

# Propositional Probabilistic Planning-as-Satisfiability using Stochastic Local Search

Nathan Robinson, Charles Gretton, Duc-Nghia Pham, and Abdul Sattar

SAFE Program, Queensland Research Lab, NICTA and  
Institute for Integrated and Intelligent Systems, Griffith University, QLD, Australia  
{nathan.robinson,charles.gretton,duc-nghia.pham,abdul.sattar}@nicta.com.au

## Abstract

Recent times have seen the development of a number of planners that exploit advances in SAT(sifiability) solving technology to achieve good performance. In that spirit we develop the approximate contingent planner PLSPLAN. Our approach is based on a stochastic local search procedure for solving stochastic SAT (SSAT) representations of probabilistic planning problems. PLSPLAN first constructs an SSAT representation of the  $n$ -timestep *probabilistic plangraph* for the problem at hand. It then iteratively calls a stochastic local search procedure to find a linear plan (sequence of actions) which achieves the goal (i.e. satisfies the SSAT formula) with non-zero probability. Linear plans thus generated are merged to create a single contingent plan. Successive iterations progress from deciding the outcomes of stochastic actions in order to find a linear plan quickly, to sampling the outcomes of actions. Consequently, PLSPLAN efficiently finds a linear plan which logically achieves the goal. Over time it refines its contingent plan for likely scenarios. We empirically evaluate PLSPLAN on benchmarks from the probabilistic track of the 5th International Planning Competition.

## Introduction

Stochastic Boolean satisfiability (SSAT) is an extension of Boolean satisfiability (SAT) that was developed to support reasoning about uncertainty. Just as SAT is a useful formalism for solving deterministic planning problems, SSAT is an useful formalism for solving probabilistic planning problems. Indeed, a number of state-of-the-art contingent planners have been developed in the planning-as-satisfiability paradigm. These operate on an SSAT representation of the problem, in which the logical possibilities for acting in a problem are encoded as a propositional conjunctive normal form (CNF) formula. The quantified uncertainty in the problem is captured by random variables in the formula that occur true with a specified rational probability. Existing SSAT-based contingent planners can be classified into two categories: (1) those that use Davis-Putnam-Logemann-Loveland (DPLL) procedure (complete search), and (2) those that use a stochastic local search (SLS).

In the first category, we have planners that borrow a DPLL procedure for SAT directly, e.g., APPSSAT (Majercik 2006). Otherwise, we have planners that use a modified DPLL in which the target propositional formula in-

cludes variables whose truth values are determined probabilistically. Examples of the latter include the SSAT solver EVALSSAT (Littman, Majercik, & Pitassi 2001) – and the approximate version of this SAMPLEVALSSAT – and the planners MAXPLAN (Majercik & Littman 1998a), ZANDER (Majercik & Littman 2003), and DC-SSAT (Majercik & Boots 2005). These modified DPLL procedures are specialised to planning in the way they choose an *active variable* during search. In particular, this is chosen using the discrete timestep the variable describes, starting in the first state and then moving out to the planning horizon. Intuitively such a variable ordering heuristic works because planning problems are Markovian – i.e., when an action is executed, the current state alone determines possible successor states and the probability that they occur. Whether the DPLL procedure is borrowed from the SAT community or designed specifically for the SSAT case, standard pruning techniques such as *unit propagation* and *purification* are employed. Also, in order to avoid repeating expensive computation in planning for contingencies, DPLL approaches use techniques for caching solutions to subproblems (Majercik & Littman 1998b). APPSSAT is the most unique solver in this category of planners. It uses the DPLL-based SAT solver ZCHAFF to find sequences of actions that achieve the goal with non-zero probability. These are combined to form a single contingent plan.

To the best of our knowledge, the only contingent planner in the category based on an SLS procedure is RANDEVALSSAT (Littman, Majercik, & Pitassi 2001). This uses a preprocessing phase that restricts the search to a small set of legal instantiations of random variables. Each instantiation is generated by sampling the truth values of the random variables. An SLS procedure based on WALKSAT (Selman, Kautz, & Cohen 1994) then searches for a contingency plan, greedily accommodating as many contingencies as possible. The RANDEVALSSAT solver was not designed specifically for planning, and unfortunately it is not an appealing technique in the planning setting because: (1) vanilla SLS algorithms such as WALKSAT are notoriously bad at solving planning problems, (2) the SLS heuristic guides RANDEVALSSAT to plans with many contingencies, rather than to the optimal plan which may only require a few contingencies; and (3) where the only plans that achieve the goal do so with a low probability, the solver misses these without an

impractical amount of sampling.

Addressing advances in SAT technology more generally, in recent times we have seen very promising developments regarding Boolean satisfiability (SAT) approaches to solving deterministic combinatorial problems. From a planning perspective, the most visible example of this is in the SAT based planners SATPLAN-04/06 (Kautz, Selman, & Hoffmann 2006) which took first place in the optimal STRIPS (deterministic) track of the 2004 International Planning Competition, and then tied for first place with SAT-based planner MAXPLAN (Chen, Xing, & Zhang 2007) in 2006.

Looking at SAT technology from the perspective of local search, since the introduction of GSAT (Selman, Levesque, & Mitchell 1992), SLS algorithms for SAT have been intensively improved, now being able to solve many highly complicated real-world problems including planning and scheduling problems. SLS based SAT solving techniques have proved to be very effective for large size problems, and are the undisputed champions for solving hard random SAT problems.<sup>1</sup> Also, SLS was recently shown to solve large highly structured problems when certain classes of variable dependencies are made available to be exploited by the search (Pham, Thornton, & Sattar 2007; 2008).

Taking recent advances in SAT technology for planning, and SLS for structured SAT problems, this paper develops PLSPLAN, an SLS-based planning-as-satisfiability approach for approximate contingent planning in fully observable propositional probabilistic planning problems. PLSPLAN operates as follows:

- Where the given planning horizon is  $n$ , compute and prune the  $n$ -timestep probabilistic plangraph.
- Compute an SSAT representation of the problem posed by the plangraph.
- Compute a list of planning constraints, such as “only one action can be executed in each timestep”, to be exploited by the PLSPLAN SLS procedure.
- Iteratively compute *linear plans*<sup>2</sup> for progressively more likely SAT encoded determinisations of the SSAT problem using a version of weighted-SLS that respects the planning constraints. Successively generated plans are incorporated into a single contingent plan.

The rest of the paper is organised as follows. We describe the problem of propositional probabilistic planning for the fully observable case. We then describe the probabilistic plangraph data structure, and how the planning problem posed by it can be captured using an SSAT representation. We then describe PLSPLAN, our SLS algorithm for generating a contingent plan. Finally we present experimental results before concluding with some remarks about related and future work.

## Probabilistic Planning

A propositional probabilistic planning problem is given in terms of a finite set of stochastic actions  $\mathcal{A}$ , deterministic

<sup>1</sup><http://satcompetition.org>

<sup>2</sup>Sequences of actions that achieve the goal with non-zero probability.

outcomes  $\mathcal{O}$ , and propositions  $\mathcal{P}$ . A problem state  $s$  is a set of propositions  $s \subseteq \mathcal{P}$ . For  $p \in s$  we say proposition  $p$  characterises state  $s$ . There is always a unique starting state  $s_0$ . The goal  $\mathcal{G}$  is a set of propositions, and we say that state  $s$  is a goal state iff  $\mathcal{G} \subseteq s$ .

Every stochastic action has a precondition  $\text{poss}(a)$ , which is a set of propositions. An action can be executed at a state  $s$  when  $\text{poss}(a) \subseteq s$ . We denote  $\mathcal{A}(s)$  the set of actions that can be executed at state  $s$ . When  $a \in \mathcal{A}(s)$  is executed, nature decides amongst a small set of deterministic outcomes  $\mathcal{O}(a) \equiv \{o_1, \dots, o_k\}$  what actually occurs. To keep this exposition simple, for any two distinct actions  $a_i \neq a_j$ , if outcome  $o$  is a possibility for  $a_i$  then it cannot also be a possibility for  $a_j$  – i.e., if  $o \in \mathcal{O}(a_i)$  then  $o \notin \mathcal{O}(a_j)$ . We denote  $\mu_a(o_i)$  the (rational) probability that nature takes outcome  $o_i$ , and for all  $a$  we require  $\sum_{o_i \in \mathcal{O}(a)} \mu_a(o_i) = 1$ . The outcome that nature chooses is observable, and its effect is given in terms of two lists of propositions called the add list  $\text{add}(o)$  and delete list  $\text{delete}(o)$ . If a proposition is in the add list of  $o$ , then it cannot be in the delete list and vice versa. If outcome  $o$  with  $\text{add}(o) := [p_1, \dots, p_n]$  and  $\text{delete}(o) := [p_1, \dots, p_m]$  is executed at state  $s$ , then the resultant state is  $(s \cup \text{add}(o)) \setminus \text{delete}(o)$  – i.e., propositions from  $\text{add}(o)$  are added to  $s$ , and those from  $\text{delete}(o)$  are removed from  $s$ .

The solution to a probabilistic planning problem is a contingent plan. This consists of an assignment of actions to states at each discrete timestep up to the planning horizon  $n$ . The optimal contingent plan is one which prescribes actions to states that maximise the probability that the goal is achieved within  $n$  steps from the starting state  $s_0$ . For the purposes of this paper we say a plan fails, i.e. achieves the goal with probability 0, in situations where it does not prescribe an action. Computing the optimal plan for a problem is computationally intractable, and an important direction for research in the field is to develop heuristic mechanisms for generating small linear plans quickly (Littman, Goldsmith, & Mundhenk 1998).

## Probabilistic Plangraph

The *plangraph* is a data structure that was devised for the efficient GRAPHPLAN framework of deterministic planning in STRIPS domains with restrictions on where negated propositions occur in the problem specification (Blum & Furst 1997). Since its inception, the plangraph has been adapted to various richer classes of planning problems including that of probabilistic planning. Here, we are concerned with the *probabilistic plangraph* (Blum & Langford 1999), an adaptation of the plangraph for problems that feature quantified uncertainty about the effects of actions. Like its deterministic counterpart, the probabilistic plangraph captures necessary conditions for goal achievement, and is computationally cheap to build, taking polynomial time in the number of propositions and action outcomes. In addition, the size of the generated graph is also polynomial in those factors. In the rest of this section, we first describe the plangraph for the deterministic setting and then demonstrate how to adapt it for the probabilistic setting. Finally, we described a tech-

nique we adopt from (Blum & Langford 1999) for pruning the probabilistic plangraph.

For a deterministic problem with actions  $\mathcal{A}$  and propositions  $\mathcal{P}$ , the *plangraph* is a directed *layered* graph.<sup>3</sup> Vertices are labelled with a timestamp, and either a proposition or an action. Each layer is a set of vertices, each of which is labelled with the same timestamp. Additionally, vertices in a layer have their labels drawn from a set of actions or a set of propositions. For each timestep up to the planning horizon, the graph comprises two layers, one action layer and the other propositional. There is also a final propositional layer for the goal. The propositional layer with timestamp  $t$  contains a set of propositions. Elements in the powerset of this are states which might be reachable in  $t$  timesteps from  $s_0$ . Similarly, an action layer with timestamp  $t$  contains the set of actions which might be executable  $t$  steps into a plan.

In more detail, the first layer has a vertex for each proposition in the starting state  $s_0$ . For every proposition and action layer with the same timestamp  $t$ , the action layer has a vertex for each action in  $\mathcal{A}(s_0)$ , as well as an artificial action  $a_p$  for each  $p$  in the propositional layer. We define  $a_p$  to have the precondition that  $p$  is true, and the one (add) effect of keeping it true. There is an arc from vertex labelled  $p$  in the propositional layer to that labelled  $a$  in the action layer if  $p$  is a precondition for  $a$ . For every action layer with timestamp  $t$  and propositional layer with timestamp  $t + 1$ , the propositional layer has a vertex for each proposition that is a positive effect of an action from the action layer. There is an arc labelled *positive* from each action to propositions that are *add* effects of that action. There is an arc labelled *negative* from each action to propositions that are *delete* effects of that action.

A *probabilistic plangraph* is a simple adaptation of the *plangraph* to planning problems that feature quantified uncertainty about the effects of actions. In this case the action layer comprises vertices from the stochastic actions. There are outgoing positive and negative arcs from action vertices for the add and delete effects of each deterministic outcome. These arcs are labelled with the associated outcome, and the probability of that outcome occurring. We adopt a simple pruning mechanism from (Blum & Langford 1999) to reduce the size of the plangraph, and hence the solution space that PLSPLAN searches. In particular, we remove a vertex from the plangraph if there is no path to a goal proposition from that vertex. This is a simple pruning step that is performed by iterating backward through the plangraph from the goal layer.

## Stochastic SAT

Here we review SSAT, the formalism on which our encoding of the planning problem posed by an  $n$ -timestep probabilistic plangraph is based. An SSAT problem  $\langle \text{Qs}, \phi \rangle$  is defined in terms of a list of quantified boolean variables  $\text{Qs} := Q_1 v_1 Q_2 v_2, \dots, Q_n v_n$  and a propositional CNF formula  $\phi$  such that all variables in  $\phi$  are quantified in Qs. A quantifier  $Q_i$  is either an existential  $\exists$  or random  $R^r$  where super-

<sup>3</sup>A problem is deterministic when each action has only one outcome which occurs with 100% certainty.

script  $r$  is a rational number  $0 < r < 1$ . If a variable is existentially quantified we call it a *choice* variable, otherwise it is randomly quantified and called a *chance* (or random) variable. A problem is a SAT problem if all the quantifiers in Qs are existential.

We write  $\phi[v = \top]$  for  $\phi$  with  $v$  assigned to true and  $\phi[v = \perp]$  for  $\phi$  with  $v$  assigned to false. Writing  $\bullet$  for the empty list and  $Qv : \text{Qs}$  for a list of quantified variables with head  $Qv$  and tail Qs, the value of an SSAT problem  $\langle \text{Qs}, \phi \rangle$  given an assignment  $\alpha$  to the choice variables is given by Algorithm 1. For an assignment  $\alpha$ , we write  $\alpha(p)$  for the value, true  $\top$  or false  $\perp$ , assigned to  $p$  according to  $\alpha$ .

---

### Algorithm 1 Evaluating SSAT

---

$eval(\bullet, \phi, \alpha)$	= 0 if $\phi$ contains the empty clause
$eval(\bullet, \phi, \alpha)$	= 1 if there are no clauses in $\phi$
$eval(\exists v : \text{Qs}, \phi, \alpha)$	= $eval(\phi[v = \alpha(v)])$
$eval(R^r v : \text{Qs}, \phi, \alpha)$	= $r \cdot eval(\phi[v = \top]) + (1 - r) \cdot eval(\phi[v = \perp])$

---

For an assignment  $\alpha$  to the choice variables, *eval* computes the probability that the CNF  $\phi$  will be satisfied. For our purposes the solution to  $\langle \text{Qs}, \phi \rangle$  is an assignment  $\alpha^*$  to the choice variables in Qs which maximises the evaluation.<sup>4</sup>

$$\alpha^* = \underset{\alpha}{argmax} eval(\text{Qs}, \phi, \alpha)$$

Note that the above definition of SSAT is more general than we require. In planning it makes more sense that the order of evaluation be determined by the consequence relation between an action and its outcomes, rather than by the order in which variables occur in a quantifier list. This is because: (1) uncertainty is limited to the choice nature makes when an action is executed, and (2) only one action can be executed in a given timestep and only one action outcome can occur. We will take advantage of (2) below as we use SSAT formula to encode a linear, resp. contingent, planning problem.

## SSAT Encoding of the Probabilistic Plangraph

We devise *evalp* (Algorithm 2), an evaluation algorithm specific to an SSAT encoding of an  $n$ -timestep probabilistic plangraph. We adopt the notations  $a^t \in \mathcal{A}^t$ ,  $p^t \in \mathcal{P}^t$ , and  $o^t \in \mathcal{O}^t$  for action, proposition and choice symbols at timestep  $t$ . For the composition of Qs, each  $p^t$  and  $a^t$  symbol is a choice variable, and each  $o^t \in \mathcal{O}^t$  is a chance variable quantified according to  $R^r o^t$  where  $r := \mu_a(o)$  – i.e., the probability of  $o^t$  occurring if its corresponding stochastic action is executed. For *evalp* the order in which variables occur in Qs is not important. Leaving the logical structure of the planning problem  $\phi$  aside for the moment, since it will

<sup>4</sup>Note that the order in which variables appear in the quantifier list is important. This is because a formula is essentially copied in the evaluation rule for the random quantifier. Consequently, an existentially quantified variable  $v$  that occurs after a randomly quantified variable  $v'$  in Qs can take on a different value for the case that  $v' = \top$  and the case  $v' = \perp$ .

be captured in the CNF  $\phi$  to be described in a moment,  $evalp$  is defined as follows:

---

**Algorithm 2** Evaluating SSAT

---

$$\begin{aligned}
evalp(\bullet, \phi, \alpha) &= 0 \text{ if } \phi \text{ contains the empty clause} \\
evalp(\bullet, \phi, \alpha) &= 1 \text{ if there are no clauses in } \phi \\
evalp(\exists p^t : Qs, \phi, \alpha) &= evalp(Qs, \phi[p^t = \alpha(p^t)], \alpha) \\
evalp(R^r o^t : Qs, \phi, \alpha) &= evalp(Qs, \phi, \alpha) \\
evalp(\exists a^t : Qs, \phi, \alpha) &= \\
&\Sigma_{o_i^t \in \mathcal{O}^t(a^t)} r_i \cdot evalp(Qs, \\
&\quad \phi[a^t = \alpha(a^t), \\
&\quad \quad o_i^t = \top, \\
&\quad \quad \forall o_j^t \in \mathcal{O}^t(a^t). \text{ if } (j \neq i) \text{ then } o_j^t = \perp], \\
&\quad \alpha)
\end{aligned}$$

where symbol  $o_i^t$  is quantified  $R^{r_i}$

---

The planning problem is thus to find  $\alpha^*$  so that

$$\alpha^* = \underset{\alpha}{argmax} evalp(Qs, \phi, \alpha)$$

As a final step in modelling an  $n$ -timestep probabilistic planning problem in SSAT, we require the CNF  $\phi$ , the conjunct of the axioms we enumerate below. Here, we adopt the notation  $\exists! p \in P.p$  to mean that exactly one of the propositions in the set  $P$  must be true.

1. *Starting state and goal axioms:* To encode the starting state we have the formula  $\forall p^0 \in s_0.p^0$ , and similarly for the goal we have  $\forall p \in \mathcal{G}.p^n$ .

2. *Nature's choice axioms:* For each action  $a^t \in \mathcal{A}^t$ , we have the formula  $a^t \rightarrow \exists! o^t \in \mathcal{O}(a^t).o^t$ , which says that when action  $a^t$  is executed, exactly one of the random outcomes associated with it must occur.

3. *Action precondition axioms:* For each action  $a^t \in \mathcal{A}^t$ , we have the formula  $a^t \rightarrow \bigwedge_{p \in \text{poss}(a^t)} p^t$ , which says action  $a^t$  cannot be executed unless its precondition is satisfied. We also assert *action exclusivity*, thus the action physics are such that only one action can be executed at any timestep. To achieve this, for each  $\mathcal{A}^t \in \mathcal{A}$ , we have the formula  $\exists! a^t \in \mathcal{A}^t.a^t$ .

4. *Successor states axioms:* We require formulae that encapsulate how the truth value of propositions  $p^t \in \mathcal{P}^t$  change from state to state. In the case that a proposition is effected by an action, for each stochastic action  $a^t \in \mathcal{A}^t$  and choices associated with this  $o^t \in \mathcal{O}(a^t)$ , we have the formula

$$(a^t \wedge o^t \rightarrow (\bigwedge_{p \in \text{add}(o)} p^{t+1} \wedge \bigwedge_{p \in \text{delete}(o)} \neg p^{t+1})) \quad (1)$$

We also require *frame axioms* which state that the truth value of a proposition is only changed by nature's choices which have that proposition in their add or delete lists. To this purpose we define  $\text{make}(p) \subseteq \mathcal{A} \times \mathcal{O}$  to be the set of pairs of actions and outcomes so that, for each element  $\langle a, o \rangle \in \text{make}(p)$  we have that  $o$  is a choice available to nature when we execute  $a$  and also that  $p \in \text{add}(o)$ .  $\text{break}(p) \subseteq \mathcal{A} \times \mathcal{O}$  is defined similarly only in this case  $p \in \text{delete}(o)$ . For each  $p^t \in \mathcal{P}^t$  for  $t = 1..n$  we have the formula

$$(p^t \not\leftrightarrow p^{t-1}) \leftrightarrow (\neg p^{t-1} \wedge \bigvee_{\langle a, o \rangle \in \text{make}(p)} (a^{t-1} \wedge o^{t-1})) \vee (p^{t-1} \wedge \bigvee_{\langle a, o \rangle \in \text{break}(p)} (a^{t-1} \wedge o^{t-1})) \quad (2)$$

## Generating Linear Plans with SLS

A key component of our approach is an SLS-based SAT solver we developed called SLSPLAN. In constructing a contingent plan, PLSPLAN repeatedly invokes this procedure on different determinisations of the problem at hand. Each invocation yields a linear solution plan (where one exists) that achieves the goal with non-zero probability. Along the same lines as GSAT (Selman, Levesque, & Mitchell 1992), SLSPLAN is a typical SLS solver in that it starts with an assignment  $\alpha$  of truth values to the variables in the given formula  $\phi$ , and iteratively improves that assignment using the local knowledge of the search space. In particular, at every iteration the current assignment  $\alpha$  is perturbed to a neighboring assignment  $\alpha_v$  by flipping the value of a variable  $v$ . Many heuristics have been developed to determine what variable to flip (Hoos & Stützle 2004). We have opted for a *dynamic local search* (i.e., *clause weighting*) scheme that has been used in a wide range of modern SAT solvers. It is a good strategy to avoid having the search stagnate in local optima (Hoos & Stützle 2004).

Algorithm 3 gives the pseudo code for SLSPLAN. The local search takes as arguments a CNF representation  $\phi$  of a determinisation of the planning problem, and an upper bound on the number of iterations that can be performed  $\text{maxSteps}$ . It also takes a contingent plan  $\pi$ , which is interpreted as a collection of solutions that SLSPLAN should avoid.<sup>5</sup> The local search then constructs its initial assignment  $\alpha$  of truth values to all variables in  $\phi$  so that: (1) up to

<sup>5</sup>In our setting this is the set of linear solution plans that have

---

**Algorithm 3** SLSPLAN( $\phi, \text{maxSteps}, \pi$ )

---

- 1: generate an initial assignment  $\alpha$ ;
  - 2: initialise the weight  $\lambda_i$  of each clause  $c_i$  to 1;
  - 3: **for**  $\text{step} = 1$  to  $\text{maxSteps}$  **do**
  - 4:   **if**  $\alpha$  satisfies  $\phi$  **then**
  - 5:     **return**  $\alpha$  as the solution;
  - 6:   **else if** within walk probability  $wp$  **then**
  - 7:     Randomly select a non-unary *unsatisfied* clause  $c$ ;
  - 8:     Satisfy  $c$  by randomly flipping the value of a variable in  $\alpha$ , but avoid solutions from  $\pi$ ;
  - 9:   **else**
  - 10:    **if** there exists a variable  $v$  s.t.  
 $\Sigma_i \lambda_i c_i(\alpha_v) < \Sigma_i \lambda_i c_i(\alpha)$  and  $\alpha_v \notin \pi$   
**then**
  - 11:     Flip the value of  $v$  and update  $\alpha$ ;
  - 12:    **else**
  - 13:     add 1 to the weight of each *unsatisfied* clause;
  - 14:    **end if**
  - 15:    **end if**
  - 16: **end for**
  - 17: **return** 'no solution found';
-

the planning horizon  $n$ , at each timestep  $t \in [1 \dots n - 1]$ , propositions true in  $s^t$  are chosen uniformly at random from  $\mathcal{P}$ , (2)  $s^0 := s_0$ , (3) state  $s^n$  is also random except  $\mathcal{G} \subseteq s^n$ , and (4) an action is chosen uniformly at random from  $\mathcal{A}$  and executed at each timestep.

The heuristic Algorithm 3 used to guide search is typical of a dynamic local search. It comprises a weighting scheme that associates each clause  $c_i$  in  $\phi$  with a numerical weight  $\lambda_i$ . Initially all weights are 1. With a little notational abuse, we can treat the  $i$ th clause  $c_i$  as a 0/1 function so that  $c_i(\alpha) = 0$  if  $c_i$  is satisfied by  $\alpha$  and  $c_i(\alpha) = 1$  otherwise. SLSPLAN uses the sum of weights of all unsatisfied clauses under the current assignment  $\alpha$  as its objective function to greedily move to a “better” solution. More formally, we have that the value of assignment  $\alpha$  is  $\sum_i \lambda_i c_i(\alpha)$ . Line 13 of Algorithm 3 provides for the clause weights to be adjusted dynamically during search in order to modify the landscape to avoid and otherwise escape local minima.

At each iteration, SLSPLAN selects and flips a variable  $v$  that strictly minimises the cost function  $\sum_i \lambda_i c_i(\alpha)$ . As a detail, we break ties by selecting the least recently flipped candidate variable. The solver also ensures that flipping  $v$  will not lead to an assignment in the given contingent plan  $\pi$ . If no strict improving variable exists, the weights of all unsatisfied clauses are increased by 1. Although periodically reducing clause weights improves the performance of many weighted SLS solvers (Wu & Wah 2000; Hutter, Tompkins, & Hoos 2002; Thornton *et al.* 2004), we find that it is not the case for SLSPLAN. Rather, we include the random walk heuristic into SLSPLAN to weaken its determinism in searching for candidate variables. Within a probability  $wp$ , the solver attempts to satisfy an unsatisfied clause by flipping the truth value of a randomly selected variable in that clause; otherwise it selects and flips a variable using the weighting heuristic described above. We find that the performance of SLSPLAN with probabilistic random walks is significantly improved. Moreover, this strategy is very robust in contrast to the high sensitivity of the parameter(s) used in existing weighted SLS solvers to control when clause weights are reduced.

Finally, the reason we have developed SLSPLAN, as opposed to adopting an off-the-shelf local search procedure, is because we are targeting planning problems specifically (rather than SAT instances more generally). In particular, in SLSPLAN we take advantage of the opportunity to exploit problem structures that are otherwise obfuscated in a CNF representation of the problem. In the following we describe how SLSPLAN explicitly handles some of the planning constraints when selecting a variable to flip.

### Mutex Control and Constraint Propagation

Recent research has shown that letting SLS-based SAT solvers explicitly handle constraints such as  $\exists! p \in P.p$  or  $p \rightarrow \bigwedge_i q_i$  improves their performance significantly over implicitly encoding these into the CNF (Frisch & Peugniez 2001; Ansótegui *et al.* 2003; Pham *et al.* 2005;

previously been reported to PLSPLAN by invocations of SLSPLAN.

Pham, Thornton, & Sattar 2007). In SLSPLAN we annotate the problem  $\phi$  with compact representations of the planning constraints that assert the exclusivity of actions and their outcomes. During the search, SLSPLAN maintains these mutex constraints in a similar manner to (Pham *et al.* 2005): setting a current false variable of a mutex constraint to true and at the same setting the current true variable of that constraint to false. In addition, as each outcome is unique and is associated with a unique action, if an outcome is flipped to true then SLSPLAN will immediately enforce that its associated action is also true as part of the nature’s choices axioms. As both the outcome and its associated action are now true, SLSPLAN then performs a constraint propagation to ensure that the action effect component of the successor states axiom (described in Eq 1) is satisfied – i.e. enforcing propositions in the add and delete lists associated with the action to be true if they were not.<sup>6</sup>

## PSLSPLAN

We now describe our contingent planner PLSPLAN, that repeatedly invokes our SAT procedure SLSPLAN on determinisations of the probabilistic planning problem at hand in order to generate a collection of linear plans, each of which is merged into a single contingent plan  $\pi$ . The pseudo code is given in Algorithm 4. Initially, PLSPLAN generates an SSAT representation  $\langle Qs, \phi \rangle$  of the original plangraph by (1) generating and pruning the  $n$ -timestep probabilistic plangraph from the original problem specification, and (2) translating this graph into SSAT. The algorithm then iteratively generates a determinisation  $\phi'$  of the SSAT problem by sampling the truth values of a number  $m$  of the random variables while respecting the planning constraints. At this point all the remaining variables in  $\phi'$  are treated as choice variables, which means that our SLSPLAN procedure can determine the values of some random variables whose truth values were not sampled, by treating them as if they were existentially quantified.

Apart from our use of an SSAT-based SLS procedure SLSPLAN to solve probabilistic problems, a key feature of PLSPLAN is the way random variables, and hence the probabilistic effects of actions, are treated by the algorithm. Over successive invocations of SLSPLAN, the truth values of random variables can be increasingly determined by sampling. Thus, for the first few iterations we can have that SLSPLAN decides the truth values of random variables as if they were existentially quantified. In later iterations, we can have that these are determined wholly by sampling. Our strategy of allowing the SLS procedure to determine the assignment to some chance variables has two important consequences: (1) if the only plans that achieve the goal do so with a low probability, we are able to efficiently find them at the outset, and (2) over time the plan search can be focused on planning for likely contingencies.

<sup>6</sup>We find that constraint propagation for actions precondition axioms, which have the same form as effect axioms, are not effective.

## Experimental Evaluation

We implemented the PLSPLAN planner in C++ and evaluate our implementation here using a number of problems from the probabilistic track of the 5<sup>th</sup> International Planning Competition (IPC-5, 2006). In the following sub-sections, we briefly describe the problem domains used in our evaluation of PLSPLAN, and then present and discuss the results.

### Benchmark Problem Domains

We selected problems from the following 4 domains out of 9 used in the probabilistic track from IPC-5: blocks-world, exploding-blocks-world, elevator, and tireworld. We omitted the other 5 domains: random, zenoworld, drive, pitchcatch, and schedule, because translating these problems into the standard probabilistic planning formalism was too clumsy. Below we use the same naming/numbering scheme from the competition to label benchmark instances, appending  $-tn$  where  $n$  is the planning horizon.<sup>7</sup> Below, we give brief summaries of the domains we have considered.

The blocks-world is a stochastic version of the classic blocks-world planning benchmark with probabilistic actions. The key property that makes this domain hard is a group of probabilistic actions that allow for a stack of blocks to be held. Being able to hold a stack of blocks makes achieving the goal easy in few timesteps. However, when holding a stack, placing a block  $A$  on another block  $B$  fails 90% of the time, resulting in  $A$  usually being stacked on the table. Thus, although holding a stack of blocks yields a short linear plan to the goal, the probability of success in this case is small.

The exploding-blocks-world domain is a stochastic version

<sup>7</sup>Details of all the instances, problem files and domain files from the IPC-5 competition are available at [www ldc.usb .ve/~bonet/ipc5/](http://www ldc.usb .ve/~bonet/ipc5/).

---

#### Algorithm 4 PLSPLAN(*Problem*)

---

- 1: generate and prune the  $n$ -timestep probabilistic plan-graph for the given *Problem*;
  - 2: generate the SSAT representation  $\langle Qs, \phi \rangle$  of the problem posed by the probabilistic plan-graph;
  - 3: initialise the contingent plan  $\pi$  to be empty;
  - 4:  $m := 0$ ;
  - 5: **repeat**
  - 6:   sample an assignment  $\alpha'$  to  $m$  chance variables (chosen uniformly at random) from  $Qs$  such that the choice axioms  $a^t \rightarrow \exists ! o^t \in \mathcal{O}(a^t). o^t$  in  $\phi$  can never be violated;
  - 7:   compute  $\phi'$  by simplifying  $\phi$  according to  $\alpha'$ ;
  - 8:    $\alpha := \text{SLSPLAN}(\phi', 10^6, \pi)$ ;
  - 9:   **if**  $\text{eval}_p(Qs, \phi, \alpha) > 0$  **then**
  - 10:     add assignment  $\alpha$  to plan  $\pi$ ;
  - 11:   **end if**
  - 12:    $m := m + 1$ ;
  - 13: **until** ‘user terminates plan search’
  - 14: **return**  $\pi$ ;
- 

of the classic blocks-world in which blocks can (probabilistically) detonate when they are placed on the table or another block. The result of a detonation is that the object under a detonated block explodes. No block can be stacked on an exploded object. Blocks are more likely to detonate when they are placed on the table, thus plans which avoid placing blocks on the table are more likely to succeed.

The elevator domain comprises an  $n$  story building where each floor consists of an  $m$  length list of positions some of which are adjacent to an elevator shaft. If position  $i$  at floor  $j$  is adjacent to a shaft, then, where they exist, positions  $i$  at floors  $j \pm 1$  are also adjacent to that shaft. There is a passenger who starts in the first position of the first floor, and can either (1) move to an adjacent position, (2) use the elevator if she is in a position adjacent to a shaft, or (3) collect a coin. In the case that she uses an elevator, she can go to a specific position in the floor above (or below). All actions associated with using the elevator are deterministic, however specific positions at specific floors are gated. Moving out of a gated position to an adjacent position on the same floor probabilistically results in the passenger being transported back to the first position on the first floor. The goal is achieved when the passenger has collected all the coins.

The tireworld problem requires that we guide a vehicle across a set of locations to a goal location. Moving from one location to another can result in a flat tire in which case the vehicle can no longer move until the tire is changed. The vehicle can carry only one spare tire, and can acquire a spare at designated locations. Although, the probability of successfully changing one tire is .5, the action can be repeated until the desired effect occurs.

Some of the competition instances for the domains in which we evaluated PLSPLAN do not feature in our experiments. In particular, we were unable to benchmark PLSPLAN on the blocks-world and exploding-blocks-world problems 11...15 because our computer running PGRAPHPLAN runs out of memory (2 Gigabytes) before generating the plan-graph.

## Results and Discussion

Our experiments were conducted on an AMD Opteron(tm) 64 Processor 2.6GHz machine with 2 gigabytes of RAM. The results are tabulated in Table 1. We set  $wp$  to 0.01 in all our experiments, and dynamically adjusted  $maxSteps$  so that it is two times the maximum number of iterations SLSPLAN has taken to solve  $\phi$  in a given run of PLSPLAN, or 100 thousand, whichever is greater. In our experiments, we always terminate PLSPLAN after 700 seconds. PLSPLAN could not solve any problems in under 10 minutes for blocks-world problems 6...11, exploding blocks-world problems 6...11, and elevator 10...15. In Table 1 we do not report those results.

In the first place our experimental results demonstrate the problem features that make PLSPLAN perform well. In particular: (1) If there are few, or no, low quality linear plans that achieve the goal, then we find good solutions quickly. This is the case in elevator problems after pre-processing with plan-graph techniques; (2) If there are many linear plans, only a few of which are good quality, then by

Instances	optimal		10% sampled		50% sampled		acc sampled	
	qual	secs	qual	secs	qual	secs	qual	secs
tw-p01-t10	0.21	0.02	6.22e-03	0.01	0.03	0.15	n/a	700
tw-p01-t5	0.13	0.00	0.13	0.77	0.13	83.87	0.13	4.56
tw-p02-t1	1.00	0.00	1.00	0.35	1.00	0.13	1.00	0.00
tw-p03-t2	0.60	0.00	0.60	0.13	0.60	0.43	0.60	0.06
tw-p03-t7	0.97	0.01	0.03	0.00	0.06	0.02	n/a	700
tw-p04-t3	0.36	0.00	0.36	0.14	0.36	4.10	0.36	0.36
tw-p04-t8	0.89	0.16	0.03	0.45	0.03	0.04	n/a	700
tw-p05-t2	0.60	0.00	0.60	0.04	0.60	1.33	0.60	0.07
tw-p05-t7	0.97	0.04	0.03	0.01	0.10	0.03	n/a	700
tw-p06-t2	0.60	0.00	0.60	0.02	0.60	0.26	0.60	0.12
tw-p06-t7	0.97	0.05	0.13	0.02	0.09	0.17	n/a	700
tw-p07-t3	0.36	0.00	0.36	0.60	0.14	36.37	0.36	0.01
tw-p07-t8	0.89	0.06	0.02	0.02	0.03	0.78	n/a	700
tw-p08-t2	0.60	0.00	0.60	0.03	0.60	0.00	0.60	0.02
tw-p08-t7	0.82	0.14	0.05	0.09	0.04	0.01	n/a	700
tw-p09-t3	0.36	0.00	0.36	0.13	0.36	0.62	0.36	0.02
tw-p09-t8	0.81	0.21	0.04	0.06	0.06	0.04	n/a	700
tw-p10-t1	1.00	0.00	1.00	0.05	1.00	0.01	1.00	0.00
tw-p11-t2	0.60	0.00	0.60	0.00	0.60	0.24	0.60	0.09
tw-p11-t7	0.97	0.03	0.16	0.08	0.03	0.02	n/a	700
tw-p12-t1	1.00	0.00	1.00	0.03	1.00	0.13	1.00	0.00
tw-p13-t2	0.60	0.00	0.60	0.02	0.60	0.50	0.60	0.00
tw-p13-t7	0.97	1.71	0.05	0.36	0.09	0.17	n/a	700
tw-p14-t2	0.60	0.00	0.60	0.00	0.60	0.29	0.60	0.00
tw-p14-t7	0.78	1.15	0.05	0.14	0.07	0.21	n/a	700
tw-p15-t3	0.36	0.00	0.36	0.39	0.36	1.71	0.36	0.03
tw-p15-t8	0.81	0.11	0.05	0.03	0.03	0.43	n/a	700
bw-p01-t12	0.11	0.83	8.34e-05	5.25	1.78e-04	5.04	n/a	700
bw-p01-t7	7.91e-04	0.01	7.91e-04	0.57	n/a	700	n/a	700
bw-p02-t10	6.26e-03	0.09	n/a	700	n/a	700	n/a	700
bw-p02-t15	0.52	2.56	n/a	700	n/a	700	n/a	700
bw-p03-t14	0.61	1.14	n/a	700	n/a	700	n/a	700
bw-p03-t9	0.03	0.04	3.34e-03	23.66	n/a	700	n/a	700
bw-p04-t10	0.01	0.09	n/a	700	n/a	700	n/a	700
bw-p04-t15	0.30	2.40	2.64e-04	31.42	n/a	700	n/a	700
bw-p05-t14	0.61	0.86	0.02	3.81	n/a	700	n/a	700
bw-p05-t9	0.03	0.03	n/a	700	n/a	700	n/a	700
ebw-p01-t6	1.00	0.00	0.06	0.76	n/a	700	n/a	700
ebw-p02-t4	1.00	0.00	0.19	2.76	0.10	119.89	0.09	8.22
ebw-p03-t6	1.00	0.00	n/a	700	n/a	700	n/a	700
ebw-p04-t12	0.36	0.01	n/a	700	n/a	700	n/a	700
ebw-p04-t17	0.54	0.07	n/a	700	n/a	700	n/a	700
ebw-p05-t2	1.00	0.00	1.00	1.51	1.00	0.00	1.00	0.02
el-p01-t13	1.00	0.02	0.50	0.79	n/a	700	n/a	700
el-p02-t8	1.00	0.00	1.00	0.00	1.00	0.01	1.00	36.13
el-p03-t15	1.00	0.01	n/a	700	n/a	700	n/a	700
el-p04-t13	1.00	0.00	1.00	0.34	n/a	700	n/a	700
el-p05-t11	1.00	0.01	n/a	700	n/a	700	n/a	700
el-p06-t20	0.50	0.15	n/a	700	n/a	700	n/a	700
el-p06-t25	1.00	0.20	n/a	700	n/a	700	n/a	700
el-p07-t22	1.00	0.19	n/a	700	n/a	700	n/a	700
el-p08-t30	0.50	1.31	n/a	700	n/a	700	n/a	700
el-p08-t35	n/a	700	n/a	700	n/a	700	n/a	700
el-p09-t29	0.50	0.74	n/a	700	n/a	700	n/a	700
el-p09-t34	0.50	1.65	n/a	700	n/a	700	n/a	700

Table 1: Results of evaluating PLSPLAN and PGRAPHPLAN in IPC problems. Column *optimal* lists the quality (*qual*) and time in seconds (*secs*) for PGRAPHPLAN. Column *50% sampled* list performance statistics for PLSPLAN in the case that  $n$  from Algorithm 4 is fixed to 50. Column *10% sampled* list for PLSPLAN with  $n$  fixed to 10. Column *acc sampled* lists for the case that an equal number of calls to SLSPLAN occurs for  $n \in \{1 \dots 50\}$ .

sampling the value of many chance variables, SLSPLAN quickly finds the good solutions. This is observed particularly for instances of tireworld.

We find that the biggest cost of our approach is from performing mutex control and constraint propagation in SLSPLAN. Indeed, PLSPLAN performs best in the elevator and tireworld domains. For both of these domains, instances have relatively few (compared with variations on blocks-world) actions to choose from at most timesteps. In the case

of elevator this is due mostly to the pruning of the solution space we achieve using probabilistic plangraph techniques. For tireworld, this is a natural feature of the domain.

For the parameter settings we choose, we found that PLSPLAN does not compute good quality plans for problems where an optimal solution accommodates many interrelated contingencies. This can be seen by the quality of the plan PLSPLAN finds for blocks-world instances. For these problems there are many linear plans that achieve the goal with low probability. SLSPLAN generated short linear plans by exploiting the ability to hold stacks of blocks, or otherwise became stuck in local minima. When PLSPLAN combined the linear plans generated by PLSPLAN, the resulting contingent plan was not effective. Sampling the chance variables does not mitigate this problem, as we find that high sampling in this case means that many of the CNF passed to SLSPLAN by PLSPLAN were unsatisfiable.

Overall, the performance of PLSPLAN is competitive with that of other entrants in the planning competition in small to medium sized problems from all domains we have examined except for the blocks-world. In larger benchmark problems, PLSPLAN was unable to attempt the problems either (1) Because we could not generate the SSAT encoding of the plangraph, or (2) Because it takes more than 700 seconds to find a plan that achieves the goal.

## Related & Future Work

PLSPLAN is related via the SLSPLAN procedure to recent developments in the *planning-as-satisfiability* paradigm for the deterministic case. In particular, SLSPLAN is related to SATPLAN, a deterministic planner that encodes  $n$ -stage problems posed by a plangraph as a propositional CNF formula. The state-of-the-art DPLL-based SAT solver SIEGE is then used to find a plan, or otherwise prove that one does not exist (Kautz, Selman, & Hoffmann 2006; Kautz & Selman 1999). From a reasoning under uncertainty perspective, related work includes the general SSAT solvers EVALSSAT and RANDEVALSSAT (Littman, Majercik, & Pitassi 2001), and the existing approaches to probabilistic planning with SSAT problem encodings: MAXPLAN (Majercik & Littman 1998a), DC-SSAT (Majercik & Boots 2005), Zander (Majercik & Littman 2003), and APPSAT (Majercik 2006). In Table 2 we relate PLSPLAN (last row) to these other solvers in terms of the type of solution method used, the class of problem targeted, and whether or not the solution technique is exact or approximate.

Of the available SSAT-based probabilistic planners, APPSSAT is most similar to PLSPLAN. APPSSAT iterates over the set of assignments of truth values to the random variables in the problem at hand, from most likely to least likely, and for each assignment fixes the truth values of the random variables in the CNF. APPSSAT then uses the DPLL-based SAT solver ZCHAFF to solve the resulting CNF. Thus, for a given horizon and for each possible choice of outcomes from actions that nature can make, ZCHAFF builds a linear plan (if one exists) that achieves the goal. As with PLSPLAN, successively generated linear plans are added to a single contingent plan. APPSSAT is inefficient

Table 2: Existing approaches to contingent-planning-as-satisfiability. **Solver:** lists the names of the solvers. **Class:** lists whether the solver is based on DPLL, a modified version of this M-DPLL, or SLS. **Prob:** lists whether the solver targets *unobservable*, *partially observable*, *fully observable*, or classical *deterministic* planning problems, or SSAT problems in general. **Opt:** lists whether the solution method produces optimal (O), or approximate (A) solutions.

Solver	Class	Prob	Opt
MAXPLAN	M-DPLL	Unobservable	O
ZANDER	M-DPLL	Partially Observable	O
DC-SSAT	M-DPLL	Fully Observable <sup>8</sup>	O
APPSSAT	DPLL	Partially Observable	A
EVALSSAT	M-DPLL	SSAT	O
RANDEVALSSAT	SLS	SSAT	A
PSLSPLAN	SLS	Fully Observable	A

in situations where the only linear plans that achieve a goal do so with a low probability. It is also inefficient when the solution space is large and ZCHAFF spends all its time generating good but unrelated linear plans, rather than a group of interrelated plans which form an excellent contingent plan.

From an SLS standpoint there are a number of directions we could take our work in the future to make it more effective. For instance, we did not explore the use of a *novelty+* mechanism for escaping local minima during the search (McAllester, Selman, & Kautz 1997). It would be very interesting to see how an unweighted SLS that uses *Novelty+* — or a variation of this such as *AdaptNovelty+* — to escape from minima competes with our approach that uses clause weights.

From the standpoint of encoding the planning problem as SSAT there are a number of improvements that our work should consider in the future. For instance, we could include the pairwise neediness constraint from (Blum & Langford 1999) in the CNF component of our SSAT problem encoding, or perhaps as a constraint that is maintained by SLS-PLAN. Respecting such constraints should greatly effect plan search time in domains where a condition has to hold true until some milestone is achieved. Along the same lines, it might be a good idea to compute and then include mutex constraints between propositions at the layers of the plan-graph. Essentially, in the future we should examine strategies for mutex control and constraint propagation which better address the trade-off between guiding the search, and the speed of the search.

## Acknowledgements

Thanks to Stephen Majercik for useful discussions. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

## References

Ansótegui, C.; Larrubia, J.; Li, C. M.; and Manyà, F. 2003. Mv-Satz: A SAT solver for many-valued clausal forms. In *JIM*.

- Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* (90):281–300.
- Blum, A., and Langford, J. 1999. Probabilistic planning in the graphplan framework. In *ECP*, 319–332.
- Chen, Y.; Xing, Z.; and Zhang, W. 2007. Long-distance mutual exclusion for propositional planning. In *Proc. IJCAI*.
- Frisch, A. M., and Peugniez, T. J. 2001. Solving non-Boolean satisfiability problems with stochastic local search. In *IJCAI*, 282–290.
- Hoos, H. H., and Stützle, T. 2004. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.
- Hutter, F.; Tompkins, D. A.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *CP*, 233–248.
- Kautz, H. A., and Selman, B. 1999. Unifying SAT-based and Graph-based planning. In *IJCAI*, 318–325.
- Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In *In Proc. Abstracts of the 5th International Planning Competition*.
- Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research* 9:1–36.
- Littman, M. L.; Majercik, S. M.; and Pitassi, T. 2001. Stochastic boolean satisfiability. *Journal of Automated Reasoning* 27(3):251–296.
- Majercik, S. M., and Boots, B. 2005. DC-SSAT: A divide-and-conquer approach to solving stochastic satisfiability problems efficiently. In *AAAI*, 416–422.
- Majercik, S. M., and Littman, M. L. 1998a. MAXPLAN: A new approach to probabilistic planning. In *AIPS*, 86–93.
- Majercik, S. M., and Littman, M. L. 1998b. Using caching to solve larger probabilistic planning problems. In *AAAI/IAAI*, 954–959.
- Majercik, S. M., and Littman, M. L. 2003. Contingent planning under uncertainty via stochastic satisfiability. *Artif. Intell.* 147(1-2):119–162.
- Majercik, S. M. 2006. APPSSAT: Approximate probabilistic planning using stochastic satisfiability. *International Journal of Approximate Reasoning*.
- McAllester, D. A.; Selman, B.; and Kautz, H. A. 1997. Evidence for invariants in local search. In *AAAI*, 321–326.
- Pham, D. N.; Thornton, J.; Sattar, A.; and Ishtaiwi, A. 2005. SAT-based versus CSP-based constraint weighting for satisfiability. In *AAAI*, 455–460.
- Pham, D. N.; Thornton, J.; and Sattar, A. 2007. Building structure into local search for SAT. In *IJCAI*, 2359–2364.
- Pham, D. N.; Thornton, J.; and Sattar, A. 2008. Efficiently exploiting dependencies in local search for SAT. In *AAAI*, to appear.
- Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *AAAI*, 337–343.
- Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *AAAI*, 440–446.

Thornton, J.; Pham, D. N.; Bain, S.; and Ferreira Jr., V. 2004. Additive versus multiplicative clause weighting for SAT. In *AAAI*, 191–196.

Wu, Z., and Wah, B. W. 2000. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *AAAI*, 310–315.