rightarrow \left(x_o^h \equiv 0 \wedge x_s^{h+1} \equiv 0\right)$)
    list-add(clauses, $x_p^h \equiv 0 \rightarrow x_p^{h+1} \equiv 0$)
    for $k \in A$ **do**
      for $i \in S$ **do**
        list-add(clauses, $(x_p^h \equiv 1 \wedge x_p^{h+1} \equiv 0 \wedge x_a^{h+1} \equiv k \wedge x_s^{h+1} \equiv i) \rightarrow x_r^{h+1} \equiv k|\mathcal{S}| + i$)
        for $j \in S$ **do**
          list-add(clauses, $(x_p^h \equiv 1 \wedge x_a^h \equiv k \wedge x_s^h \equiv i \wedge x_s^{h+1} \equiv j) \rightarrow x_{T_{k,i}}^{h+1} \equiv j$)
        end for
        for $z \in O$ **do**
          list-add(clauses, $(x_p^h \equiv 1 \wedge x_a^h \equiv k \wedge x_s^{h+1} \equiv i \wedge x_o^h \equiv z) \rightarrow x_{\Omega_{k,i}}^h \equiv z$)
        end for
      end for
    end for
  end for
  return SSAT(vars, distr, clauses)
end procedure

5.1.1 Example

Let's consider the tiger problem from the literature for a horizon of 2. The output of applying Algorithm 4 is shown. The parameters are in Table 5.1. First, following Line 5 the reward function is scaled and translated to be in the interval [0,1] and sums to 1 (valid probability distribution).

Table 5.1: The parameters for the tiger POMDP problem with normalized rewards.

A_t	S_t	S_{t+1}	$\Pr(S_{t+1}\|S_t, A_t)$
0	0	0	1.0
0	1	1	1.0
1	0	0	0.5
1	0	1	0.5
1	1	0	0.5
1	1	1	0.5
2	0	0	0.5
2	0	1	0.5
2	1	0	0.5
2	1	1	0.5

A	S	$R(S_t, A_t)$
0	0	0.236842
0	1	0.236842
1	0	0.0
1	1	0.263158
2	0	0.263158
2	1	0.0

A_t	S_{t+1}	O_{t+1}	$\Pr(O_{t+1}\|S_{t+1}, A_t)$
0	0	0	0.85
0	0	1	0.15
0	1	0	0.15
0	1	1	0.85
1	0	0	0.5
1	0	1	0.5
1	1	0	0.5
1	1	1	0.5
2	0	0	0.5
2	0	1	0.5
2	1	0	0.5
2	1	1	0.5

Second, we need to define the variables that will be used and their distribution. Following Lines 11 to 19 of Algorithm 4, the policy can be represented by a set of variables with the following quantifier ordering

$$\exists x_a^1, \text{Ч} x_p^1, \text{Ч} x_o^1, \exists x_a^2, \text{Ч} x_p^2 \tag{5.33}$$

that corresponds to an action, an observation, the next action and then the process terminates. Note that the auxiliary variables, x_p, govern when the process terminates. The distribution of these variables are given below:

$$\Pr(x_p^t \equiv i) = \frac{1}{2},\ 1 \leq t \leq 2, 0 \leq i \leq 1 \tag{5.34}$$

$$\Pr(x_o^1 \equiv z) = \frac{1}{|O|} \forall z \in O \tag{5.35}$$

Finally, we have the set of variables in a last randomized block that is generated in Lines 21 to 35:

$$\mathfrak{U}x_s^1, x_s^2, x_r^1, x_r^2, \tag{5.36}$$

$$x_{T_{0,0}}^1, x_{T_{0,1}}^1, x_{T_{1,0}}^1, x_{T_{1,1}}^1, x_{T_{2,0}}^1, x_{T_{2,1}}^1, \tag{5.37}$$

$$x_{\Omega_{0,0}}^1, x_{\Omega_{0,1}}^1, x_{\Omega_{1,0}}^1, x_{\Omega_{1,1}}^1, x_{\Omega_{2,0}}^1, x_{\Omega_{2,1}}^1 \tag{5.38}$$

The distribution of the randomized variables for T = {1, 2} timesteps are:

$$\Pr(x_s^t \equiv i) = \frac{1}{2} \forall t \in T, i \in S \tag{5.39}$$

$$\Pr(x_r^t \equiv k|\mathcal{S}| + i) = R(k, i)\ \forall t \in T, i \in S \tag{5.40}$$

$$\Pr(x_r^t \equiv 0) = 0.236842,\ \forall t \in T \tag{5.41}$$

$$\Pr(x_r^t \equiv 1) = 0.236842,\ \forall t \in T \tag{5.42}$$

$$\Pr(x_r^t \equiv 2) = 0.0,\ \forall t \in T \tag{5.43}$$

$$\Pr(x_r^t \equiv 3) = 0.263158,\ \forall t \in T \tag{5.44}$$

$$\Pr(x_r^t \equiv 4) = 0.263158,\ \forall t \in T \tag{5.45}$$

$$\Pr(x_r^t \equiv 5) = 0.0,\ \forall t \in T \tag{5.46}$$

$$\Pr(x_{T_{k,i}}^1 \equiv j) = \Pr(s_{t+1} = j|s_t = i, a_t = k)\ \forall k \in A, i \in S \tag{5.47}$$

$$\Pr(x_{\Omega_{k,j}}^1 \equiv z) = \Pr(o_{t+1} = z|s_{t+1} = j, a_t = k)\ \forall k \in A, j \in S \tag{5.48}$$

where the transition and observation distributions are unchanged from the POMDP model.

Finally, we use Lines 39 to 55 to generate the clauses. The clauses in Table 5.2 are associated with the reward function and when the process terminates through the auxiliary variables.

Next, the clauses in each column of Table 5.3 represent the transition constraints and observation constraints. These constraints ensure that the right probabilities are implied based on the next state and observation.

Table 5.2: Example encoding of the Tiger problem from Table 5.1 for a horizon of length 2.

reward constraints

$$x_a^1 \equiv 0 \wedge x_s^1 \equiv 0 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 0$$
$$x_a^1 \equiv 0 \wedge x_s^1 \equiv 1 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 1$$
$$x_a^1 \equiv 1 \wedge x_s^1 \equiv 0 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 2$$
$$x_a^1 \equiv 1 \wedge x_s^1 \equiv 1 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 3$$
$$x_a^1 \equiv 2 \wedge x_s^1 \equiv 0 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 4$$
$$x_a^1 \equiv 2 \wedge x_s^1 \equiv 1 \wedge x_p^1 \equiv 1 \Rightarrow x_r^1 \equiv 5$$

$$x_a^2 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 0$$
$$x_a^2 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 1$$
$$x_a^2 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 2$$
$$x_a^2 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 3$$
$$x_a^2 \equiv 2 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 4$$
$$x_a^2 \equiv 2 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \wedge x_p^2 \equiv 1 \Rightarrow x_r^2 \equiv 5$$

$$x_p^1 \equiv 0 \Rightarrow x_o^1 \equiv 0$$
$$x_p^1 \equiv 0 \Rightarrow x_s^2 \equiv 0$$

$$x_p^1 \equiv 0 \Rightarrow x_p^2 \equiv 0$$
$$x_p^2 \equiv 0$$

Table 5.3: Example encoding of the transition and observation distributions in the Tiger problem from Table 5.1 for a horizon of length 2.

transition constraints	observation constraints
$x_a^1 \equiv 0 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{0,0}}^1 \equiv 0$	$x_a^1 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{0,0}}^1 \equiv 0$
$x_a^1 \equiv 0 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{0,0}}^1 \equiv 1$	$x_a^1 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{0,0}}^1 \equiv 1$
$x_a^1 \equiv 0 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{0,1}}^1 \equiv 0$	$x_a^1 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{0,1}}^1 \equiv 0$
$x_a^1 \equiv 0 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{0,1}}^1 \equiv 1$	$x_a^1 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{0,1}}^1 \equiv 1$
$x_a^1 \equiv 1 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{1,0}}^1 \equiv 0$	$x_a^1 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{1,0}}^1 \equiv 0$
$x_a^1 \equiv 1 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{1,0}}^1 \equiv 1$	$x_a^1 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{1,0}}^1 \equiv 1$
$x_a^1 \equiv 1 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{1,1}}^1 \equiv 0$	$x_a^1 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{1,1}}^1 \equiv 0$
$x_a^1 \equiv 1 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{1,1}}^1 \equiv 1$	$x_a^1 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{1,1}}^1 \equiv 1$
$x_a^1 \equiv 2 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{2,0}}^1 \equiv 0$	$x_a^1 \equiv 2 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{2,0}}^1 \equiv 0$
$x_a^1 \equiv 2 \wedge x_s^1 \equiv 0 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{2,0}}^1 \equiv 1$	$x_a^1 \equiv 2 \wedge x_s^2 \equiv 0 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{2,0}}^1 \equiv 1$
$x_a^1 \equiv 2 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{2,1}}^1 \equiv 0$	$x_a^1 \equiv 2 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 0 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{2,1}}^1 \equiv 0$
$x_a^1 \equiv 2 \wedge x_s^1 \equiv 1 \wedge x_s^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{T_{2,1}}^1 \equiv 1$	$x_a^1 \equiv 2 \wedge x_s^2 \equiv 1 \wedge x_o^2 \equiv 1 \wedge x_p^1 \equiv 0 \Rightarrow x_{\Omega_{2,1}}^1 \equiv 1$

5.2 Summary

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 4 variables initially (action, policy horizon, current state, and reward) and $5 + 2|\mathcal{A}||\mathcal{S}|$ variables per time step for the remaining $h - 1$ time steps. This gives a complexity of:

$$|X| = O\Big(h|\mathcal{A}||\mathcal{S}|\Big) \tag{5.49}$$

The number of clauses initially is $1 + |\mathcal{A}||\mathcal{S}|$ and $3 + |\mathcal{A}||\mathcal{S}| + |\mathcal{A}||\mathcal{S}|^2 + |\mathcal{A}||\mathcal{S}||\mathcal{O}|$ clauses per time step for the remaining $h - 1$ time steps. This gives a complexity of:

$$|C| = O\Big(h|\mathcal{A}||\mathcal{S}|\big(|\mathcal{S}| + |\mathcal{O}|\big)\Big) \tag{5.50}$$

5.3 Inference ⇒ SSAT

In this section, we propose a solution for inference in Bayesian Networks by encoding inference queries as SSAT problems. The intuition of the encoding is to represent each full joint assignment of the variables as an assignment to matching indicator variables in the SSAT encoding and the probability of satisfiability corresponds to the probability of just such an assignment from the joint distribution. We now show a reduction from probabilistic inference to SSAT. Start by (1) encoding the probabilistic variables and (2) the parameters of each distribution.

In the quantifier prefix, we'll have a set of randomly quantified variables followed by existentially quantified variables. Let the existentially quantified indicator variable x_{y_i} be associated with each probabilistic variable, y_i, and have cardinality $|y_i|$. For each conditional probability distribution, $\Pr(y_0|y_1, ..., y_m)$, create a randomly quantified variable, $\lambda_{y_0=v|y_1,...,y_m}$, with cardinality $|y_0|$ and distribution

$$\Pr(\lambda_{y_0=v|y_1,...,y_m} = v) = \Pr(y_0 = v|y_1, ..., y_m) \tag{5.51}$$

for each instantiation of the parents' values. The corresponding clauses are:

$$x_{y_1} \wedge x_{y_2} \cdots \wedge x_{y_m} \wedge x_{y_0=v} \Rightarrow \lambda_{y_0=v|y_1,...,y_m} \tag{5.52}$$

They indicate that if all the parent indicator variables are set and $y_0 = v$ then the parameter, λ, takes on value v with probability $\Pr(y_0 = v|y_1, ..., y_m)$.

Now it is simple to encode additional constraints on probabilistic variables to incorporate evidence through clauses that contain strictly indicator variables. These extra clauses are usually beneficial to reduce the problem complexity significantly.

5.3.1 Example

To get a feel for how the encoding works, let us use the example network from Section 3.2.1 shown here below.

The set of variables includes a block of randomized variables for the parameters of each conditional probability distribution followed by an existentially quantified block for the variables in the network:

$$\mathrm{R}x_a, \mathrm{R}x_{b|a=0}, \mathrm{R}x_{b|a=1}, \mathrm{R}x_{c|b=0}, \mathrm{R}x_{c|b=1}, \exists x_A, \exists x_B, \exists x_C \tag{5.53}$$

Table 5.4: Conditional distribution for random variables A, B, and C.

a	$\Pr(A)$
0	0.4
1	0.6

a	b	$\Pr(B\|A)$
0	0	0.2
0	1	0.8
1	0	0.7
1	1	0.3

b	c	$\Pr(C\|B)$
0	0	0.0
0	1	0.0
0	2	1.0
1	0	0.2
1	1	0.6
1	2	0.2

The distribution for each of the randomly quantified variables is:

$$\Pr(x_a) = (0.4, 0.6) \tag{5.54}$$

$$\Pr(x_{b|a=0}) = (0.2, 0.8), \quad \Pr(x_{b|a=1}) = (0.7, 0.3) \tag{5.55}$$

$$\Pr(x_{c|b=0}) = (0.0, 0.0, 1.0), \quad \Pr(x_{c|b=1}) = (0.2, 0.6, 0.2) \tag{5.56}$$

The clauses in each column of Table 5.5 are associated with different conditional distributions:

Table 5.5: Example encoding of the Bayesian network in Table 5.4.

$$x_A \equiv 0 \Rightarrow x_a \equiv 0$$
$$x_A \equiv 1 \Rightarrow x_a \equiv 1$$

$$x_A \equiv 0 \wedge x_B \equiv 0 \Rightarrow x_{b|a=0} \equiv 0$$
$$x_A \equiv 0 \wedge x_B \equiv 1 \Rightarrow x_{b|a=0} \equiv 1$$
$$x_A \equiv 1 \wedge x_B \equiv 0 \Rightarrow x_{b|a=1} \equiv 0$$
$$x_A \equiv 1 \wedge x_B \equiv 1 \Rightarrow x_{b|a=1} \equiv 1$$

$$x_B \equiv 0 \wedge x_C \equiv 0 \Rightarrow x_{c|b=0} \equiv 0$$
$$x_B \equiv 0 \wedge x_C \equiv 1 \Rightarrow x_{c|b=0} \equiv 1$$
$$x_B \equiv 0 \wedge x_C \equiv 2 \Rightarrow x_{c|b=0} \equiv 2$$
$$x_B \equiv 1 \wedge x_C \equiv 0 \Rightarrow x_{c|b=1} \equiv 0$$
$$x_B \equiv 1 \wedge x_C \equiv 1 \Rightarrow x_{c|b=1} \equiv 1$$
$$x_B \equiv 1 \wedge x_C \equiv 2 \Rightarrow x_{c|b=1} \equiv 2$$

We see that the encoding has similarities with that of weighted model counting from Section 3.2.1. In terms of complexity, the number of clauses generated is about half since implication is used instead of equivalence. Also, a similar number of indicator variables and parameter variables is used (assuming we are restricted to binary variables). However, using a finite domain over variables will allow us to use fewer variables in general.

Note here that the use of equivalence would lead to incorrect models. The reason is

that different parameter values in the CPT are shared across randomized variable values in SSAT and that will lead to a conflicting assignment. For instance consider the current example for the CPT of variable C.

$$x_B \equiv 0 \wedge x_C \equiv 0 \iff x_{c|b=0} \equiv 0 \tag{5.57}$$
$$x_B \equiv 0 \wedge x_C \equiv 1 \iff x_{c|b=0} \equiv 1 \tag{5.58}$$

If we convert these formulas to CNF and consider what happens under the assignment $x_{c|b=0} \equiv 0$, we get:

$$x_B \not\equiv 0 \vee x_C \not\equiv 0 \vee x_{c|b=0} \equiv 0 \tag{5.59}$$
$$x_B \equiv 0 \vee x_{c|b=0} \not\equiv 0 \tag{5.60}$$
$$x_C \equiv 0 \vee x_{c|b=0} \not\equiv 0 \tag{5.61}$$
$$x_B \not\equiv 1 \vee x_C \not\equiv 2 \vee x_{c|b=1} \equiv 2 \tag{5.62}$$
$$x_B \equiv 1 \vee x_{c|b=1} \not\equiv 2 \tag{5.63}$$
$$x_C \equiv 2 \vee x_{c|b=1} \not\equiv 2 \tag{5.64}$$

Then (5.59) is satisfied by $x_{c|b=0} = 0$, (5.60) implies $x_B = 0$, and (5.61) implies $x_C = 0$. However, it also follows that (5.62) is satisfied by $x_{c|b=1} \equiv 2$, but that conflicts with our original assignment $x_{c|b=0} = 0$.

5.4 Summary

We show an encoding to reduce probabilistic inference to SSAT by encoding probabilistic variables and the parameters of the distributions separately. This is similar to the encoding used in weighted model counting however finite domain variables and implication instead of equivalence operators are exploited to reduce the total number of variables and clauses.

Chapter 6

SSAT Solver

In this chapter, we describe our Stochastic SAT solver, SSAT-Prime, with many successful techniques incorporated from the SAT and #SAT literature including watch literals, component decomposition, caching, and symmetry. We also consider incremental bounds for when an exact solution cannot be found in time. Afterwards, pure literal and clause learning techniques were tried, but did not transfer well into the SSAT problems we are interested in solving.

6.1 Finite Domain

Although SAT solvers usually operate on strictly Boolean variables, in our case for easy encoding to SSAT, consider an extension to finite domain variables with equality ($=$) and non-equality ($\neq$) as literals. As an example, the clause $(x_1 \equiv v_0 \vee x_2 \not\equiv v_3)$ with variables whose domains are $x_1 \in \{v_0, v_1\}$ and $x_2 \in \{v_0, v_1, v_2, v_3\}$ is satisfied when either $x_1 = v_0$ or $x_2 \neq v_3$.

6.2 Unit Rule

In satisfiability problems the unit rule is very effective across all problems. 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. The unit rule says that it is valid to assign $x_1 = v_0$ and 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.

Let's define an active literal as a literal whose associated variable has not been assigned and an active clause as a clause with at least one active literal. We can apply the unit rule efficiently by tracking which clauses mention each literal for all variables. Whenever we assign a variable to a value with literal, $x_i = v$, we know that all clauses that contain the literal $x_i \equiv v$ are satisfied. However, all clauses containing $x_i \not\equiv v$ have a counter that is decremented. In each clause, the counter is initialized to the number of literals in the clause and represents the number of unassigned literals remaining. To detect unit clauses, it is only required to know when the number of unassigned literals goes from 2 to 1. Furthermore, in backtracking we have to undo all the decrements with increments.

To analyze the complexity, assume a k-SAT problem with parameters $(k, |C|, |V|, \tau, \theta, \pi)$ is generated such that there are $|V|$ variables, $|C|$ clauses, exactly k literals per clause, each variable is of τ dimensions, the probability that a variable will be existentially quantified is θ (and $1 - \theta$ for randomly quantified variables) and for randomly quantified variables, the associated distribution π_i is sampled from a uniform Dirichlet of τ dimensions.

If a variable appears in $z = |C|\binom{|V|}{k}$ clauses and each value of a variable appears in $z_i = \frac{|C|}{\tau}\binom{|V|}{k}$ clauses, then in the worst case we will perform z_i updates for $\tau - 1$ values of an assigned variable. In backtracking, the reverse computation will also be required for a total of $2z_i(\tau - 1)$ or $\frac{2|C|\cdot(\tau-1)}{\tau}\binom{|V|}{k}$ updates.

In practice, most SAT solvers spend up to 90% of their time, according to [64], doing unit rule updates. Depending on the implementation, updates are made before each assignment and after each unassignment to the clause and variable.

6.2.1 Two-Literal Watch Scheme

The watch literal data-structure is an efficient way to determine assignments from the unit rule. We would like to minimize the number of clauses we check when assigning a variable and unassigning in backtracking. According to the explanation above, we learned that 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 $x \neq v$. Assume one of the two watch literals, WL1, is falsified by the assignment. There are two possible outcomes:

1. Invariance is maintained: the clause remains active since we are able to find another literal, WL0, in the clause that is unassigned and there by replacing the currently falsified literal WL1.

2. Watch literal implied: we are unable to find another unassigned literal that is not being watched to replace WL1. Therefore, we have a unit clause and we can imply the other literal being watched WL2 (which could be a source of conflict).

The key benefit of the two-literal watching scheme is that there is no further work necessary upon backtracking and unassigning a variable can be done in constant time without any change to the watch literals. In fact, reassigning a variable to a different value will in general be faster the second time since the first time we removed watch literals that refer to the variable. Note that the initial choice of which literals to watch is arbitrary, however it is recommended to spread them apart.

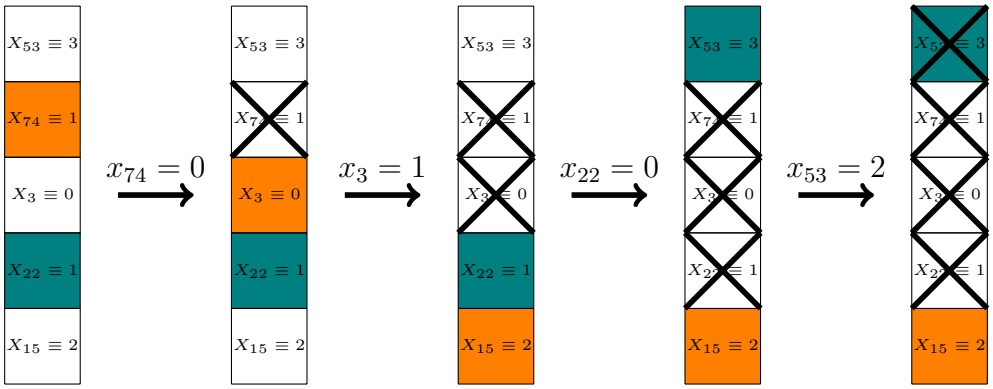


Figure 6.1: Watch literals for a clause when making the assignments $x_{74} = 0, x_3 = 1, x_{22} = 0, x_{53} = 2$. Orange is watch literal 1 and teal is watch literal 2.

Consider an example as shown in Figure 6.1 where for a clause with watch literal 1 (WL1 is orange) and watch literal 2 (WL2 is teal) are initialized to indices 1 and 3. First, the assignment $x_{74} = 0$ is made, which falsifies WL1 and forces us to find another literal that is active to preserve the invariance. We pick the literal $X_3 \equiv 0$. We continue until

the last column where we assign $x_{53} = 2$, but WL2 (teal) is watching $x_{53} \equiv 3$. There is no other unassigned literal besides WL1. Therefore, we can conclude WL1 must hold and make the implication $x_{15} = 2$.

6.2.2 Improved Watch Literal Scheme

One 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 large a 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:

$$\text{clauses[c].satisfied} = \text{stack-size(satisfy)} \tag{6.1}$$

$$\text{stack-push(satisfy, c)} \tag{6.2}$$

In Algorithm 5, we define **is-satisfied** that determines in constant time if a clause is satisfied. 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 5 Determine in constant time if a clause is satisfied

```
1: procedure IS-SATISFIED(satisfy, c)
2:     s = clauses[c].satisfied
3:     return stack-index(satisfy, s) ≡ c ∧ s <stack-size(satisfy)
4: end procedure
```

To show the rule is consistent and complete, first, if the clause at position c is satisfied, then it was 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.

Given an assignment literal, $x \equiv v$, the complete function is shown in Algorithm 6. First set the probability to satisfiability to the default value of 1.0 and if the variable is randomly quantified we update the probability. Next, mark all clauses containing the literal satisfied if they weren't before. Finally, update all literals belonging to variable x that were previously watch literals. If those clauses containing x were already satisfied, we can skip to the next clause. Afterwards, find a replacement watch literal and if none is found the other watch literal is implied. We conclude by checking if the alternative watch literal is unassigned then the assignment is followed, else there is a conflict.

Algorithm 6 Two-Literal Watching Scheme

```
1: procedure CONSTRAINT-PROPAGATE(x ≡ v)
2:     path = 1.0
3:     if Q(x) ≡ ℵ then
4:         path ← path · Pr(x ≡ v)
5:     end if
6:     for c ∈ clause-with-literal(x ≡ v) do
7:         if ¬clause-satisfied(c) then
8:             clause(c, satisfied) = true
9:         end if
10:    end for
11:    for v̂ ∈ {0, ..., |x| − 1} − {v} do
12:        for c ∈ clauses-watched(x, v̂) do
13:            if clause-satisfied(c) then
14:                break                          ▷ abort satisfied clauses in constant time
15:            end if
16:            found-no-replacement = true
17:            for index ∈ {1, ..., |c|} do       ▷ assume match literal is watch-index(c, 1)
18:                w ← watch-index(c, 1) + index  mod |c|
19:                y ≡ u ← c[w]                   ▷ watch-index(c, 1) corresponds to falsified watch literal
20:                if watch-index(c, 1) ≡ w then
21:                    continue
22:                else if is-unassigned(y) then
23:                    watch-index(c, 1) ← w
24:                    update watch literal from x ≡ v̂ to y ≡ u in clause c
25:                    found-no-replacement = false
26:                    break
27:                end if
28:            end for
29:            if found-no-replacement then
30:                z ≡ v_z ← c[watch-index(c, 2)]
31:                if is-unassigned(z) then
32:                    path ← path · constraint-propagate(z ≡ v_z))
33:                else if ¬is-unassigned(z) ∧ z ≠ v_z then
34:                    return 0.0                 ▷ there is a conflict and we need to abort!
35:                end if
36:            end if
37:        end for
38:    end for
39:    return path
40: end procedure
```

6.3 Component Decomposition

Component decomposition was introduced by [5] in the context of #SAT for model counting. The idea is given a graph G, 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.

The connected subgraph is defined such that each vertex is reachable from every other vertex. The connected subgraph is maximal when it is the largest such set. All the connected subgraphs of G can be computed in linear time, $O(|X| + |C|)$, in the size of the graph as shown in [36]. Additionally, we need to maintain the relative quantifier ordering between variables within each component.

Initially component decomposition was only developed as a pre-optimization procedure that is applied on the instance once before the solver attempts to solve the problem. Each connected subgraph in the constraint graph represents a component or a completely independent subproblem from the original satisfiability instance. We apply the procedure recursively after each decision variable is assigned and the problem is simplified by constraint propagation.

Given 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

$$\Pr(X) = \prod_{i}^{|G|} \Pr(X_i) \tag{6.3}$$

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. Here, there is room for different optimization choices by changing the order in which subproblems are solved even though it has no effect on the correctness. Although it has no effect on correctness, we order subproblems by the most constrained subproblem first or incrementally find the components. Note that if any of the components are unsatisfiable, we can backtrack early and the original problem is unsatisfiable as well.

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 [41, 23].

6.3.1 Example

As an example, consider the problem (6.4) below:

$$F = \exists x_8 \mathfrak{U} x_3 \exists x_7 \mathfrak{U} x_{14} \exists x_{11} \mathfrak{U} x_{23} \mathfrak{U} x_5 \tag{6.4}$$
$$(x_7 \equiv 0 \vee x_{14} \equiv 1 \vee x_3 \equiv 0) \wedge (x_{11} \equiv 1 \vee x_{23} \equiv 0 \vee x_3 \equiv 1) \wedge (x_5 \equiv 2 \vee x_8 \equiv 1)$$

where the randomized variables have distributions:

$$\Pr(x_5) \in \{\frac{3}{5}, \frac{1}{5}, \frac{1}{5}\}, \Pr(x_{23}) \in \{\frac{1}{5}, \frac{4}{5}\} \tag{6.5}$$
$$\Pr(x_{14}) \in \{\frac{4}{5}, \frac{1}{5}\}, \Pr(x_3) \in \{\frac{1}{4}, \frac{3}{4}\} \tag{6.6}$$

Initially, we can decompose the problem into two independent components that share no variable as shown in Figure 6.2:

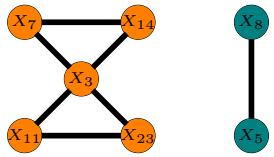


Figure 6.2: The constrained graph of a problem decomposed into a joint set of components.

where the analytical form of the components are given below:

1. $\exists x_8 \mathfrak{U} x_5 (x_5 \equiv 2 \vee x_8 \equiv 1)$
2. $\mathfrak{U} x_3 \exists x_7 \mathfrak{U} x_{14} \exists x_{11} \mathfrak{U} x_{23} (x_7 \equiv 0 \vee x_{14} \equiv 1 \vee x_3 \equiv 0) \wedge (x_{11} \equiv 1 \vee x_{23} \equiv 0 \vee x_3 \equiv 1)$
   (a) $\exists x_7, \mathfrak{U} x_{14} (x_7 \equiv 0 \vee x_{14} \equiv 1)$
   (b) $\exists x_{11} \mathfrak{U} x_{23} (x_{11} \equiv 1 \vee x_{23} \equiv 0)$

If we evaluate variable x_3 next depending on the value then component 2 can be further divided into two more components shown in (2a) and (2b). In the final result, this gives us 3 components to solve that are individually easier than the original problem.

6.4 Component Caching

Memorizing solutions of E-MAJSAT problems dates back to [58] and specifically component caching has been used by [3] in model counting for #SAT. A proof system was later developed by [6]. A huge advantage of caching is that a component produced in one branch of the search space can reappear numerous times throughout the search space and we only need to find a solution once. We show an extension to SSAT. Given a component, $G_k = \{X_k, C_k\}$, that is defined by a set of variables X_k and clauses C_k. Note that all variables will be unassigned and clauses unsatisfied where clauses contain only mentions of variables in X_k.

Assume we are given a hash-table, H, to be used as cache. We first compute a hash of G_k by using the active clauses and whenever we encounter a new subproblem in the future, we consult the cache first to see if we've already solved it and return the solution else compute its value and store the result in the cache as well.

6.4.1 LRU Cache

In larger problems there are many more subproblems to be stored than available memory. In general, basic caching of components is a memory expensive technique and usually leads to the exhaustion of main memory. One approach is to limit the number of components in the cache and to replace components as the need arise. There are many replacement policies studied in general, but the simple Least Recently Used (LRU) policy will be the approach we take based on results from [58].

6.4.2 LRU-sizeof Cache

Even with a LRU cache of a fixed size there is still the issue of getting the number of components just right such that you take advantage of all available memory without exhausting memory. In general, this is extremely difficult to achieve since individual problems have varying component distributions.

Instead, if we place a limit on the size of the memory that the cache and all it's components consume, then for a system with limited memory we can solve a variety of problems without ever exhausting memory. Whenever we add a new component, check if it will exceed our memory limit then free a set of stored components according to the LRU policy until the total size of the cache is under the limit.

6.5 Symmetry

The idea of 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.

A symmetry of variables can arise from interchangeability in the original domain over objects. When interchangeability is encoded into CNF, it generates symmetries for each possible permutation. Furthermore, symmetry of values can occur when mapping finite domains to binary values and the values are exchangeable with no proper ordering.

In [21] a general technique to break symmetries was introduced by imposing a lexicographic ordering to the variables that form a symmetric group. Later, [75] used element constraints to break symmetries on variables. This is shown to reduce the set to a single solution by ruling out unsorted solutions. However, in [95] they show that it is in general intractable to remove all the symmetries from a group for a sufficiently complex domain. In [73, 74, 9] Puget outlined further techniques to break symmetries for CSPs.

6.5.1 Canonical Representation

Usually, breaking symmetries requires adding 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. Our work will focus on dynamic or conditional symmetries that occur during the execution of the solver.

Although the combination of component decomposition and component caching gives major speed improvements there are unfortunately many redundant components that are symmetric and therefore have to be re-solved for each problem. Planning problems encoded into satisfiability variables for different timesteps usually share the same structure and randomized variables share the same distribution. Together these structures produce many overlapping subproblems that are symmetric.

Therefore, we could actually do better by removing symmetric components. That is many of the redundant components are permutations of the variables or clauses in a different position since we are comparing at the syntax level. By extension, the use of symmetries

makes component decomposition and caching more efficient since more components will be likely to match a component already in the cache and hence consume less space in the cache.

This requires further integration of symmetry detection and symmetry breaking into the solver. In [41] 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.

Furthermore, [41] showed that it is best to represent the component as a graph that is in a normal form invariant to any symmetries. The advantage of this approach is that we can encode variable and value symmetries explicitly into the graph. The graph isomorphism package by McKay, nauty [62], is usually used for graph labeling and the package called saucy [22] presents an improvement that focuses on sparse graphs generated by CNF encodings.

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.

Consider the previous example from Section 6.3.1 that we were able to reduce into 3 separate components. If we apply the transformation above, we get Figure 6.3:

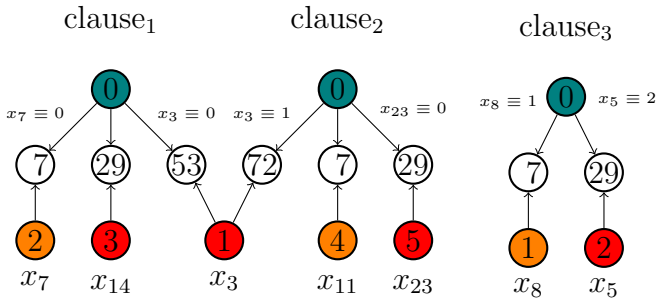


Figure 6.3: Representation of Eq. (6.3.1) in canonical form.

where the problem is decomposed into 2 independent components. The 1st subproblem with clause$_1$ and clause$_2$ can be solved first by assigning x_3 according to the quantifier ordering of the variables. Which leads to the components in Figure 6.4.

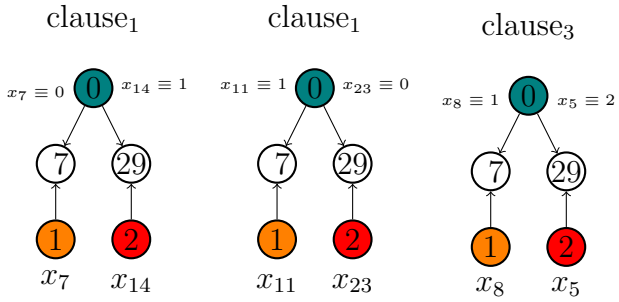


Figure 6.4: Components of Eq. (6.3.1) in canonical form after assigning x_3.

Although the components are syntactically different in the graph encoding they have the same labeling and hence are equivalent. Therefore, the problem could be represented by just one component.

6.5.2 Component Projection

Sparse graph canonization is usually fast for one time events (on the order of a few seconds) on large scale graphs, but this becomes questionable for hundreds of thousands to millions

of calls per problem. We propose Component Projection, which is a simplification of graph canonization that only does the mapping to a coloured graph without graph relabeling.

The reasoning is that the most expensive part of the encoding is not present and moreover 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 as helpful for most problems. Especially since the restriction on quantifier ordering limits the order of variables in a subproblem.

6.6 Branch and Bound

Bounds on the optimal probability of satisfiability is another useful technique that can be applied to SSAT problems. Assume the current partial assignment is σ and the probability of the current variable assignment is α. If we are evaluating variable x with quantifier $Q(x) = \mathfrak{H}$ and the value of the current value tried is $\Pr(x = v_i|\sigma) = p_i$ then we can maintain a lower and upper bound on the global probability by updating the bounds using the rules below:

$$LB(\sigma) := LB(\sigma) + \alpha p_i, \text{if } \sigma \text{ is a solution} \tag{6.7}$$

$$UB(\sigma) := UB(\sigma) - \alpha(1 - p_i), \text{otherwise} \tag{6.8}$$

Initially, LB is set to 0 and UB is set to 1. These lower and upper bound rules are guaranteed to converge to the correct probability as long as there are only randomized quantifiers.

6.7 Summary

In summary, I introduced my solver, SSAT-Prime, that solves general SSAT problems using DPLL inspired methods from the SAT, #SAT and SSAT communities. An improvement to the two-watch literal scheme for unit propagation is shown where searching through already satisfied clauses is avoided. Component decomposition is exploited dynamically during execution of the solver to detect components that would only appear conditioned on certain assignments.

In addition, the components are stored in a cache for reuse whenever possible. This means that components only need to be solved once and could be used in a different context. In spite of the improvements made, many problems lead to components that are

symmetric to many other components. They may be just a permutation of the variables or different variables all together. A symmetry detection scheme was used to match equivalent components in the cache dynamically. Finally, a simple scheme to track the upper and lower bounds on a subset of SSAT problems was given.

Chapter 7

Experiments

In this chapter, I show some experimental results on improvements made to the solver and run some tests using the encodings developed in the previous chapters to determine their practicality. My improvements include a faster way to detect unit clauses as described in Section 7.1.1, an improved caching policy described in Section 7.1.2, detecting symmetries described in Section 7.1.3, and computing an estimate of the upperbound described in Section 7.1.4.

Furthermore, the results for the encoding from Section 5.3 are shown in Section 7.2 for inference problems. In Section 5.1, I showed an encoding that allowed me to transform POMDPs to SSAT and test the Prime solver on solving some benchmark problems in Section 7.3. Finally, I generate instances from a random distribution and show some results regarding the relative difficulty of solving random SSAT.

Before we begin, note that 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.

7.1 Improvements

Many of the techniques implemented in my solver are borrowed from the SAT, #SAT and CSP communities. However, I have made some improvements and extension for SSAT and in the next sections I describe some experiments to infer what benefits, if any, are achieved.

I test the improvements on a variety of problems from 3 different benchmark types as shown in Table 7.1. The random benchmarks tend to be a series of variables with

alternating quantifiers in 3-SAT and 10-SAT forms. The POMDP problems consist of two easy and two hard problems that have quite a large number of literals per clause and variable cardinality. Finally, the inference problems tend to be highly structured and contain a large number of variables and clauses. I try to test on the same problems throughout the chapter for a better understanding of the differences that each feature induces.

Benchmark	Problem	#var	#clause	avg #value	avg #literal
RANDOM	fail-learn1.ssat	50	120	2.00	3.00
	pure1.ssat	50	120	2.00	3.00
	big1.ssat	30	450	2.00	10.00
	big2.ssat	15	60	4.00	10.00
POMDP	tiger.95.H10	157	304	2.31	5.60
	ejs7.H10	121	212	2.16	4.58
	query.s4.H2	657	27,868	42.68	160.40
	aloha.10.H3	1,094	18,637	17.14	64.39
INFERENCE	mastermind_04_08..	6,319	14,670	2.00	2.90
	fs-29.uai	327,787	803,068	2.00	2.74

Table 7.1: Basic information for each benchmark problem.

7.1.1 Unit Rule

In Section 6.2, I proposed an improvement to the watch literal rule by keeping a lazy structure that allows us to determine in constant time if a clause is satisfied. This was necessary to perform component decomposition efficiently if we wanted to continue using watch literals. I did not consider alternative unit rule variants that relied on updating a counter or requires extra work during backtracking.

In Table 7.2, I show the results of both unit rule algorithms on a variety of problems. UR-Improved is my new algorithm and Unit-Rule is the original implementation. My approach yielded an improvement in time (seconds) across all benchmarks and 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.

Benchmark	Problem	Unit-Rule	UR-Improved	Speedup	Satisfied
RANDOM	fail-learn1.ssat	2.79	**2.73**	1.02	64.68%
	pure1.ssat	2.91	**2.85**	1.02	64.92%
	big1.ssat	72.03	**63.88**	1.13	99.24%
	big2.ssat	8.36	**6.69**	1.25	85.38%
POMDP	tiger.95.POMDP.H10	4.79	**4.67**	1.03	88.55%
	ejs7.POMDP.H10	64.35	**63.56**	1.01	81.08%
	query.s4.POMDP.H2	113.56	**21.18**	5.36	99.99%
	aloha.10.POMDP.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 7.2: Improvement to the watch literal rule.

7.1.2 Component Caching

In previous work, a memory was introduced to store the solutions of re-occurring subproblems in the context of #SAT. In SSAT problems, any fixed memory could eventually be exhausted so alternative caching policies were considered based on the simplicity of Least Recently Used (LRU) policy. Table 7.3 compares (1) Cache-Unrestricted (CU) that adds components to memory until it is exhausted (2) Cache-LRU (CLRU) that implements the LRU policy using a fixed number of components and (3) Cache-LRU-Sizeof (CLRUS) that uses a fixed space in memory for LRU based on the size of each component. The duration to solve each problem is measured in seconds, Cache Miss is the number of times the component was not found in the cache and % Cache Miss is the percentage of requests to the cache that resulted in no hit or were unsuccessful in finding a matching component.

On the RANDOM benchmarks, CLRU performed the best by having the best times on the 4 problems and the number of cache miss were comparable except on big1 where CLRUS had a lot more misses. However, on the POMDP and INFERENCE benchmarks both CU and CLRU timeout on larger problems where some components could not be fit in memory (CU) and even with a fixed size cache, memory was still exhausted because of the cumulative size of each component. In general, finding the right size cache for CLRU is quite a challenge since one setting might not work for other benchmark types. In fact, we observed that different benchmarks usually have varying distributions for the number of components used and their size.

Problem	Cache	Duration	Cache Miss	Percent Cache Miss
fail-learn1	CU	2.53	184,600	43.52%
	CLRU	**2.46**	184,600	43.52%
	CLRUS	2.72	191,627	43.26%
pure1	CU	2.73	211,475	44.63%
	CLRU	**2.65**	211,475	44.63%
	CLRUS	2.86	211,539	44.63%
big1	CU	50.60	4,412,327	38.34%
	CLRU	**48.33**	4,412,327	38.34%
	CLRUS	64.19	5,130,325	37.86%
big2	CU	6.33	676,853	72.01%
	CLRU	**6.05**	676,853	72.01%
	CLRUS	6.68	685,673	72.39%
tiger.95.H10	CU	4.49	118,095	10.62%
	CLRU	**4.42**	118,095	10.62%
	CLRUS	4.68	118,095	10.62%
ejs7.H10	CU	X	X	X
	CLRU	X	X	X
	CLRUS	**63.71**	1,398,111	18.05%
query.s4.H2	CU	**21.00**	398	0.02%
	CLRU	21.12	398	0.02%
	CLRUS	21.07	398	0.02%
aloha.10.H3	CU	6.69	1,552	2.68%
	CLRU	**6.67**	1,552	2.68%
	CLRUS	**6.67**	1,552	2.68%
mastermind_04_08..	CU	X	X	X
	CLRU	X	X	X
	CLRUS	**27.67**	42,620	61.57%
fs-29	CU	**11.72**	1,332	0.52%
	CLRU	11.75	1,332	0.52%
	CLRUS	12.07	1,357	0.53%

Table 7.3: Different improvements to the cache.

7.1.3 Symmetry

In this section, I compare the effect of using symmetry. First, I 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 shown in Table 7.4 where duration is in seconds, no C. indicates the number of components generated and the last column indicates the percentage of reused components.

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 and among the remaining problems, CP performed the best by a significant margin of up to x100 speed improvement. In general, we found the results hold across most problems from the POMDP benchmarks. It should be noted that there is only a huge improvement over CB 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. Therefore, the cost of CC was too high, especially on the RANDOM and INFERENCE benchmarks.

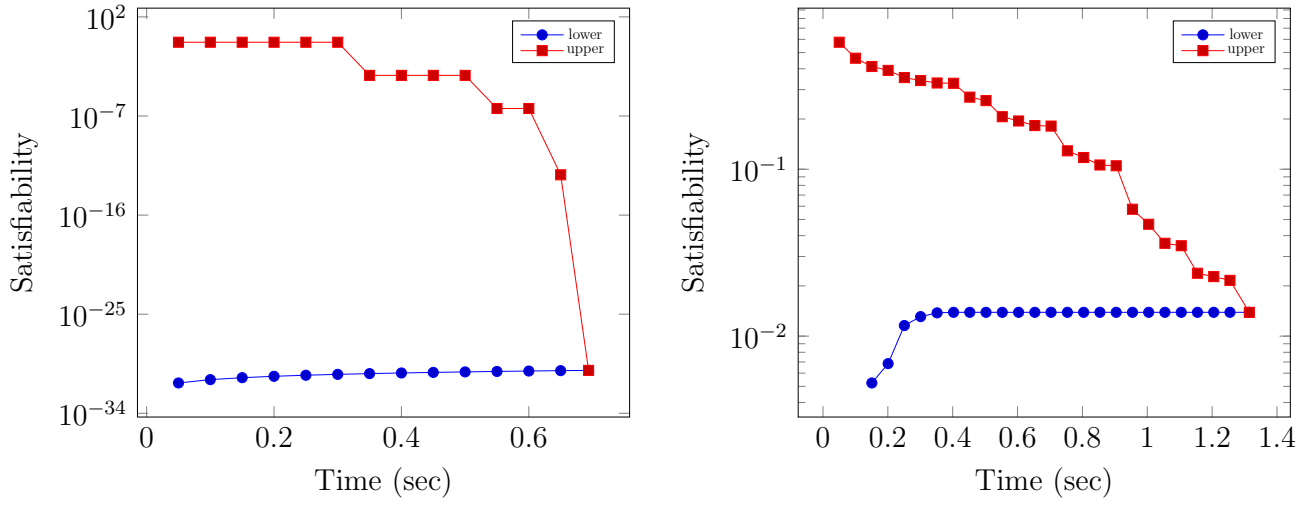
7.1.4 Upper Bound

Up until now, the solver has mainly focused on refining the lower bound incrementally, but in Section 6.6, we introduced a way to also track incremental improvements to the upper bound. The results are shown in Figure 7.1 for two problems from the INFERENCE benchmark (a) fs-10 and (b) variant of blockmap. The x-axis is time in seconds and probability of satisfiability is shown on the y-axis in log scale.

In both problems, the lower bound quickly converges, but the upper bound requires more refinement. This holds for most inference problems since the lower bound is only summing a few satisfying solutions, however the upper bound has to reject all other solutions. Unfortunately, in much larger problems we observe some numerical instability, so this will not be a general solution.

Problem	Symmetry	Duration	No. C	Percentage 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 7.4: Results for improvement in symmetry by Canonical and Projection relabeling.



(a) Bound for fs-10

(b) Bound for blockmap_10_02-0013

Figure 7.1: Convergence of lower and upper bound on two probabilistic problems.

7.1.5 Extra Techniques

There are a variety of features we have not implemented fully, but two in particular will be the focus of this section. These are pure literal elimination and clause learning.

The benefit of the pure literal elimination rule is usually quite significant when it can be performed efficiently in traditional SAT environments. Specifically, when there is a significant number of existential variables, pure literal elimination can be effective. Unfortunately for randomly quantified variables there is no equivalent reduction rule. This automatically rules out improvements in inference related problems. In POMDP problems, only action variables are existentially quantified which usually cannot be implied by unit clause. Therefore, pure literal elimination is a poor fit for the type of problems that we are interested in solving.

I did not implement clause learning in this version of the solver. The main reason being that I am working with finite domains and I do not have an explicit representation for negation. To perform clause learning, I must first determine when a conflict occurs and then extract a succinct clause that summarizes the reason for the conflict. In Table 7.5, I show the number of conflicts encountered in the problems I used for different benchmarks. I note that the POMDP and INFERENCE benchmarks usually have significantly fewer conflicts compared to RANDOM problems.

Benchmark	Problem	#conflict
RANDOM	fail-learn1.ssat	77,702
	pure1.ssat	64,125
	big1.ssat	32,751
	big2.ssat	1,212,635
POMDP	tiger.95.H10	1
	ejs7.H10	1
	query.s4.H2	865
	aloha.10.H3	34,686
INFERENCE	mastermind_04_08..	37,047
	fs-29.uai	1

Table 7.5: Basic information for each benchmark type.

7.1.6 Summary

Going forward, I incorporate the key features discussed earlier as default configuration into the solver, including

- Improved Watch Literal: the changes to the watch literal rule was at least an improvement in every benchmark tested on with larger gains on the POMDP problems.
- Cache LRU Sizeof: the added time for managing the cache and recursively calculating the size of each item was reasonable and yields more stability memory wise on larger problems.
- CP: even though its running times were greater than CB for all but one problem, it provides the most potential, especially on POMDP benchmarks where the observed symmetry could be beneficial.
- No upper bound: this feature is turned off by default since it only works on a subset of SSAT problems.

7.2 Inference Competition

In the following section, I test the solver on a benchmark from the 2008 Inference Competition. The benchmark data is a set of relational Bayesian networks constructed with the Primula tool. There are 251 networks that are all binary variables. The networks contain quite large tree-widths, however there are high levels of structure and determinism to compensate. That is, most problems contain a significant number of deterministic variables.

#Problem	Name	min #var	max #var
150	Blockmap	700	59,404
80	Mastermind	1,220	3,692
11	Friends & Smoker	10	76,212
10	Students	376	376

Table 7.6: The composition of the 251 problems in the Relational benchmark from the Inference competition.

7.2.1 Results

The results are shown in Figure 7.2. The solvers have 1000 seconds to solve each problem and the problems are sorted by run time from shortest to longest. Problems that were not solved are shown by a flat line at the end. In the competition, the winner is usually the solver that solves the most problems and ties are broken by the cumulative time.

Overall, ace was the clear champion of the 2008 competition by a wide margin of 20 more problems solved than the 2nd place solver. In comparison, I re-ran Ace on the same machine as my solver. My solver, ssat-prime, was able to solve 234 problems in $14,691.73$ seconds compared to 228 for ace in $14,691$ seconds. All of the failed problems for SSAT-Prime were from the Mastermind problem set where as the failed problems for ace were from the Blockmap problem set.

7.2.2 Summary

In the previous section, I demonstrated how to solve probabilistic inference problems encoded as SSAT problems from the 2008 Inference competition against the past winner, ACE, which uses model counting. My contribution is achieving state of the art results on

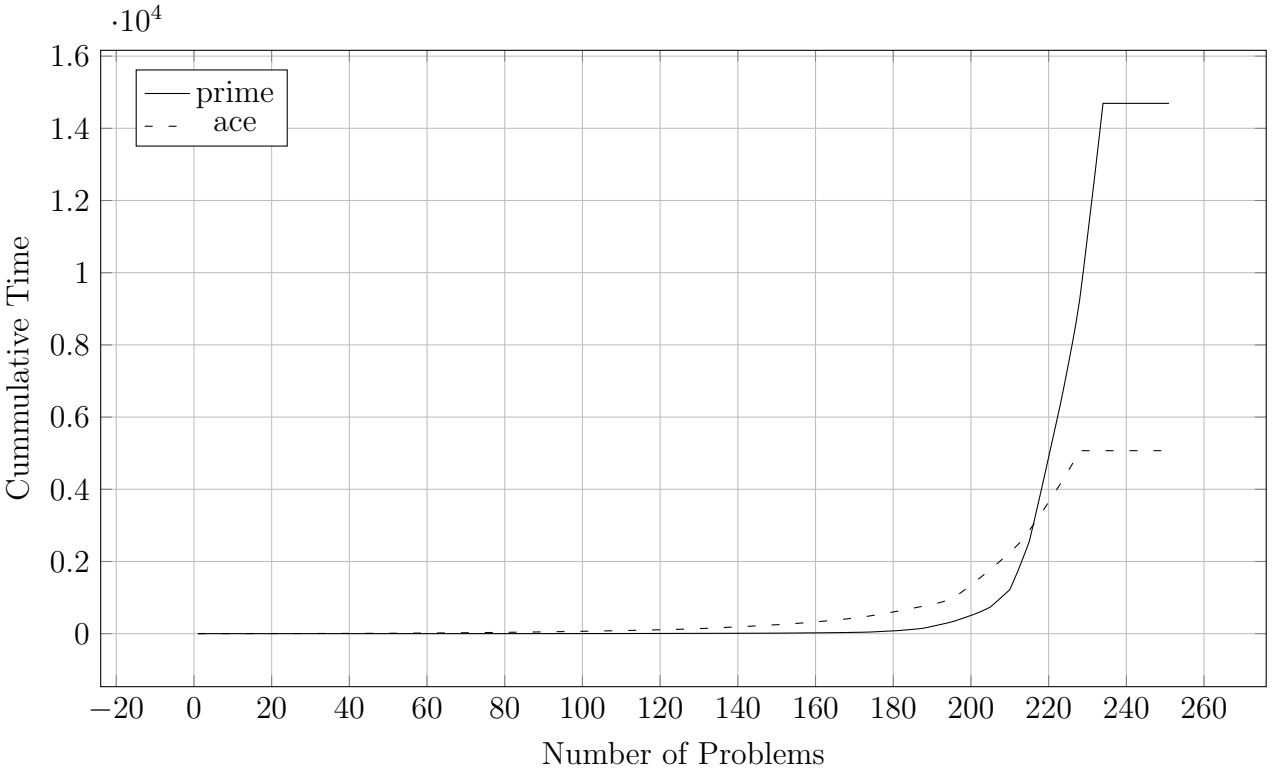


Figure 7.2: Cumulative time on inference benchmark from 2008 Competition

these benchmarks. Solving inference problems as SSAT is very promising in that I am now able to solve more problems and demonstrate scalability of SSAT-Prime on large networks in comparison to other techniques.

One of the advantages of ACE and similar solvers that use compilation is that after a steep up front cost for compilation, all further queries can be answered in linear time with respect to the compiled representation. In the future, I would like to amortize the cost of computing solutions to components and use that to speed up the performance across multiple queries in similar time.

7.3 POMDP Benchmarks

Next, I show some results on POMDP problems from the literature, Cassandra's thesis, and a collection of other problems [17]. Various subsets of these have been used by other researchers to benchmark POMDP solvers. However, there is a large portion of those problems that remains unsolved due to being intractable such as pentagon, milos or forth. I will compare a solver from [17] that is the classical POMDP solver using incremental pruning (PRUNE), my solver ssat-prime (PRIME) and a previous exact SSAT solver (ZANDER). PRUNE is configured to use a discount of 1.0 and a finite horizon (not undiscounted case). Since ZANDER only works with binary variables the problems were re-encoded to binary variables and to ZANDER's format, which increases the complexity of the problems. It was verified that all solvers found equivalent optimal policies and the correctness of the encoding was also checked empirically.

7.3.1 Results

Now we compare native POMDP solvers against SSAT solvers on POMDP problems. The results are in Table 7.7 where the first column is the problem name followed by the number of actions, states and observations and then the results for the three solvers. The number in each column is the maximum horizon (limit of 20) each solver is able to successfully solve in a time limit of 1000 seconds. First, we notice that PRUNE was far superior to the SSAT solvers and the number of times each solver reaches the maximum horizon (20) for each problem is (45 for PRUNE, 27 for SSAT-PRIME, 1 for ZANDER). PRIME was not able to quickly solve the simpler problems but tend to perform reasonably well on the tougher problems such as *learning* and *query*. ZANDER's results are poor for most of the problems except for a few cases. The big issue was memory since ZANDER works by constructing optimal subproblems into larger building blocks. Most of the problems were quite large after being converted into a binary encoding.

PRUNE was able to quickly solve problems that could be represented compactly. However, if we remove the smaller benchmark sets such as *cheng* (11 problems) and *ejs* (7 problems) the PRUNE and PRIME solvers are surprisingly closer with 28 and 24 problems solved to the maximum horizon respectively. It is clear that problems where a low number of alpha vectors is sufficient to represent the optimal value function correspond to problems that PRUNE does well on.

problems	$\|A\|$	$\|S\|$	$\|O\|$	prune	prime	zander
1d	2	4	2	**20**	14	0
4x3.95	4	11	6	**10**	4	0
4x4.95	4	16	2	0	**6**	0
4x5x2.95	4	39	4	0	**4**	0
aloha.10	9	30	3	3	**4**	0
aloha.30	29	90	3	0	**2**	0
baseball	6	7681	9	0	**1**	0
bridge-repair	12	5	5	**6**	4	0
bulkhead.A	6	10	6	**11**	4	0
cheese.95	4	11	7	**20**	4	0
cheng.D3-1	3	3	3	**20**	7	1
cheng.D3-2	3	3	3	**20**	7	1
cheng.D3-3	3	3	3	**20**	7	1
cheng.D3-4	3	3	3	**20**	7	1
cheng.D3-5	3	3	3	**20**	7	1
cheng.D4-1	4	4	4	**20**	5	0
cheng.D4-2	4	4	4	**20**	5	0
cheng.D4-3	4	4	4	**20**	5	0
cheng.D4-4	4	4	4	**20**	5	0
cheng.D4-5	4	4	4	**20**	5	0
cheng.D5-1	3	5	3	**20**	6	0
cit	4	284	28	0	**2**	0
concert	3	2	2	**20**	9	9
ejs1	4	3	2	**20**	8	1
ejs2	2	2	2	**20**	12	0
ejs3	2	2	2	**20**	11	0
ejs4	2	3	2	**20**	11	1
ejs5	2	2	2	**20**	**20**	11
ejs6	2	2	2	**20**	**20**	11
ejs7	2	2	2	0	**11**	**11**
ejs-ft-counter	2	2	2	**20**	11	0
fourth	4	1052	28	0	**1**	0
hallway2	5	92	17	**2**	**2**	0
hallway	5	60	21	**3**	2	0
hanks.95	4	4	2	**20**	8	0
iff	4	104	22	1	**2**	0
learning.c2	8	12	3	2	**4**	0
learning.c3	12	24	3	2	**3**	0
learning.c4	16	48	3	1	**2**	0
line4-2goals	2	4	1	0	**19**	2
machine	4	256	16	0	**2**	0
marking2	4	9	3	**20**	6	0
marking	4	9	3	**20**	6	0
mcc-example1	3	4	3	**20**	7	1
mcc-example2	3	4	3	**20**	7	1
milos-aaai97	6	20	8	2	**4**	0
mini-hall2	3	13	9	**20**	4	0
mit	4	204	28	0	**2**	0
network.95	4	7	2	**20**	6	0
network	4	7	2	**20**	6	0
paint.95	4	4	2	**20**	8	0
parr95.95	3	7	6	**20**	6	0
pentagon	4	212	28	0	**2**	0
query.s2	2	9	3	5	**7**	0
query.s3	3	27	3	3	**4**	0
query.s4	4	81	3	**2**	**2**	0
saci-s100-a10-z31	10	100	31	**2**	1	0
saci-s12-a6-z5.95	6	12	5	**6**	4	0
shuttle.95	3	8	5	**10**	5	0
stand-tiger.95	4	4	4	**20**	7	0
sunysb	4	300	28	0	**2**	0
test-simple	2	4	1	0	**20**	0
tiger.95	3	2	2	**20**	14	9
tiger.aaai	3	2	2	**20**	14	9
tiger-grid	5	36	17	**2**	**2**	0
web-ad	3	4	5	5	**6**	1
web-mall	3	2	2	**20**	14	9

Table 7.7: Solving POMDP problems with a native solver PRUNE compared to encoding into SSAT and using PRIME and ZANDER.

7.3.2 Summary

In this section, I compared my solver against a native POMDP solver, PRUNE and another SSAT solver ZANDER over many common POMDP benchmark problems. My solver is not quite competitive against PRUNE yet, but it fares well against ZANDER. Overall, my contribution is in developing a solver that is able to find solutions to POMDPs encoded as SSAT and showing that it is possible. Furthermore, my solver is competitive on the larger benchmarks, which shows progress in the right direction. PRUNE is better to find solutions to problems that can be accurately represented by a small set of alpha vectors. In future work, it will be key to represent the policies of problems more compactly and to quickly find an optimal policy.

7.4 Satisfiability Benchmarks

I also attempted to show experimental results for converting satisfiability (SAT, QBF and SSAT) problems to POMDP and applying a modified version of the GapMin [71] solver. For SAT problems the planning horizon was too large for POMDP solvers in comparison to state of the art SAT solvers.

However, for QBF problems encoded as POMDPs we require that the value of the policy be equal to 1 for satisfiable instances and less than 1 for unsat instances. Unfortunately, due to numerical instability induced by summing over exponentially many sequences of observations that each have tiny probabilities (corresponding to universally quantified variables in QBF) the correctness cannot be guaranteed for horizons greater than 60. In some cases the value of the optimal policy is one, but our computed value is less than 1 (and vice versa).

7.5 Random Satisfiability

Finally, in the last section I study my solver's sensitivity to satisfiability problems generated according to different parameters and attempt to see how this affects the difficulty and probability of satisfaction. The phase transition phenomenon found in randomly generated SAT problems has been extensively studied in the literature. For k-SAT problems, if we let $\alpha = C/V$ and vary the number of variables and the number of clauses, this parameter becomes a good indicator of the problem difficulty. There exists a threshold α_0 such that all problems with a smaller α are relatively easy to solve and all problems with a greater

α are also easier but a little harder. However, around the value α_0, problems become extremely difficult to solve for the current class of solvers. It has been empirically observed that the hardest problems for 3-SAT are at $\alpha_0 \approx 4.26$.

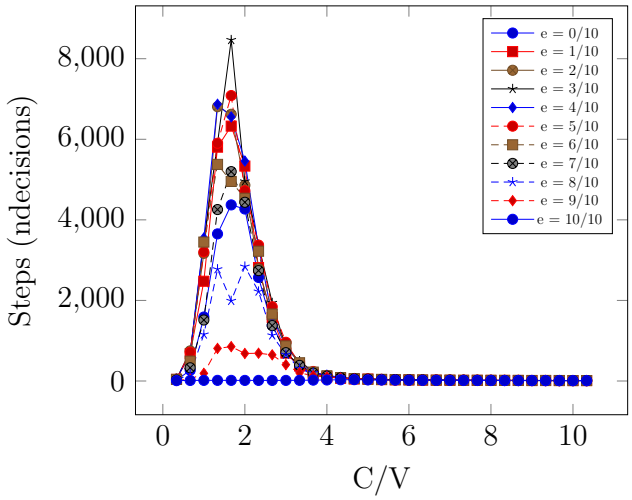
7.5.1 Results

For SSAT, we generate random problems for 3-SAT, 4-SAT, and 5-SAT parameterized by e where the ith variable, v_i, has probability e of being randomly quantified with a distribution over values that follows a uniform Dirichlet $Dir(1)$ and vary e from 0 to 1. Results are in Figure 7.3. They are averaged over 50 trials where by fixing the number of variables to 30, we generate problems with clauses within a range of $[10, 320]$ to get the x-axis. The y-axis of the graphs in the first column is the number of decisions required to solve each problem. The y-axis of the graphs in the second column is the probability of satisfaction.

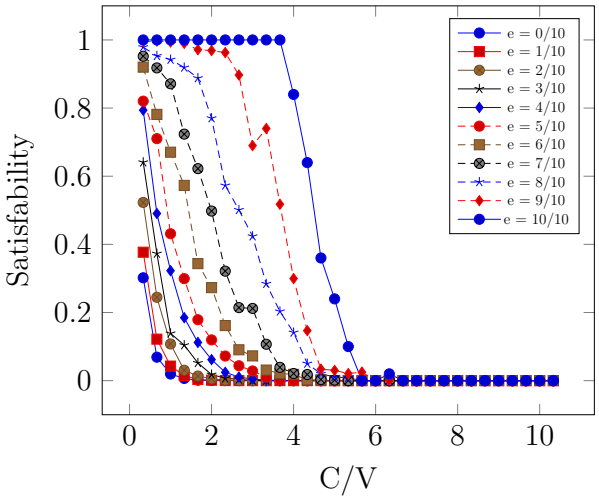
Similar to the SAT graphs, there is a peak corresponding to easy-hard-easy zones of difficulty and various values of e seem more difficult. We have 2 special cases where $e = 0.0$ corresponds to regular SAT in NP and is shown as a flat line since it's relatively easy to solve at this scale and $e = 1.0$ corresponds to probabilistic inference in PP. In general, the alternating sequence of quantifiers should be the most difficult at $e = 0.5$, even though there is no guarantee that every instance will have perfectly alternating quantifiers. In fact this is supported by the data. In (a) it is clear that there is only 1 maximum value for $\alpha = C/V$ however in (c) and (e) the peak is more noisy and there are signs of multi-modal distributions for some e values. But I believe this to be noise.

Alternatively, the probability of satisfaction in Column 2 shifts from being almost surely satisfiable with probability 1 to 0. Consider (b), in particular for $e = 0$ there is a phase shift at $\alpha = 4.26$ that is confirmed by the data. I found in general that for parameters e_i and e_j such that $e_i < e_j$ then the phase transition occurs between 0 and 4 with the constraint phase-shift(e_i) < phase-shift(e_j). For (d) I see a smoother decline in satisfiability over a much larger region, however in many cases the decline is not monotone and it is much more pronounced in (f).

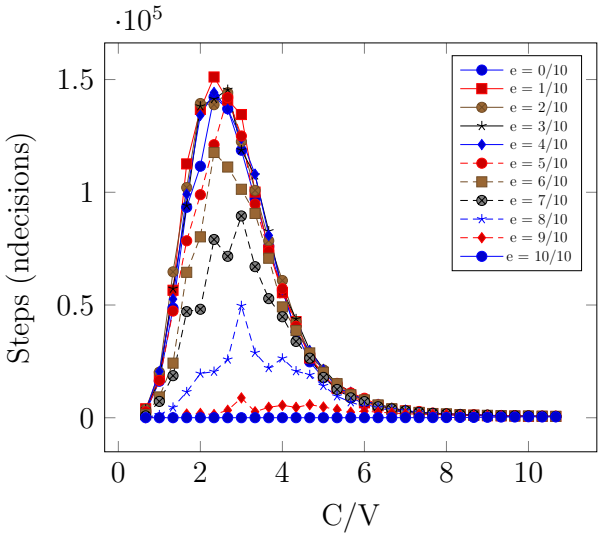
Estimates for the region in which problems are more difficult to solve are in Figure 7.4. I show a bounded range for the most difficult problems for each value of e on 3-SAT, 4-SAT, and 5-SAT problems.



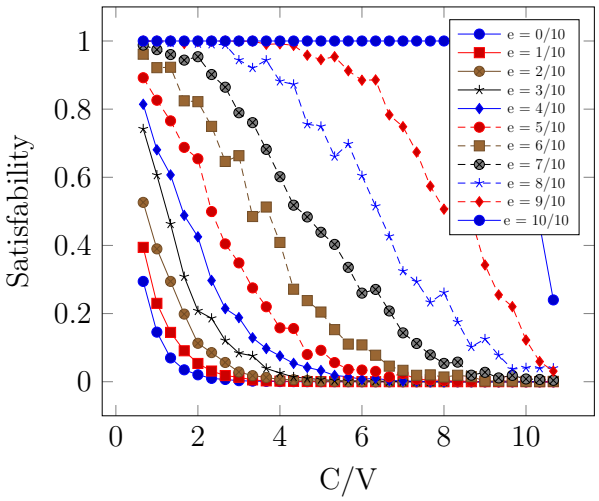
(a) 3-SAT Steps



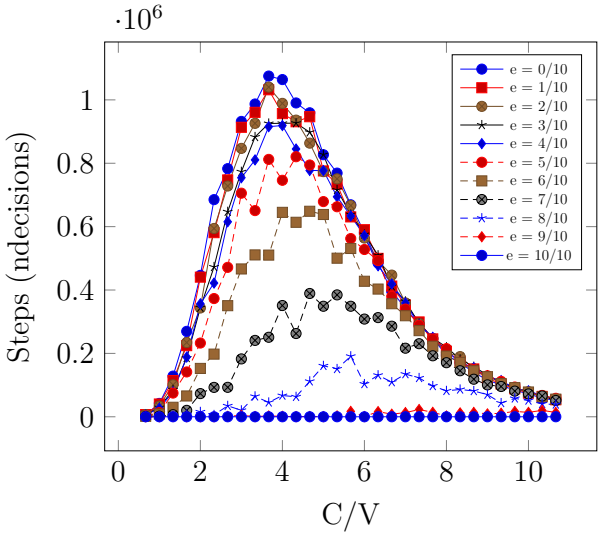
(b) 3-SAT Satisfiability



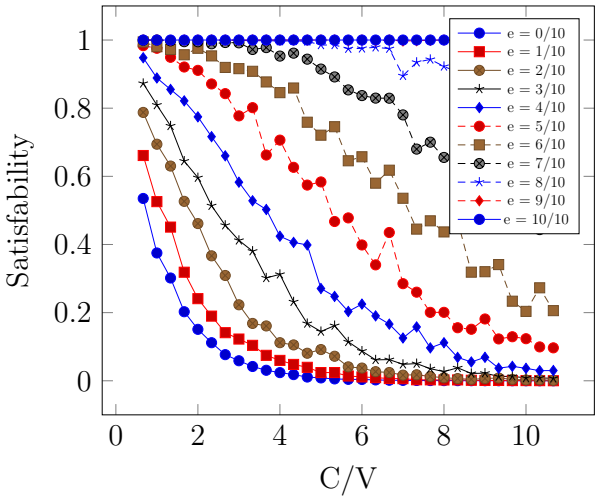
(c) 4-SAT Steps



(d) 4-SAT Satisfiability



(e) 5-SAT Steps



(f) 5-SAT Satisfiability

Figure 7.3: The number of steps and probability of satisfiability for stochastic 3-SAT, 4-SAT, and 5-SAT problems

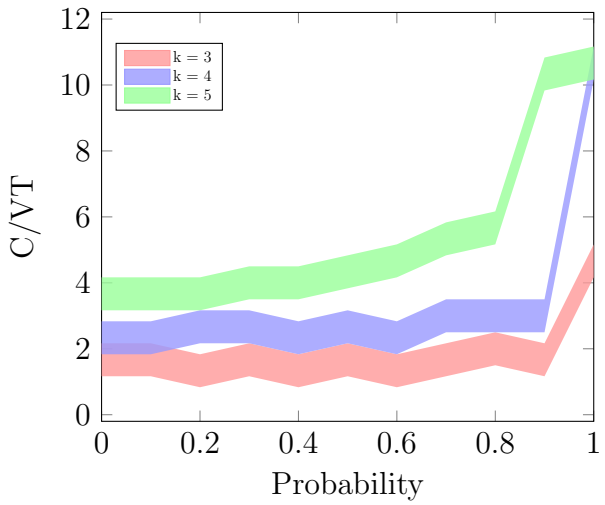


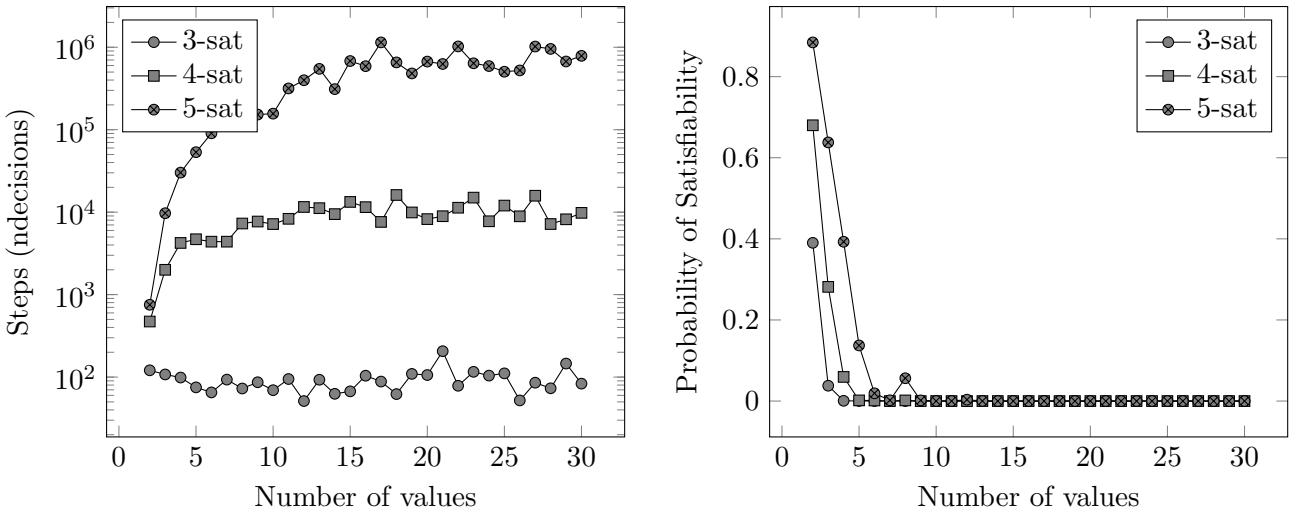
Figure 7.4: The difficulty threshold for stochastic 3-SAT, 4-SAT, and 5-SAT problems

Next, I study the effect of increasing the number of values for each variable in randomly generated SSAT problems. The data is generated in the same way as before, but with a fixed number of variables (15), number of clauses (30), and $e = 0.5$. In Figure 7.5, I show how the cardinality of the domain of the variables affects the computation. In (a) I report the number of steps necessary for different k-SAT problems as I vary the cardinality of the domain of each variable on a semi-log scale and in (b) I report the probability of satisfaction for the same problems.

I expected an exponential increase in difficulty as the cardinality changes. However, for 3-SAT the difficulty remains relatively stable even up to a cardinality of 30. For 4-SAT and especially 5-SAT there was a sharp rise in difficulty but it eventually settled into a stable difficulty level. In (b) it was expected that the satisfiability approaches 0 as the cardinality of the domain of the variables increases since in the limit values used could only satisfy one clause. In general, clauses with more literals lead to more satisfiability as we observe in Figure 7.5. However, after a cardinality of 8, satisfiability is almost zero.

7.5.2 Summary

I showed some interesting properties of satisfiability. By extending the quantifier to randomized variables, we witness an easy-hard-easy curve for randomly generated problems that is similar to the curve for random SAT problems. However, the curve is more complex by being multi-modal and the probability of satisfiability is also no longer a smooth



(a) Number of steps required to solve a problem.

(b) Probability of Satisfiability

Figure 7.5: The number of steps and the probability of satisfiability for stochastic SAT for variables with increasing number of values.

downward slope but a multi peak graph where the differences become more pronounced for problems with variables that have a uniform distribution over quantifier type.

7.6 Conclusion

In this chapter, I tested various features of my SSAT solver, SSAT-Prime, that demonstrated improvements over standard implementations. These include an improvement to the watch literal scheme, a new policy for caching components, detecting symmetries and an upper bound for inference problems.

I showed competitive results on benchmarks from the 2008 Inference Competition where I was able to solve more problems than the winning solver, ACE, which is using model counting techniques. Prime was also tested against a native solver and another SSAT solver on POMDP benchmarks, but the results were less convincing. After adjusting for the few small problems where performance was poor, Prime, showed promise on the larger benchmarks by being competitive. In general, the results were good for an initial comparison.

I conclude by stating that like model counting, using a SSAT solver for inference tasks

can achieve state of the art results. However, POMDP problems will require more improvements in the solver or the encoding.

Chapter 8

Conclusion and Future Work

In this chapter, I summarize the work that was done in this thesis regarding (1) the encodings, (2) my SSAT-PRIME solver and (3) the impact. Finally, I discuss some future work that could improve the encodings and the solver.

8.1 Conclusion

In summary, one of my contributions is in showing how to encode various satisfiability problems (SAT, QBF, SSAT) into flat POMDPs without any exponential blowup. Surprisingly, the results show that the number of clauses in satisfiability determines the number of states in POMDP problems and the number of variables corresponds to the horizon length. In the reverse direction, from POMDP to SSAT, I gave a constructive procedure to perform the transformation with a proof. Similarly, I showed an encoding of inference problems to SSAT.

Furthermore, I developed a SSAT solver, SSAT-Prime, that improves techniques such as watch literals, component caching, symmetry detection, and a valid upper bound for a subclass of problems. For watch literals, I can detect in constant time if a clause was previously satisfied using a set of lazy data structures that require minimal work in backtracking. Next, detecting components in a partially evaluated formula and caching components allows significant improvements in running time. This is more beneficial for SSAT problems where all the solutions must be enumerated. In fact, many of the solutions are permutations of particular variables and values. This led to symmetry detection and ways to generalize the representation of components used by mapping the problem to a canonical labeling of

the constraint graph. Finally, for the class of problems that corresponds to probabilistic inference, we have valid lower and upper bounds on the probability of satisfiability.

Overall, the solver was most competitive on a probabilistic inference task from the inference competition against the past winner. The results on POMDP encoded problems were very good compared to ZANDER (another SSAT solver), however against PRUNE (a native POMDP solver) PRIME was not competitive on some smaller problems where the optimal value function could be represented compactly by a small set of alpha vectors.

The overall impact of this research is a better understanding of the relationship between satisfiability and POMDP problems. Furthermore, with access to solvers in both domains, techniques from POMDP solvers could be used in SSAT solvers and vice versa. Some of these features may induce additional restrictions on the types of SSAT problems that are able to be solved. Finally, my solver SSAT-Prime is a modern solver than implements many of the important features from satisfiability. This can motivate industry applications and further research into SSAT problems that will expand the literature.

8.2 Future Work

I propose future work into two directions. First, to improve the PRIME solver through additional features or improvements to current solution techniques. Second, to extend the scope of the problems that can be solved by the theory through extensions and various encodings.

8.2.1 Improve Solver Efficiency

One of the most prolific features in modern satisfiability solvers is clause learning, however its efficiency gains do not translate well to SSAT problems. This can be partly explained by noting the difference in finding one solution compared to enumerating all solutions. It is not as beneficial to record a conflict.

Component Templates

In [41], the authors introduced the idea of component templates that showed significant speedups in detecting dynamic components that are cached. This is an important contribution since decomposing the constraint graph after every decision can be costly in general. There are clear benefits to detecting symmetry in components. In some cases, I

saw significant improvements in running time in spite of the overhead for some benchmark problems. There is room for improvement in further interleaving component decomposition and symmetries that would minimize the cost in cases when symmetries are absent.

Branch and Bound

Branch and bound is another useful technique that can be applied to SSAT problems. Assume the current partial assignment is σ. If we are evaluating variable x with quantifier $Q(x) = \exists$ and the probability of satisfiability for the first value tried is $\Pr(x = v_0|\sigma) = p_0$, then by maintaining lower and upper bounds we can backtrack earlier and thereby prune subtrees when the upper bound for $x = v_1$ is less than p_0.

The same idea can be used to backtrack earlier for universal quantifiers by updating a lower bound. Unfortunately, there is nothing equivalent for randomly quantified variables. Note that this is different from the thresholding technique introduced in [50]. There, parameters $\theta = (\theta_l, \theta_h)$ were used for the upper and lower bound probabilities of satisfaction. If the probability was not within this range, the search could terminate early. We are in fact calculating the exact probability of satisfaction regardless if its value is within some predefined range.

Learn Parameters

One of the obstacles for further adoption of this work is in domains where a model is not known or is difficult to extract. That is for a particular domain the constraints might be constant for each instance of a problem but the distribution of randomly quantified variables could change depending on values observed so far in the process.

As an example the process of belief monitoring in POMDPs where you are updating the probabilities of being in a state given a history of observations and actions. If these parameters are needed to make decisions, then the solver can learn these parameters based on observed values (either from existential or randomize variables).

Continuous Domains

In this section, the motivation is in solving planning problems and in particular POMDP problems. In spite of the flexibility of current theories in solving problems over discrete domains, there is a large set of industry problems from motor control to robot navigation where the state space or action space is continuous. Here, traditional discrete approaches

do not carry over and usually continuous spaces are discretized or function approximators are used.

In the case of continuous variables, function approximators can be embedded into the solver, which for computation will lead to an integral for randomly quantified variables and continuous optimization for existentially quantified variables. Therefore the computation for each quantifier type is now:

$$\exists x \; F(x) = \max_x F(x) \tag{8.1}$$

$$\text{ϒ}x \; F(x) = \int_{x_0}^{x_1} \Pr(x = v) F(x = v) \mathrm{d}v \tag{8.2}$$

POMDP Features

One of the original goals outlined at the start of this research was to look for ways to convert POMDP techniques into satisfiability and vice versa. Now with an encoding to go back and forth between the different problems and a better understanding it will be easier to see which features are beneficial.

8.2.2 Changing the Encoding space

In addition to changes made to the solver, we can improve the complexity of solving problems by changing the theory and encodings.

Inference Problems

Conceptually, the solver can handle additional types of inference that were discussed in Section 2.3, but this will require a change to the current encoding. My focus so far has been on computing the partition function and probability of query variables. This can be extended to compute the probability of query variables given some evidence by fixing the values of certain indicator variables.

Also, computing the most likely assignment to all variables given evidence can be encoded using a set of existential and randomized variables. Furthermore, both Bayesian networks and Markov networks can be reduced to the same encoding for inference. This will allow benchmark comparisons with a greater variety of solvers.

Min-Max Games

We can extend the expressiveness of the solver by including universally quantified variables without increasing the complexity class. This model, Extended SSAT (ESSAT) [50], will be more compact in representing various planning problems. The inter-mixing of existential, randomized and universal variables corresponds to two player adversarial games that perform alternating min-max functions with stochastic outcomes. Furthermore, ESSAT will be more compact in representing various planning problems in adversarial domains similar to SSAT in the stochastic domain.

Extensions of POMDPs

Various extensions of POMDPs could all be naturally encoded into SSAT and solved by the same solver. In comparison, a POMDP solver would require special adjustments. For instance, factored POMDP, and the associated dynamics could be represented with a similar encoding or non-stationary dynamics. Additionally, like inference, prior domain knowledge on actions or observations can be encoded as extra clauses.

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] K. J. Astrom. Optimal control of markov decision processes with incomplete state estimation. *J. Math. Anal. Appl.*, 10:174–20, 1965.

[3] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for# sat and bayesian inference. In *Foundations of computer science, 2003. proceedings. 44th annual ieee symposium on*, pages 340–351. IEEE, 2003.

[4] Tom Balyo, Marijn J. H. Heule, and Matti Jrvisalo. Proceedings of SAT competition 2016; solver and benchmark descriptions. In *SAT Competition 2016*. University of Helsinki, Department of Computer Science, 2016.

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

[6] Paul Beame, Russell Impagliazzo, Toniann Pitassi, and Nathan Segerlind. Memoization and dpll: Formula caching proof systems. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 248–259. IEEE, 2003.

[7] Dimitri Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2012.

[8] Dimitri Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[9] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.

[10] Elazar Birnbaum and Eliezer L Lozinskii. The good old davis-putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, 10:457–477, 1999.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

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

[13] Blai Bonet and Hector Geffner. Solving pomdps: Rtdp-bel vs. point-based algorithms. *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pages 1641–1646, 2009.

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

[15] Cassio Polpo De Campos and Fabio Gagliardi Cozman. The inferential complexity of bayesian and credal networks. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1313–1318, 2005.

[16] A. R. Cassandra. *Exact and approximate algorithms for partially observable Markov decision problems*. PhD thesis, Department of Computer Science, Brown University, 1988.

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

[18] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[19] H. T. Cheng. *Algorithms for partially observable Markov decision processes*. PhD thesis, University of British Columbia, 1988.

[20] 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.

[21] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. *KR*, 96:148–159, 1996.

[22] Paul Darga, Karem Sakallah, and Igor L. Markov. Faster symmetry discovery using sparsity of symmetriess. In *Proceedings of the 45th Design Automation Conference*, pages 149–154, 2008.

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

[24] Adnan Darwiche. A logical approach to factoring belief networks. *KR*, 2:409–420, 2002.

[25] Adnan Darwiche. New advances in compiling cnf into decomposable negation normal form. In *In ECAI*. Citeseer, 2004.

[26] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[27] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1060.

[28] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, 2004.

[29] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[30] J. Gill. omputational complexity of probabilistic turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[31] 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.

[32] Carla P Gomes, Ashish Sabharwal, and Bart Selman. Model counting. 2008.

[33] Eric A Hansen. An improved policy iteration algorithm for partially observable mdps. In *Advances in Neural Information Processing Systems*, pages 1015–1021, 1998.

[34] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[35] Matt Hoffman, Hendrik Kueck, Nando de Freitas, and Arnaud Doucet. New inference strategies for solving markov decision processes using reversible jump mcmc. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 223–231. AUAI Press, 2009.

[36] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

[37] H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI92)*, pages 359–363, 1992.

[38] Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

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

[40] Igor Kiselev and Pascal Poupart. Pomdp planning by marginal-map probabilistic inference in generative models. In *Proceedings of the 2014 AAMAS Workshop on Adaptive Learning Agents*, 2014.

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

[42] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[43] 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.

[44] Johan Kwisthout. The computational complexity of probabilistic inference. Technical report, Radboud University Nijmegen, 2011.

[45] 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.

[46] W. Li, P. Poupart, and P. van Beek. Exploiting structure in weighted model counting approaches to probabilistic inference. *Journal of Artificial Intelligence Research*, 40:729–765, 2011.

[47] Michael L. Littman. The witness algorithm: Solving partially observable markov decision processes. Technical report, 1994.

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

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

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

[51] Jun S. Liu. The collapsed gibbs sampler in bayesian computations with applications to a gene regulation problem. *Journal of the American Statistical Association*, 89(427):958–966, 1994.

[52] W. S. Lovejoy. Suboptimal policies with bounds for parameter adaptive decision processes. *Operations Research*, 41:583–599, 1993.

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

[54] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[55] Stephen M Majercik. Nonchronological backtracking in stochastic boolean satisfiability. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, pages 498–507. IEEE, 2004.

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

[57] 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.

[58] Stephen M Majercik and Michael L Littman. Using caching to solve larger probabilistic planning problems. In *AAAI/IAAI*, pages 954–959, 1998.

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

[60] Mausam and Andrey Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan & Claypool, 2012.

[61] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI97)*, pages 321–326, 1997.

[62] Brendan McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[63] Thomas P Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[64] 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.

[65] Radford M. Neal. Suppressing random walks in markov chain monte carlo using ordered overrelaxation. *University of Toronto, Department of Statistics*, Technical report 9508, 1995.

[66] Christos Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441450, 1987.

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

[68] James D Park. Map complexity results and approximation methods. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 388–396. Morgan Kaufmann Publishers Inc., 2002.

[69] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1025 – 1032, August 2003.

[70] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in neural information processing systems*, pages 823–830, 2004.

[71] 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.

[72] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2011.

[73] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. *Methodologies for Intelligent Systems*, pages 350–361, 1993.

[74] Jean-François Puget. Dynamic lex constraints. In *CP*, pages 453–467. Springer, 2006.

[75] Jean-François Puget. An efficient way of breaking value symmetries. In *AAAI*, volume 6, pages 117–122, 2006.

[76] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 2009.

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

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

[79] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd Edition*. Prentice Hall, 2003.

[80] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. *SAT*, 4:7th, 2004.

[81] Tian Sang, Paul Beame, and Henry Kautz. Heuristics for fast exact model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 226–240. Springer, 2005.

[82] Bart Selman, Hector Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *In Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI92)*, pages 459–465, 1992.

[83] Guy Shani, Ronen I Brafman, and Solomon Eyal Shimony. Forward search value iteration for pomdps. In *IJCAI*, pages 2619–2624, 2007.

[84] Olivier Sigaud and Olivier Buffer. *Markov Decision Processes in Artificial Intelligence*. Wiley-ISTE, 2010.

[85] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[86] Marc Thurley. sharpsat–counting models with advanced component caching and implicit bcp. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 424–429. Springer, 2006.

[87] Marc Toussaint, Laurent Charlin, and Pascal Poupart. Hierarchical pomdp controller optimization by likelihood maximization. In *UAI*, volume 24, pages 562–570, 2008.

[88] 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.

[89] 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.

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

[91] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

[92] 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.

[93] 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.

[94] Martin J. Wainwright, Tommi S. Jaakkola, and Alan S. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions On Information Theory*, 51(7):2313–2335, 2005.

[95] Toby Walsh. Symmetry breaking constraints: Recent results. In *AAAI*, 2012.

[96] Peter van Beek Wei Li, Pascal Poupart. Exploiting structure in weighted model counting approaches to probabilistic inference. *Artificial Intelligence Research*, 40:729–765, 2011.

[97] Lintao Zhang. *Searching for truth: Techniques for satisfiability of boolean formulas*. PhD thesis, Princeton University, 2003.

[98] N. L. Zhang and W. Liu. Planning in stochastic domains: problem characteristics and approximation. *Department of Computer Science, Hong Kong University of Science and Technology*, Technical Report HKUST-CS96-31, 1996.

APPENDICES

Appendix A

Stationary Encoding of SAT

A SAT problem can be encoded as an undiscounted (no discount factor), unobservable (one uninformative observation), finite horizon POMDP. At each time step, the decision assigns a value to a variable (actions) and receives a reward proportional to the number of clauses satisfied (for the first time) by this assignment. An optimal policy satisfies the largest number of clauses possible. Here is a formal encoding of SAT as a POMDP.

A.1 SAT $\Rightarrow$ POMDP

Our basic goal is to borrow ideas from SAT solvers. To learn more about SAT problems and how they relate to POMDP problems we develop an encoding that transform SAT problems to POMDP problems and compare the states and actions. It is known that POMDPs are in the complexity class PSPACE-complete [66] and any problem in PSPACE or a lower complexity class such as NP can be reduced to a POMDP. The plan is to see if common SAT techniques that operate on variables and clauses carry over to POMDP problems.

We now show a transformation from SAT to POMDP where the policy corresponds to a satisfiable set of variable assignments in SAT and the horizon length will be the number of variables. We formally define SAT as a set of clauses $C = \{c_1, ..., c_{|C|}\}$ where each c_i is a disjunction of literals from the set of variables $X = \{x_1, ..., x_{|X|}\}$.

Therefore, the reduction from a SAT problem to POMDP problem given a formula F is:

- $S = \{c_0, c_1, c_2, ..., c_{|C|}\} \cup \{x_0, x_1, x_2, ..., x_{|X|}\}$ where $s_t \in S$ is the state at time t. A state labeled by c_i (with $i > 0$) indicates that clause c_i has not been satisfied yet.

The state labeled by c_0 can be interpreted as the entire formula is satisfied. Similarly, a state labeled by x_j (with $j > 0$) indicates that variable x_j has not been assigned a value yet. The state labeled by x_0 can be interpreted as all variables have been assigned a value. We define the initial belief as follows: $b_0(c_0) = 0$, $b_0(x_0) = 0$, $b_0(c_i) = \frac{1}{|C|+|X|}$ and $b_0(x_j) = \frac{1}{|C|+|X|}$ for all $i > 0$ and $j > 0$.

- $a_t \in A$ where $A = \{x_1 = true, x_1 = false, ..., x_{|X|} = true, x_{|X|} = false\}$ such that each action is an assignment of $x_i \in X$. Let $x_{i,0}$ denote $x_i = false$ and $x_{i,1}$ denote $x_i = true$ for all $i \in \{1, 2, ..., |X|\}$.

- $o_t \in O$ where $O = \{o\}$ is the singleton set of observations for all time steps such that no information is provided after each transition.

$$Pr(o_{t+1} = o | a_t, s_{t+1}) = 1, \text{ for all } a_t \text{ and } s_{t+1}$$

- The transition function is a deterministic function of the action, a_t. When in a state, $s_t = c_i$, the process will transition to the satisfiable state, c_0, if the current action satisfies clause c_i regardless if there are more variable assignments remaining. Otherwise, the process remains in the current state when the assignment does not satisfy the clause or the clause does not contain the current variable at time t. Similarly, for a state, $s_t = x_i$, the process will transition to the all assigned state, x_0, if the current action is an assignment of variable x_i otherwise it stays in the current state.

  Our belief state is updated by the elimination of invalid worlds based on actions selected and their probability mass transferred to the satisfiable state, i.e.,

$$Pr(s_{t+1}|a_t, s_t) = \begin{cases} 1 & \text{if } s_t = c_i \text{ and } a_t \text{ satisfies } c_i \text{ and } s_{t+1} = c_0 \\ 1 & \text{if } s_t = c_i \text{ and } a_t \neg\text{satisfy } c_i \text{ and } s_{t+1} = c_i \\ 1 & \text{if } s_t = x_i \text{ and } a_t = x_{i,\cdot} \text{ and } s_{t+1} = x_0 \\ 1 & \text{if } s_t = x_i \text{ and } a_t \neq x_{i,\cdot} \text{ and } s_{t+1} = x_i \\ 0 & \text{otherwise} \end{cases}$$

- Reward function:

$$R(s_t, a_t) = \begin{cases} \frac{|C|+|X|}{|C|} & \text{if } s_t = c_i \text{ and } a_t \text{ satisfies } c_i \\ 1 & \text{if } s_t = x_i \text{ and } a_t = x_{i,\cdot} \\ -\frac{1}{|X|-1} & \text{if } s_t = x_i \text{ and } a_t \neq x_{i,\cdot} \\ 0 & \text{otherwise} \end{cases}$$

  Therefore, the possible actions available at time t is equivalent to the number of

remaining unassigned variables since negative rewards will be accumulated otherwise.

- the parameter γ should be set to 1.
- A satisfiable set of assignments exists if after a horizon of length $|X|$ the belief at $b(c_0) = \frac{|C|}{|C|+|X|}$ and $b(x_0) = \frac{|X|}{|C|+|X|}$ with a reward of 1 accumulated. The solution to the SAT problem is encoded in the policy, $\pi(b_t)$, such that each x_t assignment corresponds to an action chosen by $\pi(b_t)$.

A.2 Theorem

Here, we present an outline of a proof that shows the equivalence between SAT and our POMDP reduction. We will show that (1) any satisfiable SAT solution corresponds to an optimal policy in the equivalent POMDP and (2) an assignment from any optimal policy in our reduces POMDP corresponds to a satisfiable solution in SAT.

1. Assume there exists a satisfiable joint assignment $\mathbb{S} = \langle x_1 := a_1, x_2 := a_2, ..., x_{|X|} := a_{|X|} \rangle$. We argue this will correspond to an optimal policy in the POMDP. Consider a policy, $\pi^{\mathbb{S}}$, that makes the same assignments as $\mathbb{S}$. After $|X|$ steps all clauses will be satisfied and we will receive total reward of 1, hence, $\pi^{\mathbb{S}}$ is optimal.
2. Assume $\pi^{\mathbb{S}}$ is an optimal policy in our reduction to POMDP. We argue that this policy corresponds to a satisfiable set of assignments in the SAT problem. Since the belief state has all its probability mass in state c_0 and x_0 after $|X|$ steps, this implies that for each clause c_i there exists satisfiable action a_t. Therefore, the joint assignments $\mathbb{S} = \langle x_1 := a_1, x_2 := a_2, ..., x_{|x|} := a_{|X|} \rangle$ is sufficient to satisfy all the clauses in the SAT problem and hence is a solution.
3. We can use the same reason to show that for the unsatisfiable cases the reduction holds. Assume there is no satisfiable joint assignments $\mathbb{S} = \langle x_1 := a_1, x_2 := a_2, ..., x_{|X|} := a_{|X|} \rangle$. We argue this will correspond to a policy in the POMDP with value less than 1. Consider a policy, $\pi^{\mathbb{S}}$, that makes the same assignments as $\mathbb{S}$. After $|X|$ steps, not all clauses will be satisfied and therefore we will receive a reward less than 1 in the POMDP, hence, the value of $\pi^{\mathbb{S}}$ is less than 1.

States	b_0	b_1	b_2	b_3	b_4	b_5
c_0	0	1/8	2/8	2/8	2/8	3/8
$c_1 = x_3 \vee x_4 \vee \neg x_5$	1/8	1/8	0	0	0	0
$c_2 = \neg x_1 \vee \neg x_2 \vee x_4$	1/8	0	0	0	0	0
$c_3 = x_1 \vee \neg x_2 \vee x_5$	1/8	1/8	1/8	1/8	1/8	0
x_0	0	1/8	2/8	3/8	4/8	5/8
x_1	1/8	0	0	0	0	0
x_2	1/8	1/8	1/8	1/8	0	0
x_3	1/8	1/8	0	0	0	0
x_4	1/8	1/8	1/8	0	0	0
x_5	1/8	1/8	1/8	1/8	1/8	0

Table A.1: States updated after taking actions $x_1 = false$, $x_3 = true$, $x_4 = false$, $x_2 = true$, and $x_5 = true$.

A.3 Example

Lets focus on the state representation by using Eq. 2.8 as an example. Shown in column two of Table A.1 is the initial belief state the process could be in where each clause has equal probability of being unsatisfiable. In columns b_1 to b_5 is the belief state if we consider a policy, π^F that recommends the assignments $\neg x_1 \wedge x_3 \wedge \neg x_4 \wedge x_2 \wedge x_5$. Since the belief state has all its mass concentrated in the satisfiable state, c_0, after all variables have been assigned in x_0, we can conclude that the original SAT problem is satisfiable and π^F is an optimal policy. An immediate reward of 1 is also gained from being in the satisfiable state.

In summary, we have shown how to reformulate a SAT problem as a POMDP where the size of the state space, $|C|+|X|+2$, is linear in the number of clauses $|C|$ and variables $|X|$.