A New d-DNNF-Based Bound Computation Algorithm for Functional E-MAJSAT

Knot Pipatsrisawat and **Adnan Darwiche**
Computer Science Department
University of California, Los Angeles
{thammakn,darwiche}@cs.ucla.edu

Abstract

We present a new algorithm for computing upper bounds for an optimization version of the E-MAJSAT problem called functional E-MAJSAT. The algorithm utilizes the compilation language d-DNNF which underlies several state-of-the-art algorithms for solving related problems. This bound computation can be used in a branch-and-bound solver for solving functional E-MAJSAT. We then present a technique for pruning values from the branch-and-bound search tree based on the information available after each bound computation. We evaluated the proposed techniques in a MAP solver and a probabilistic conformant planner. In both cases, our experiments showed that the new techniques improved the efficiency of state-of-the-art solvers by orders of magnitude.

1 Introduction

E-MAJSAT [Littman *et al.*, 1998] is an extension of the boolean satisfiability problem that can be used to model real-world problems involving uncertainty. E-MAJSAT is also a special case of a more general class of problems called stochastic satisfiability (SSAT) [Littman *et al.*, 2001]. Given a CNF Δ over variables $\mathbf{E} \cup \mathbf{R}$, E-MAJSAT asks whether there exists an assignment $\mathbf{e}$ to the variables in $\mathbf{E}$ such that the majority of complete assignments to $\mathbf{R}$ satisfy the formula Δ conditioned on $\mathbf{e}$. This problem is $\mathsf{NP}^{\mathsf{PP}}$-complete [Littman *et al.*, 1998; Park, 2002], as we need to search in an exponential search space (the NP part) while checking whether a candidate constitutes a solution requires solving a counting problem (the PP part).

E-MAJSAT is an important problem in AI as it can be used to model many problems such as probabilistic conformant planning [Drummond and Bresina, 1990; Hanks, 1990; Kushmerick *et al.*, 1995], finding maximum a posteriori hypothesis (MAP) [Park and Darwiche, 2004], and solving for maximum expected utility (MEU) [Dechter, 1996]. Many exact algorithms for solving E-MAJSAT and related problems have been proposed. For example, in [Littman, 1999; Littman *et al.*, 2001], the authors proposed a modified version of the DPLL algorithm [Davis *et al.*, 1962] for solving SSAT, Dechter used bucket elimination for solving MAP [1996], while recursive conditioning was used for solving the same problem in [Darwiche, 2000].

In this work, we investigate an optimization version of E-MAJSAT called *functional E-MAJSAT*. We propose an algorithm for computing upper bounds on the solution of this problem. The proposed algorithm can be viewed as an improvement of those used in [Huang *et al.*, 2006; Huang, 2006], which take advantage of a compilation language called d-DNNF for computing upper bounds. In this work, we point out the cause of bound looseness in these existing algorithms and propose a method for reducing the inaccuracy in bound values. We then discuss an integration of the algorithm into a branch-and-bound solver for solving functional E-MAJSAT. Next, we describe a technique of using information available from the new bound computation to dynamically prune branches of the search tree to speed up the search. For evaluation, we integrate our techniques into branch-and-bound solvers for finding maximum a posteriori hypothesis and for solving probabilistic conformant planning problems. These solvers are, by themselves, state-of-the-art solvers for the respective problems. Our results show that the proposed techniques improve these solvers by orders of magnitude.

The rest of the paper is organized as follows. We first discuss basic notations and definitions in Section 2. Then, in Section 3, we review existing techniques for solving and computing bounds of functional E-MAJSAT based on d-DNNF. In Section 4, we present a new algorithm for computing tighter bounds and discuss some of its properties. Section 5 discusses its integration with a branch-and-bound solver and presents a pruning technique based on our new bound computation. Experimental results are presented in Section 6 and we conclude in Section 7.

2 Basic Notations and Definitions

Given a literal ℓ, which is either a variable or the negation of a variable, we use $var(\ell)$ to refer to its variable. An *assignment* is simply a consistent set of literals (interpreted as their conjunction). If Δ is a propositional sentence and $\mathbf{s}$ is an assignment, we write $\mathbf{s} \models \Delta$ iff $\mathbf{s}$ satisfies Δ. Moreover, we use $\Delta|\mathbf{s}$ to denote the formula obtained from Δ by substituting every literal $\ell \in \mathbf{s}$ with $\mathbf{true}$ and every literal ℓ such that $\neg\ell \in \mathbf{s}$ with $\mathbf{false}$.

Unless stated otherwise, in this paper, we will assume that every variable of a given formula has been designated as either a *choice variable* or a *chance variable*. Intuitively, choice

variables are the ones we get to assign, while chance variables will be assigned by the nature. We assume that the probabilities θ of the literals of the chance variables are given. We use $\theta(\ell)$ to denote the probability of a chance literal ℓ. If Ψ is a formula containing only chance variables, then we define the *probability* of Ψ to be

$$\mathsf{Pr}(\Psi) = \sum_{\mathbf{r} \models \Psi} \left(\prod_{\ell \in \mathbf{r}} \theta(\ell) \right).$$

The summation above is over all complete assignments $\mathbf{r}$ of the chance variables that satisfy Ψ. The probability of Ψ is essentially the weighted model count of Ψ, where the weight of each model is simply the product of the probabilities of its chance literals.[1]

Given a CNF Γ, the functional E-MAJSAT problem on Γ is to compute

$$M = \max_{\mathbf{e}} \mathsf{Pr}(\Gamma|\mathbf{e}),$$

where $\mathbf{e}$ is a complete assignment over the choice variables. Notice that $\Gamma|\mathbf{e}$ is a formula containing only chance variables. We will refer to M as the *maximum probability* of the functional E-MAJSAT problem on Γ.

3 Functional E-MAJSAT Bound Computation with d-DNNF

Our approach to bound computation is based on knowledge compilation. In particular, given a functional E-MAJSAT problem on CNF Γ, we compile Γ into a d-DNNF and compute bounds from it. In this section, we will discuss an existing approach for computing upper bounds based on d-DNNF. We begin by talking briefly about d-DNNF.

3.1 Deterministic Decomposable Negation Normal Form (d-DNNF)

A negation normal form (NNF) is a rooted DAG whose leaves are truth constants or literals and whose internal nodes are either AND nodes or OR nodes. A deterministic decomposable negation normal form (d-DNNF) [Darwiche and Marquis, 2002] is simply an NNF which satisfies the following properties: (1) decomposability: children of each AND node do not share a variable (2) determinism: children of each OR node do not share a model. These properties allow many operations such as consistency checking and model counting to be performed efficiently [Darwiche and Marquis, 2002]. Figure 1 shows two d-DNNFs (letters and numbers will be explained later).

In this work, we use C2D to compile CNF into d-DNNF.[2] The output d-DNNFs produced by C2D satisfy the following key property: every OR node is of the form $\alpha = (x \wedge \beta) \vee (\neg x \wedge \gamma)$. Here, x is called the *decision variable* of α, denoted $dec(\alpha)$, and is made available in the output of C2D. We will assume that every d-DNNF discussed here has this special form. Moreover, since every variable in a functional E-MAJSAT problem is either choice or chance, we will say that an OR node is a choice (chance) OR node if its decision variable is a choice (chance) variable. Decision variables of OR nodes in Figure 1 are shown in brackets.

Given a d-DNNF compiled from a functional E-MAJSAT problem, the relative positions of the choice and chance variables in the d-DNNF affect the quality of bounds computed from the d-DNNF. A d-DNNF is said to be *constrained* if no choice variable appears below any chance OR node. It is *unconstrained* otherwise. If x, y are the choice variables, then the d-DNNF in Figure 1 (a) is constrained (i.e., no choice literal appears below any chance OR node) while the one in Figure 1 (b) is unconstrained (i.e., x, y appear below the root, which is a chance OR node).

3.2 Functional E-MAJSAT Bound Computation Using d-DNNF

Given a functional E-MAJSAT problem, if the corresponding d-DNNF is constrained, we can easily compute the maximum probability of the problem with a linear-time traversal of the d-DNNF [Huang *et al.*, 2006]. More generally, given a constrained d-DNNF Δ and a (not necessarily complete) assignment $\mathbf{s}$ of the choice variables, we can compute the maximum probability of the respective functional E-MAJSAT problem as follows. We perform a bottom-up traversal of Δ and, for each node α, we compute its value $val^{\alpha}_{\mathbf{s}}$, defined as

$$\begin{cases} \theta(\alpha), & \text{if } \alpha \text{ is a chance literal} \\ 0, & \text{if } \alpha \text{ is a literal falsified by } \mathbf{s} \\ 1, & \text{if } \alpha \text{ is a literal not falsified by } \mathbf{s} \\ \prod_i val^{\alpha_i}_{\mathbf{s}}, & \text{if } \alpha = \bigwedge_i \alpha_i \\ val^{\alpha_1}_{\mathbf{s}} + val^{\alpha_2}_{\mathbf{s}}, & \text{if } \alpha = \alpha_1 \vee \alpha_2 \text{ and} \\ & dec(\alpha) \text{ is a chance variable.} \\ \max(val^{\alpha_1}_{\mathbf{s}}, val^{\alpha_2}_{\mathbf{s}}), & \text{if } \alpha = \alpha_1 \vee \alpha_2 \text{ and} \\ & dec(\alpha) \text{ is a choice variable.} \end{cases}$$

The maximum probability of the problem is simply the value at the root, as stated by the following result.[3]

Proposition 1 *Given a CNF Γ and an equivalent constrained d-DNNF Δ, $val^{\Delta}_{\mathbf{true}}$ is the maximum probability of the functional E-MAJSAT problem on Γ.*

When the d-DNNF is constrained, it is not hard to see that maximization always comes after summation and that the above algorithm computes the values according to the definition of the problem given in Section 2.

Consider the d-DNNF in in Figure 1 (a) again. We set $\theta(a) = 0.8, \theta(b) = 0.6, \theta(c) = 0.8, \theta(e) = 0.5$. Using the above algorithm, we get the value of the root $val^{\Delta}_{\mathbf{true}} = 0.34$, which is the solution to this problem. The value of each node according to this algorithm is shown inside parentheses.

Requiring the compilation output to be a constrained d-DNNF is, however, often impractical as the complexity is exponential in the *constrained treewidth* of the CNF [Park and Darwiche, 2003].[4] Unconstrained d-DNNF offers a more

[1] Note that this definition does not require Ψ to be in any particular form.

[2] Available at http://reasoning.cs.ucla.edu/c2d.

[3] A similar claim was made in [Huang *et al.*, 2006] without a proof.

[4] The constrained treewidth is the minimal treewidth induced by any variable order in which the choice variables are eliminated last.

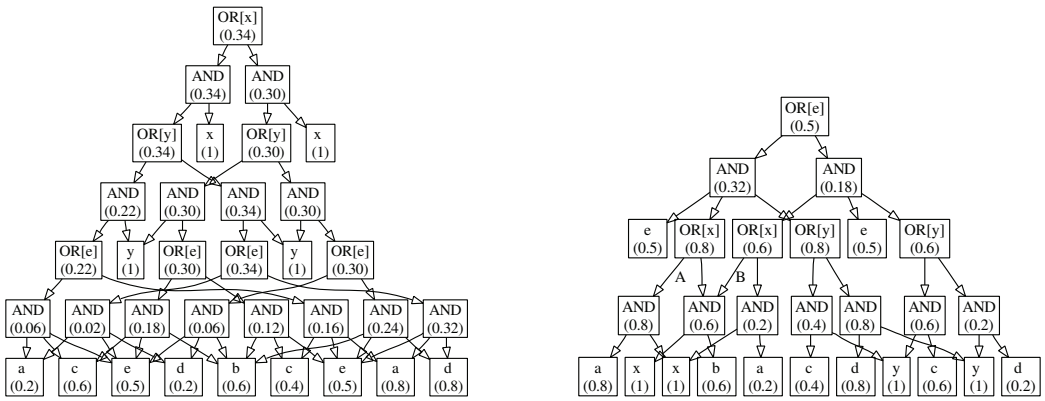

Figure 1: (a) a constrained d-DNNF in which x and y are choice variables. (b) an unconstrained d-DNNF. Every node is labeled with a $val^{\alpha}_{\mathbf{true}}$ and each OR node is also labeled with its decision variable.

practical solution in this case, as the compilation will only be exponential in the treewidth [Darwiche, 2001], which could be significantly smaller than the constrained treewidth according to [Park and Darwiche, 2003]. However, the trade-off here is that the value of the root node (given by the above algorithm) will no longer be the exact solution of the problem. Instead, it becomes an upper bound [Huang *et al.*, 2006]. Nevertheless, such a bound could still be very useful for branch-and-bound solvers as demonstrated in [Huang, 2006] and [Huang *et al.*, 2006]. Figure 1 (b) shows the computation of an upper bound of the same functional E-MAJSAT problem as the previous example. In this case, the root's value is 0.5, which is larger than the actual solution.

3.3 The Cause of Bound Looseness

Roughly speaking, the more unconstrained the d-DNNF is, the looser the bounds tend to become. Let us now illustrate the cause of bound looseness with an example. Consider again the bound computation performed on the d-DNNF in Figure 1 (b). There are two AND nodes at depth 1 (whose values are 0.32 and 0.18 respectively). The value of the left AND node is partially attributed to the part of the formula that assigns $x = \mathbf{true}$ (indicated by the edge labeled with A). On the other hand the value of the right AND node is partially due to the part of the formula that assigns $x = \mathbf{false}$ (indicated by the edge labeled with B). Eventually, the values of these two AND nodes are combined at the root, because it is a chance OR node. Therefore, the value 0.5 at the root does not correspond to a valid assignment, because x cannot be assigned to both **true** and **false**. Note that the situation described here could only arise in unconstrained d-DNNFs.

4 Computing Tighter Bounds Using Option Pairs

In this section, we present an algorithm for computing tighter bounds for functional E-MAJSAT. Operating on unconstrained d-DNNF, our solution reduces the extent to which invalid assignments affect the bound values. The key idea is to compute bound values that are conditioned on the values of choice variables. To do so, we will need to make more information available at each node of the d-DNNF. The following definition is needed in the discussion of our algorithm.

Definition 1 (Option Pair) *Given a d-DNNF node α, a partial instantiation of the choice variables $\mathbf{s}$ and a choice variable x not mentioned by $\mathbf{s}$, $\psi = (x, op^+, op^-)$ is an option pair of α if*

- *op^+ is an upper bound of $val^{\alpha}_{x \wedge \mathbf{s}}$ and*
- *op^- is an upper bound of $val^{\alpha}_{\neg x \wedge \mathbf{s}}$.*

In this case, x is called the option variable, denoted $v(\psi)$. We will also call op^+ the positive option ($p(\psi)$) and call op^- the negative option ($n(\psi)$) of ψ. The best option of ψ is simply $\max(op^+, op^-)$.

An option pair contains bounds on the node's values conditioned on the values of a choice variable. In the new bound computation, instead of computing a value for each d-DNNF node, we compute option pairs on all free choice variables that appears below the node (if they exist). Thus, it is possible for a node to have zero or multiple option pairs. Before we describe an algorithm for computing option pairs, we need some definitions for the value of an option pair and the value of a node, under an assignment. Given an assignment $\mathbf{s}$ to some choice variables, the *contribution* of an option pair ψ is the largest option value it can contribute under the assignment. This is defined formally as

$$\kappa(\psi, \mathbf{s}) = \begin{cases} p(\psi), & \text{if } \mathbf{s} \models v(\psi) \\ n(\psi), & \text{if } \mathbf{s} \models \neg v(\psi) \\ \max(p(\psi), n(\psi)), & \text{otherwise.} \end{cases}$$

In light of this definition, we can tighten the value of each node. If a node α has at least one option pair, we define its new value $oval^{\alpha}_{\mathbf{s}}$ to be the smallest contribution from any option pair of the node. This is the tightest bound we can obtain from the available information, because any complete assignment must set every choice variable to a value. If the node does not have an option pair, its value is defined to be $val^{\alpha}_{\mathbf{s}}$

(as defined in Section 3). Formally, we have

$$oval^{\alpha}_{\mathbf{s}} = \begin{cases} \min_{\psi} \kappa(\psi, \mathbf{s}), & \alpha \text{ has some option pairs} \\ val^{\alpha}_{\mathbf{s}}, & \text{otherwise.} \end{cases}$$

Given an assignment $\mathbf{s}$ and a node α that mentions a free choice variable v, we compute its option pair on v as follows.

1. If α is a literal of v, its option pair on v is $(v, \theta(\alpha), 0)$ if α is positive, and $(v, 0, \theta(\alpha))$ if α is negative.
2. If $\alpha = \bigwedge_{i=1}^{n} \alpha_i$, its option pair on v is
$$(v, \prod_{i=1}^{n} oval^{\alpha_i}_{\mathbf{s}\wedge v}, \prod_{i=1}^{n} oval^{\alpha_i}_{\mathbf{s}\wedge \neg v}).$$
3. If $\alpha = \alpha_1 \vee \alpha_2$ is a choice OR, its option pair on v is
$$(v, \max(oval^{\alpha_1}_{\mathbf{s}\wedge v}, oval^{\alpha_2}_{\mathbf{s}\wedge v}), \max(oval^{\alpha_1}_{\mathbf{s}\wedge \neg v}, oval^{\alpha_2}_{\mathbf{s}\wedge \neg v})).$$
4. If $\alpha = \alpha_1 \vee \alpha_2$ is a chance OR, its option pair on v is
$$(v, oval^{\alpha_1}_{\mathbf{s}\wedge v} + oval^{\alpha_2}_{\mathbf{s}\wedge v}, oval^{\alpha_1}_{\mathbf{s}\wedge \neg v} + oval^{\alpha_2}_{\mathbf{s}\wedge \neg v}).$$

This algorithm can be repeated at each node on every free choice variable to compute all option pairs. The bound value produced by this algorithm is then the value of the root node ($oval^{\Delta}_{\mathbf{s}}$). The time complexity of this computation is $O(|\mathbf{E}||\Delta|)$, where $|\mathbf{E}|$ is the number of choice variables and $|\Delta|$ is the size of the d-DNNF. In practice, we may choose not to compute all option pairs at each node. In this case, a heuristic is needed to select which pairs should be computed.

The source of improvement of this new bound algorithm lies in Step 4 above. In this case, instead of simply adding the highest values that each child may produce together, we only add values that agree on the assignment of the given option variable together. The value of such an OR node under the new bound algorithm will be lower than the one computed the previous algorithm (without option pairs) whenever the best options of the children correspond to conflicting assignments of the option variable. Therefore, the quality of the bounds generated by option pairs is more sensitive to changes in the parameters of the problems (i.e., the probabilities of the chance literals). In any case, we have the following result regarding the bounds computed this way.

Proposition 2 *Given a d-DNNF Δ and an instantiation (possibly partial) $\mathbf{s}$ of choice variables, we have*

$$val^{\Delta}_{\mathbf{s}} \geq oval^{\Delta}_{\mathbf{s}} \geq \max_{\mathbf{x}} \mathsf{Pr}(\Delta|(\mathbf{s} \wedge \mathbf{x})),$$

where $\mathbf{x}$ is a complete assignment of the remaining choice variables.

This result shows that the values computed using option pairs are correct upper bounds that are at least as tight as those computed using the basic algorithm given in Section 3. Note that both the existing and our proposed algorithms return the exact solution if the input assignment is a complete assignment to the choice variables.

Let us now illustrate the new bound algorithm with an example. Consider the d-DNNF in Figure 2. This d-DNNF is identical to the one in Figure 1 (b) except the nodes' values

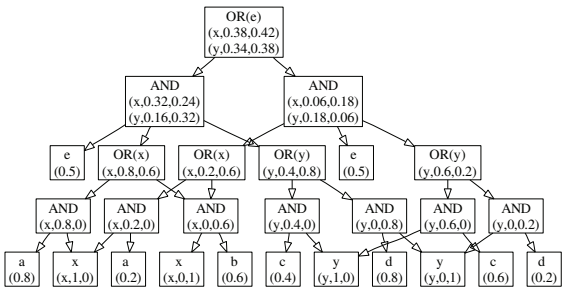

Figure 2: Bound computation using option pairs.

are now computed with option pairs. In this example, there are still two choice variables, x, y. We will now explain the computation of option pairs of some nodes in this d-DNNF.

Each leaf node labeled with a chance variable in this figure is only annotated with its value, because it has no option pairs. Other leaves are labeled with option pairs on their choice variables. Every choice OR node (all at depth 2) mentions only one choice variable (which is its decision variable). Their option pairs are obtained by simply combining the best option from each child. Each of the AND nodes at depth 1 mentions two choice variables and will have two option pairs. Consider the left AND node. For the option pair on x, the positive option is obtained by multiplying the positive option of the child that mentions x (0.8) with the values of the remaining children, which do not mention x (0.5 and 0.8). The negative option and the option pair on y can be computed in a similar way. To compute the option pair at the root node, which is a chance OR node, we simply add the compatible options of the children together. In the case of x, the option pair of the left child indicates that if $x = \mathbf{true}$, its value is no more than 0.32, while the right child indicates that its value is no more than 0.06 under the same assignment. As a result, we can conclude that, if $x = \mathbf{true}$, the root's value can be no more than 0.38. The other option values at the root can be computed in a similar way. Finally, from this bound computation, we can conclude that, no matter what value is assigned to y, the value of the root cannot be larger than 0.38.[5] Hence, we have obtained a bound value that is tighter than 0.5, the value computed by the algorithm in the previous section.

5 Utilizing the New Bound Computation in a Branch-and-Bound Algorithm

One natural application of our new bound computation is in a branch-and-bound search algorithm for solving functional E-MAJSAT. Many branch-and-bound algorithms for solving problems related to E-MAJSAT have been proposed previously [Littman, 1999; Littman *et al.*, 2001; Park and Darwiche, 2003; Huang *et al.*, 2006; Huang, 2006].

Such an algorithm searches in the space of all possible assignments to the choice variables. Each leaf of this search tree corresponds to a complete assignment $\mathbf{e}$ of the choice variables and is associated with the probability $\mathsf{Pr}(\Delta|\mathbf{e})$. Solving functional E-MAJSAT is equivalent to finding the leaf

[5] The same analysis on x yields a looser bound of 0.42. We take the smallest value as the bound.

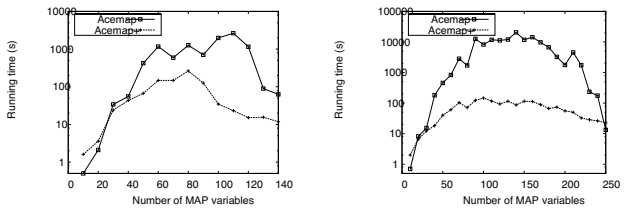

Figure 3: Running time of ACEMAP and ACEMAP+ on grid-50-12-2 (left) and grid-75-16-2 (right).

with the highest probability. During the search, the algorithm keeps track of the highest probability of any seen leaf (LB). Then, at each search node, it computes an upper bound on the probability of any leaf below the node. The current search branch can be pruned if the bound is no better than LB. We refer the reader to [Huang *et al.*, 2006] for a pseudo code of such a branch-and-bound search algorithm. The bound computation algorithm proposed here can be utilized in any standard branch-and-bound algorithm for solving E-MAJSAT-related problems.

5.1 Pruning Using Option Pairs

We will now present a technique for pruning branches of the branch-and-bound search tree. Consider a situation where we have the following option pairs at the d-DNNF root: $(x, 0.75, 0.85), (y, 0.9, 0.7), (z, 0.5, 0.88), (w, 0.81, 0.88)$. The current bound value is 0.85–the smallest best option. Let us assume further that the current value of LB is 0.8. At this point, we cannot prune the search tree yet. However, we know that if we set x to $\mathbf{true}$, the bound value will suddenly be smaller than LB. The same can be concluded about setting $y = \mathbf{false}$, and $z = \mathbf{true}$. Therefore, we can prune the following values from the current sub-tree: $x = \mathbf{true}, y = \mathbf{false}, z = \mathbf{true}$.

In general, after computing a bound using option pairs, we inspect each option pair ψ of the root node, if $p(\psi) \leq LB$, then the branch $v(\psi) = \mathbf{true}$ can be removed. If $n(\psi) \leq LB$, then the branch $v(\psi) = \mathbf{false}$ can be removed.[6] During this process, the value of each choice variable can be pruned independently, no matter what values of other variables are pruned. This process of computing a bound and removing values can be repeated as long as a new value is removed.

6 Experimental Results

We integrated our techniques into two state-of-the-art solvers from two different domains derived from functional E-MAJSAT: MAP and probabilistic conformant planning. The following subsections discuss the integrations and results. All of the experiments presented here were conducted on a Pentium 4, 3.8GHz machine with 4GB of RAM.

6.1 Finding Maximum a Posteriori Hypothesis

Given a Bayesian network over variables $\mathbf{X}$, the maximum a posteriori (MAP) hypothesis over a set of MAP variables $\mathbf{M} \subseteq \mathbf{X}$ is an assignment of $\mathbf{M}$ with the highest probability. Solving MAP is $\mathsf{NP}^{\mathsf{PP}}$-complete [Park and Darwiche, 2004] and can be naturally reduced to functional E-MAJSAT; the MAP variables are choice variables, while the remaining are chance variables. In our experiment, we integrated our techniques into a state-of-the-art solver called ACEMAP [Huang *et al.*, 2006]. ACEMAP is a branch-and-bound MAP solver which computes bounds using d-DNNF. We called the solver after the integration ACEMAP+.

We experimented with MAP problems used in [Huang *et al.*, 2006]. For each Bayesian network, we generated 10 MAP queries by selecting random subsets of variables to be the MAP variables. For the 42 grid networks (see [Huang *et al.*, 2006]), ACEMAP used 170,478 seconds to solve all queries, while ACEMAP+ used only 7,825 seconds (>20x speedup).[7] The other non-grid networks (some are used in [Huang *et al.*, 2006]) are relatively easy as both solvers finished many of them in seconds.[8] In any case, ACEMAP used 2,983 seconds to solve all queries, while ACEMAP+ used only 636 seconds.

We also evaluated both solvers on MAP queries with different difficulty levels. For this experiment, we used the networks grid-50-12-2 and grid-75-16-2, which contain 144 and 256 network variables respectively. For each network, we generated MAP queries with varied numbers of MAP variables. Figure 3 shows the running time of the solvers on a log-scale. The results from both networks show that ACEMAP+ outperformed ACEMAP by up to 2 orders of magnitude.

6.2 Solving Probabilistic Conformant Planning

Probabilistic conformant planning allows uncertainty in both the initial state and the action outcomes. Given a planning domain and a horizon N, the objective is to find a linear sequence of N actions with the highest probability of reaching the goal. We integrated our techniques into ComPlan [Huang, 2006] which is a state-of-the-art probabilistic conformant planner. ComPlan is a branch-and-bound solver that utilizes d-DNNF for bound computation. We implemented the planner based on the descriptions in [Huang, 2006]. We refer to the version of the planner with our techniques ComPlan+.

We compared the performance of ComPlan and ComPlan+ on the planning domains sand-castle [Majercik and Littman, 1998] and slippery-gripper [Kushmerick *et al.*, 1995] (as extended by [Hyafil and Bacchus, 2003]).[9] Figure 4 (left) shows both planners' running time on sand-castle on a log-scale. This plot shows that ComPlan+ is exponentially faster than ComPlan. When the horizon is 44, ComPlan takes 16,152 seconds, while ComPlan+ only takes 104 seconds (155x speedup). In our experiment (result not shown), ComPlan+ can solve the problem with horizon equals to 50 in less than 700 seconds, while ComPlan does not finish after a day.

We experimented with two versions of the slippery-gripper domain. The first one is the slippery-gripper problems de-

[6] If both values of a variable are pruned, the search algorithm can backtrack.

[7] As in [Huang *et al.*, 2006], each MAP query for these network contains 100 MAP variables (or all variables if the network has fewer than 100 variables).

[8] These include the block_map, mastermind, alarm, hailfinder, students families.

[9] The variables that represent plans are the choice variables and those that represent uncertainty are the chance variables.

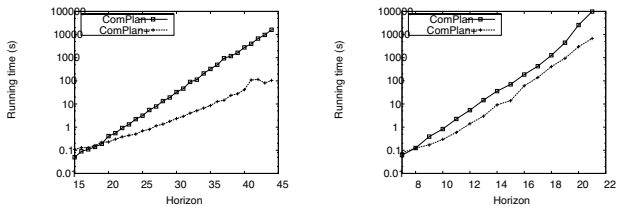

Figure 4: Running time of ComPlan and ComPlan+ on sandcastle (left) and modified slippery-gripper (right)

scribed in [Hyafil and Bacchus, 2003]. According to our results, ComPlan+ is about 1.8x faster than ComPlan on this set of problems. The second version differs from the original one only in the probabilities of success of some actions. In particular, we changed the probability of DRY being successful from 0.8 to 0.9, changed the probability that PAINT will make the gripper which is not holding the block dirty from 0.1 down to 0.05, and changed the probability of success of PICKUP when the gripper is wet from 0.5 to 0.85. These modifications make the probabilities of the actions more polarized. This modification is expected to expose the drawbacks of the existing bound algorithm more. Figure 4 (right) shows the running time of the planners on the modified version of slippery-gripper on a log-scale. The difference between ComPlan and ComPlan+ becomes more significant on this modified domain. According to this plot, when the horizon is 20, ComPlan takes 25,745 seconds, while ComPlan+ uses only 2,992 seconds (8.6x speedup). When the horizon is 21, ComPlan takes 98,359 seconds, while ComPlan+ takes 6,753 seconds (14.6x speedup).

7 Conclusions

In this paper, we proposed a new bound computation, based on the use of d-DNNF, for functional E-MAJSAT. The algorithm can be used in a branch-and-bound solver for functional E-MAJSAT and related problems. In addition to yielding tighter bounds, the new algorithm also produces additional information that can be used to prune search branches at virtually no cost. We integrated the new techniques into state-of-the-art branch-and-bound solvers for MAP and probabilistic conformant planning. Our results show orders of magnitude of improvement.

References

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[Darwiche, 2000] Adnan Darwiche. Any-space probabilistic inference. In *UAI-00*, pages 133–142, 2000.

[Darwiche, 2001] Adnan Darwiche. On the tractability of counting theory models and its application to belief revision and truth maintenance. *JANCL*, 11(1-2):11–34, 2001.

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

[Dechter, 1996] Rina Dechter. Bucket elimination: a unifying framework for probabilistic inference. In *Proceedings of UAI-96*, pages 211–219, 1996.

[Drummond and Bresina, 1990] Mark Drummond and John Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of AAAI-90*, pages 138–144, 1990.

[Hanks, 1990] Steven John Hanks. *Projecting plans for uncertain worlds*. PhD thesis, New Haven, CT, USA, 1990.

[Huang *et al.*, 2006] Jinbo Huang, Mark Chavira, and Adnan Darwiche. Solving map exactly by searching on compiled arithmetic circuits. In *AAAI-06*, pages 143–148, 2006.

[Huang, 2006] Jinbo Huang. Combining knowledge compilation and search for conformant probabilistic planning. In *ICAPS-06*, pages 253–262, 2006.

[Hyafil and Bacchus, 2003] Nathanael Hyafil and Fahiem Bacchus. Conformant probabilistic planning via csps. In *ICAPS*, pages 205–214, 2003.

[Kushmerick *et al.*, 1995] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[Littman *et al.*, 1998] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *JAIR*, 9:1–36, 1998.

[Littman *et al.*, 2001] Michael L. Littman, Stephen M. Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *J. Autom. Reason.*, 27(3):251–296, 2001.

[Littman, 1999] Michael L. Littman. Initial experiments in stochastic satisfiability. In *AAAI '99/IAAI '99*, pages 667–672, 1999.

[Majercik and Littman, 1998] Stephen M. Majercik and Michael L. Littman. Maxplan: A new approach to probabilistic planning. In *AIPS*, pages 86–93, 1998.

[Park and Darwiche, 2003] J. Park and A. Darwiche. Solving map exactly using systematic search. In *Proceedings of UAI-03*, pages 459–468, 2003.

[Park and Darwiche, 2004] James Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *JAIR*, 21:101–133, 2004.

[Park, 2002] J. Park. Map complexity results and approximation methods. In *UAI-02*, pages 388–396, 2002.