A Dynamic Approach to MPE and Weighted MAX-SAT

Tian Sang[1], **Paul Beame**[1], **and Henry Kautz**[2]
[1] Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195
{sang,beame}@cs.washington.edu
[2] Department of Computer Science
University of Rochester, Rochester, NY 14627
kautz@cs.rochester.edu

Abstract

The problem of Most Probable Explanation (MPE) arises in the scenario of probabilistic inference: finding an assignment to all variables that has the maximum likelihood given some evidence. We consider the more general CNF-based MPE problem, where each literal in a CNF-formula is associated with a weight. We describe reductions between MPE and weighted MAX-SAT, and show that both can be solved by a variant of weighted model counting. The MPE-SAT algorithm is quite competitive with the state-of-the-art MAX-SAT, WCSP, and MPE solvers on a variety of problems.

1 Introduction

Constraint Satisfaction Problems (CSP) have been the subject of intensive study; many real-world domains can be formalized by CSP models and solved by either complete or incomplete reasoning methods. Beyond classic CSP, where a solution must satisfy all hard constraints, some CSP models are capable of handling both hard and soft constraints. The definition of constraints and the measurement of the quality of a solution vary from model to model, and the goal is usually to find a best solution to the constraints, rather than simply a solution. For example, the constraints can be associated with probability, cost, utility, or weight; the goal can be to minimize costs of violated constraints, or to maximize the likelihood of a variable assignment, etc. However, all fit in a general framework for soft constraints, namely, semi-ring based CSPs [Bistarelli *et al.*, 1997]. In this paper, we focus on two specific models: MPE and weighted MAX-SAT.

MAX-SAT extends SAT to the problem of finding an assignment that maximizes the number of satisfied clauses, in case the formula is unsatisfiable. Weighted MAX-SAT extends MAX-SAT by adding a weight to each clause, with the goal of finding an assignment that maximizes the sum of weights of satisfied clauses. MAX-SAT and weighted MAX-SAT problems are solved by either incomplete local search methods or complete branch-and-bound based exhaustive search. For most MAX-SAT problems, local search methods exhibit better speed and scaling than complete methods. Local search methods, unfortunately, do not provide a *proof* that the returned solution is optimal.

The success of modern complete SAT solvers has inspired a number of researchers to develop improved complete MAX-SAT algorithms. Recent developments include using unit propagation for strong bound computation [Li *et al.*, 2005; 2006]; adapting local consistency methods developed for CSP to MAX-SAT [de Givry *et al.*, 2003]; and using fast pseudo-Boolean solvers to check the optimality of MAX-SAT bounds [Aloul *et al.*, 2002].

In probabilistic reasoning, the problem of Most Probable Explanation (MPE) is to find an assignment to all variables that has the maximum likelihood given the evidence. Exact methods for MPE on probability distributions represented by Bayesian networks include well-known methods such as the Join Tree algorithm [Jensen *et al.*, 1990], as well as a recent branch-and-bound algorithm, AND/OR Tree Search [Marinescu and Dechter, 2005]. Since solving MPE exactly is NP-Hard, local search algorithms have been introduced for approximation [Park, 2002]. In this paper, we consider MPE on CNF formulas with weighted literals, where the goal is to find a solution with maximum product or sum of its literal weights. This CNF-based MPE problem is strictly more general than MPE for discrete Bayesian networks, because any discrete Bayesian network can be converted to a CNF with weighted literals whose size is linear in the size of conditional probability tables (CPTs) of the network [Sang *et al.*, 2005b].

MPE on CNF can be viewed as a special case of Weighted Model Counting (WMC) [Sang *et al.*, 2005b], and is likely easier than WMC because we may apply branch-and-bound based pruning techniques to reduce the search space. We choose Cachet [Sang *et al.*, 2004], a state-of-the-art model counting system, as a platform on which to build our MPE solver, and we extend pruning techniques for subproblems with components. As a result, we present MPE-SAT, a decomposition-based branch-and-bound algorithm that works on top of WMC and prunes the search space effectively. Furthermore, we are able to use the dtree algorithm of [Huang and Darwiche, 2003; Darwiche, 2002] to boost the performance on problems with good decomposability.

In general, MPE and weighted MAX-SAT illustrate two complementary ways of representing soft constraints: either having weight on variables or having weight on constraints (clauses). Although they have apparently different representations of weight and goals, they can be converted to each other by adding auxiliary variables or clauses, possibly at

some loss in efficiency. In addition to describing these reductions, we show how to formulate CNF-based MPE as an iterative pseudo-Boolean satisfiability (PBSAT) process.

In our experiments we compare our MPE solver MPE-SAT with other state-of-the-art complete MPE/MAX-SAT solvers as well as a pseudo-Boolean solver on a variety of MPE and MAX-SAT problems. Our approach is quite competitive on most problems and is significantly faster than each of the other solvers on at least one of the classes of benchmarks.

2 MPE and Weighted MAX-SAT

The MPE problem originated in research on Bayesian networks, one of the most popular graphical models. A Bayesian network is a DAG, where each source node has a prior probability distribution on its values and each non-source node has a Conditional Probability Table (CPT) specifying the probability distribution of its values given the values of its parents. Most Probable Explanation (MPE) is the problem of finding a complete assignment of values to nodes that has the maximum likelihood given some node values as evidence. The likelihood of a complete assignment is the product of the corresponding entries in the conditional probability tables and the prior probabilities. To solve MPE exactly, one can either compile the Bayesian network into a junction tree and then propagate the evidence [Jensen *et al.*, 1990], or perform a branch-and-bound search, *e.g.*, AND/OR tree search [Marinescu and Dechter, 2005]. Alternatively, one can convert MPE to weighted MAX-SAT and solve it by any local search algorithm [Park, 2002].

We begin with some definitions:

Definition 1 A *CNF formula with weighted literals* is a CNF formula plus a function $weight$ that maps literals to real-valued non-negative weights.

Definition 2 Given a combination operator $\oplus$ defined on the reals, the problem of *MPE on CNF formulas with weighted literals* is to find a complete satisfying assignment s that has the maximum $\oplus_i weight(v_i)$, where v_i is either the positive or negative form of the ith variable in s. To be convenient, we also define the inverse operator $\ominus$. MPE of an unsatisfiable CNF formula is defined to be 0.

This CNF-based MPE can represent Bayesian-network-based MPE because there are linear reductions from Bayesian networks to CNF formulas with weighted literals [Sang *et al.*, 2005b; Chavira and Darwiche, 2005]. Practically, setting $\oplus$ to arithmetic $+$ or $\times$ (then $\ominus$ is either $-$ or $/$) suffices our purposes . For example, $\times$ is used for likelihood-originated MPE and $+$ is used for weighted MAX-SAT-originated MPE. In the rest of the paper, we will use the short term MPE for "MPE on CNF formulas with weighted literals", when there is no confusion.

The problem of weighted MAX-SAT on a CNF formula is to find an assignment that has the maximum sum of weights of all clauses satisfied by that assignment. When every weight is 1, weighted MAX-SAT reduces to MAX-SAT.

Both MPE and weighted MAX-SAT optimize a metric of the weight, the only difference is that MPE has a weight on each literal and weighted MAX-SAT has a weight on each clause. Not surprisingly these two representations are equivalent in that they can be directly converted to each other.

MPE to weighted MAX-SAT

We give a simple conversion for MPE on CNF to weighted MAX-SAT, which is different from the encoding in [Park, 2002] translates MPE on Bayesian Networks to weighted MAX-SAT. Assuming the combination operator is $+$,

- for each literal in MPE, a unit clause is added with weight equal to the weight of the literal.
- all original clauses in MPE are assigned an "infinite" weight, which is a number chosen to be at least as large as the sum of the weights added in the previous step.

Since all original clauses have an infinite weight, any optimal solution in weighted MAX-SAT must satisfy them and maximize the sum of weights of satisfied unit clauses, which obviously maximizes the sum of literal weights in MPE and therefore is an optimal solution in MPE as well, and vice versa. The converted formula has a mixture of hard and soft constraints, which is a challenge for exact weighted MAX-SAT solvers: to be efficient, they must take advantage of the hard constraints.

Weighted MAX-SAT to MPE

Givry *et. al.* [2003] described a Pseudo-Boolean encoding for MAX-SAT. Although that does not explicitly refer to MPE, it can be modified for MPE by adding proper weights. Our conversion is as follows:

- for every variable x in weighted MAX-SAT, let $weight(x) = weight(\neg x) = 0$.
- for every clause c_i in weighted MAX-SAT, an auxiliary literal $\neg y_i$ is added to c_i, with $weight(\neg y_i) = 0$ and $weight(y_i) = weight(c_i)$.

While the original formula for MAX-SAT may be unsatisfiable, the converted formula is guaranteed to be satisfiable because one can always obtain a trivial solution by setting all auxiliary y variables to false. When the combination operator is fixed to sum, solving MPE on the converted formula finds a solution with a maximum sum of weight of y variables, which is equivalent to maximizing the sum of weights of satisfied clauses in weighted MAX-SAT. This encoding does not add any clause, but needs as many auxiliary variables as the number of clauses. The inflated number of variables makes solving MPE more difficult, especially since auxiliary variables make every clause trivially satisfiable and thus unit propagation and conflict-driven learning become unavailable.

MPE and Pseudo-Boolean SAT

Since the encoding in [de Givry *et al.*, 2003] converts MAX-SAT to Pseudo-Boolean SAT and (weighted) MAX-SAT and MPE are equivalent, we observe that there is a close relationship between MPE as Pseudo-Boolean SAT as well.

Definition 3 Given a CNF formula ϕ and a set C of linear constraints over variables in ϕ, *Pseudo-Boolean SAT (PB-SAT)* is to find a total assignment that satisfies both ϕ and linear constraints in C. A linear constraint has the form $\sum_{i=1}^{n} a_i v_i \geq d$ where v_i is a variable that takes value 0 or

1, a_i and d are real-valued constants and n is the number of variables.

We show how to convert a likelihood-based MPE to an iterative PBSAT process. In the context of likelihood, the combination operator $\oplus$ becomes $\times$, and a variable v_i has $weight(v_i) = p_i$ and $weight(\neg v_i) = 1 - p_i$. Without loss of generality, we assume $0 < p_i < 1$, otherwise if $p_i = 0$ (1) we solve the simplified formula with v_i instantiated to 0 (1). The MPE goal is to find a complete satisfying assignment that maximizes

$$\prod_{x_i=1} p_i \prod_{x_i=0} (1 - p_i) \quad = \quad \prod_i (\frac{p_i}{1 - p_i})^{x_i} \prod_i (1 - p_i).$$

Since $\prod_i (1 - p_i)$ is a constant and log is an increasing function, this is equivalent to maximizing

$$\log [\prod_i (\frac{p_i}{1 - p_i})^{x_i}] = \sum_i \log (\frac{p_i}{1 - p_i}) x_i.$$

This linear expression can be used as the left side of a linear constraint in PBSAT, but we still need to figure out what bound d to put on the right side. By definition of MPE, we know that the optimal value must be in $[\, 0, \prod_{i=1}^{n} Max(p_i, (1 - p_i))\,]$. Now we can iteratively search for the bound d using binary search. At each step when d is set, we solve PBSAT with a single linear constraint

$$\sum_i \log (\frac{p_i}{1 - p_i}) x_i \;\geq\; d.$$

In this way, we can get d arbitrarily close to the optimal value in a bounded number of steps.

3 Algorithms and the Implementation

In this section, we first examine a simple MPE algorithm and then show how to enhance it with some advanced techniques.

3.1 DPLL for MPE

The naïve **Algorithm 1** is a simple modification of the classic DPLL search. First if the formula is empty (already satisfied), DPLL-MPE returns the optimal value (*e.g.*, sum or product) of weights of unassigned variables, which becomes part of the current value; if the formula is UNSAT, DPLL-MPE returns 0 by definition; otherwise it selects a variable to branch, recursively solves each branch and then returns the best value, which is the better one found in the two branches.

Unlike DPLL for SAT where the search halts when a SAT leaf is found, DPLL-MPE performs an exhaustive search over all possible SAT leaves for the optimal value. It can be very slow without proper pruning, and that is why branch-and-bound algorithms are widely used for many similar optimization tasks including MAX-SAT and weighted CSP.

Algorithm 1 DPLL-MPE

DPLL-MPE(ϕ) // returns MPE of CNF formula ϕ
  if ϕ is empty, return optimal value of unassigned variables
  if ϕ has an empty clause, return 0
  select an unassigned variable $v \in \phi$ to branch
  return Max(DPLL-MPE($\phi|_{v=0}$) $\oplus$ $weight(\neg v)$,
        DPLL-MPE($\phi|_{v=1}$) $\oplus$ $weight(v)$)

Algorithm 2 MPE-SAT

MPE-SAT(ϕ, lb) // returns MPE of CNF formula ϕ
  if ϕ is empty, return optimal value of unassigned vars
  if ϕ has an empty clause, do **nogood learning** and return 0
  do **dynamic component detection**: solve each separately
  do **cache lookup**: reuse previously computed values
  do **dynamic bounding**: if $E(\phi) \leq lb$ return 0 // pruning
  select an unassigned literal $v \in \phi$ by **branching heuristics**
  lresult = MPE-SAT($\phi|_{v=0}, lb \ominus weight(\neg v)$)
  update lb according to lresult
  rresult = MPE-SAT($\phi|_{v=1}, lb \ominus weight(v)$)
  result = Max($lresult \oplus weight(\neg v), rresult \oplus weight(v)$)
  do **caching**: AddToCache(ϕ, result)
  return result

3.2 Branch-and-Bound and Decomposition

Branch-and-bound is the underlying mechanism for most exhaustive search algorithms that find an optimal solution. The branch-and-bound algorithm maintains a global best solution found so far, as a lower bound. If the estimated upper bound of a node is not better than the lower bound, the node is pruned and the search continues with other branches.

Previous research has shown that decomposition and caching techniques are critical for such exhaustive search to be efficient [Darwiche, 2002; Bacchus *et al.*, 2003; Sang *et al.*, 2004; Marinescu and Dechter, 2005]. However, with dynamic decomposition the simple form of branch-and-bound must be modified. During the search, a problem may decompose into several independent sub-problems (components) that, for efficiency, one would like to analyze separately. The branch-and-bound algorithm must include a way of allocating portions of the global bounds to the individual components.

3.3 MPE-SAT

To address the above issues, we develop MPE-SAT, a new decomposition-based branch-and-bound algorithm with dynamic decomposition and caching. MPE-SAT extends DPLL-MPE with the following function blocks. Since there are a number of similarities with the related AND/OR tree search algorithm [Marinescu and Dechter, 2005], we briefly compare our methods with that algorithm.

Dynamic Component Detection A connected component detection is performed dynamically for every non-trivial ϕ in the search. Since components of ϕ are independent sub-problems with disjoint variables, they are solved separately and then their results are combined to get the result for ϕ (though **Algorithm 2** does not show details of these obvious steps due to space limitation). (In AND/OR tree search, components are determined statically using the pseudo tree constructed before the search.)

Caching and Cache lookup When the value of a component is known, the component with its value is stored in a hash table (caching) for later reuse (cache lookup), which avoids repeated computation. A component to store can be its corresponding clauses or a short signature of the clauses to save space. Even when the exact value of a component is unknown because of pruning in its subtree, a proper bound can be cached for reuse too. The purpose of this bound caching

is to save the partial work already done for a component that is to be pruned. A cached bound can be used as an estimated upper bound of the component for pruning and it is updated whenever a tighter bound or the exact value of the same component is found. (AND/OR tree search caches the context instead of the component itself.)

Dynamic Bounding The parameter lb, initially ∞ or given by local search, is the lower bound for ϕ when ϕ is created. $E(\phi)$ is an upper bound estimation of the true value of ϕ, which can be simply $\oplus_{v\in\phi} Max(weight(v), weight(\neg v))$, a cached bound or a result by special computation (as described in section 3.4). If $E(\phi)$ is at most lb, the current branch is pruned for it will not yield a better result. Note that for a subproblem, lb for its right branch may be better than lb for its left branch, because solving the left branch may improve the previous lb. When a sub-problem (component) is being solved, only local information is needed for pruning; i.e., the bound from its parent and the bounds from sibling components, which are updated dynamically. A parent's lb is immediately broken and passed into its sub-problems for pruning (top-down). For example, if sub-problem S splits into S_1 and S_2 then $lb_{S_1} = lb_S \ominus E(S_2)$. and $lb_{S_2} = lb_S \ominus E(S_1)$. However, since S_2 is solved after S_1, lb_{S_2} should be dynamically updated with the exact value of S_1 replacing $E(S_1)$. (AND/OR tree search uses dynamic bounding as well but collects bounds from sub-problems (bottom-up).)

Branching Heuristics Any dynamic heuristic good for DPLL search will work, and it turns out that decomposition-based heuristics are often very useful (as discussed in section 3.4). (AND/OR tree search uses the statically-constructed pseudo-tree heuristic, which also aims at decomposition.)

Nogood Learning The well-known conflict-driven clause learning technique for satisfiability testing, *e.g.*, [Zhang *et al.*, 2001], can be directly used for CNF-based MPE. Learned clauses implicitly prune the infeasible search space.

Comparison Since the top-down scheme passes the best known lower bound to a sub-problem once it is available, MPE-SAT may examine fewer nodes than AND/OR tree does when pruning occurs. MPE-SAT benefits from nogood learning that dynamically changes its variable ordering (with VSIDS or VSADS heuristic), while nogood learning is likely less useful for the static pseudo-tree variable ordering. The dynamic component detection in MPE-SAT is more powerful than the static detection in AND/OR tree search because the latter may miss some decomposition due to dynamic effects; however, the overhead of the former is much higher. Finally, the more expensive component caching in MPE-SAT is more powerful than context caching in AND/OR tree search, because different contexts may lead to the same component.

3.4 The Implementation

The problem of model counting (counting the number of solutions of a CNF formula) shares many features with MPE, both requiring exhaustive search. In DPLL-MPE if we replace the max operation by sum (and the optimal value of a SAT leaf is one) we will get a naïve model counting algorithm. Using the weighted model counting system Cachet [Sang *et al.*, 2004] as a platform, we have implemented the MPE-SAT algorithm and the following to better support it.

Component Processing Components are processed in a depth-first order and the search stays within a component until it is finished. This component processing strategy is well suited to dynamic bounding for sibling components, but it is different from that for model counting. In Cachet, once a component is found SAT during search, the work on the rest of that component will be temporarily suspended and an unexploited component (if available) will be processed next. That strategy works well for finding UNSAT components as soon as possible, which is good for model counting; but in MPE-SAT we want to continue working on a SAT component until it is fully done, because the known value of a finished component yields better lower bounds for its siblings when they are checked for pruning.

Optimal Solution Retrieval In order to get the optimal solution as well as the optimal value, we need to maintain all partial solutions associated with all the active components on the current search path, from which we can compose the optimal solution at backtracking. The optimal partial solution of a component must be cached together with its exact value.

Branching Heuristics The dynamic branching heuristics for SAT and model counting such as VSIDS, VSADS and EUPC [Moskewicz *et al.*, 2001; Sang *et al.*, 2005a] work well for MPE too. These heuristics aim at maximizing the effect of unit propagation or avoiding the infeasible search space by learning from conflicts. However for MAX-SAT problems, we have found that the dtree-based semi-dynamic branching heuristic [Huang and Darwiche, 2003; Darwiche, 2002; 2004] is often better, because in those problems unit propagations and conflicts that guide dynamic heuristics are no longer available whereas structural decomposition based heuristics are affected less. The dtree program by [Huang and Darwiche, 2003] computes a static variable group ordering based on dtree decomposition [Darwiche, 2002], which is fed to MPE-SAT. The semi-dynamic branching heuristic makes choices by dynamic heuristic for variables in the same group, but prefers variables in a group with higher priority and the group priorities are determined statically by dtree. The time for running dtree is usually insignificant compared to the time for solving the problem. A nice property of dtree is that after each group of variables is instantiated (from high to low priority), the problem is guaranteed to decompose. However, if a problem has a large tree-width, the sizes of the high priority groups by dtree can be so large that the search space blows up before the problem decomposes.

Upper Bound Computation We are able to extend the UP heuristic [Li *et al.*, 2005] to weighted MAX-SAT in our solver. Computing an upper bound on the weight of the satisfied clauses of formula ϕ is equivalent to computing a lower bound on the weight of violated clauses of ϕ. The weight of violated clauses of ϕ can be estimated as follows:

$CostLB = 0$
for each unit clause of ϕ
  Simplify ϕ by setting the current unit clause to true
  if there is an empty clause (conflict)
    increment $CostLB$ by the minimum weight of the empty clause and the clauses for deriving it, and
    remove clauses for deriving the empty clause from ϕ
$CostLB$ is the lower bound of violated clauses at the end.

We also adopt a trick from previous MAX-SAT solvers that when the value of a known solution and the current estimated value differ only by one, unit clauses are safely propagated.

4 Experimental Results

We tested with both CNF-based MPE and MAX-SAT. The CNF-based MPE problems are either from random 3-CNF or structured CNF formulas with random literal weights (equivalent to weighted MAX-SAT), or translated from Bayesian networks (special because of the ad-hoc Bayesian-network-to-CNF encoding). The MAX-SAT problems are structured ones used in previous literature and fault diagnosis problems generated from standard circuit benchmarks that are challenging for many solvers. More details of these problems can be found at www.cs.washington.edu/homes/sang/benchmarks.htm

We used the following state-of-the-art solvers:

Toolbar [de Givry *et al.*, 2003; 2005] : an exact weighted CSP solver that integrates efficient algorithms for maintaining various levels of local consistency. It is a generic solver for weighted CSP and weighted MAX-SAT. We used version 3.0.

UP [Li *et al.*, 2005] : an exact branch-and-bound MAX-SAT solver using unit propagation and distinguishing independent conflicts for powerful lower bound computation.

MaxSatz [Li *et al.*, 2006] : an exact branch-and-bound MAX-SAT solver extending **UP** with sophisticated lower bound computation techniques. Highly optimized for MAX-SAT, it was the winner of the 2006 MAX-SAT evaluation.

PB2 [Aloul *et al.*, 2002] : a pseudo Boolean SAT solver that takes linear constraints. It can prove the optimality of encoded MAX-SAT problems. Basically, an auxiliary variable is added for each clause and there is a linear constraint limiting the sum of these added variables to the given bound.

AoTree [Marinescu and Dechter, 2005]: the AND/OR tree implementation that works for MPE of Bayesian networks and weighted CSP.

MPE-SAT: our solver that implements the MPE-SAT algorithm. It is for CNF-based MPE and weighted MAX-SAT.

All experiments were done on a Linux machine with 2.8 GHz Pentium 4 CPU and 4 GB memory, except that we ran AoTree on a Windows machine with 2.8 GHz Pentium 4 CPU and 1.5 GB memory. The runtime cutoff is 600 seconds. Since not all solvers apply to the same domains, for each domain, we show the results only for the solvers that apply.

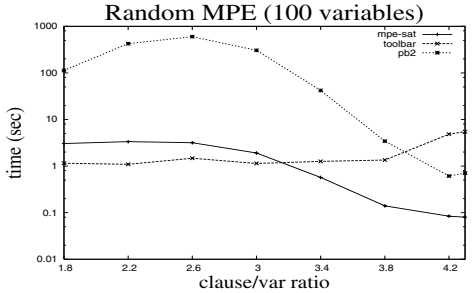

Figure 1: MPE of random 3-CNF (median runtime)

Problems	#vars	#clauses	Toolbar	PB2	MPE-SAT
ra	1236	11416	30	0.04	0.18
rb	1854	11324	X	1.2	0.72
rc	2472	17942	X	10	3.9
2bitcomp6	252	766	17	5.5	0.21
2bitmax6	150	370	X	3.5	0.28
rand1	304	578	X	X	15

Figure 2: MPE of circuit CNF and runtime in seconds.

Problems	#Node	Induced width	AoTree	MPE-SAT
75-12	144	19	20	0.47
90-15	225	23	412	0.1
75-15	225	25	X	0.56
90-25	625	41	X	5.5
90-30	900	47	X	68

Figure 3: MPE of grid Bayesian networks and runtime in seconds.

The first domain is MPE on satisfiable random 3-CNF formulas with random weights between 1 and 1000 on each positive literal and 0 weight on each negative literal. The problems were converted to weighted MAX-SAT via the encoding in section 2 (introducing only positive unit clauses). Since we use $+$ as the combination operator $\oplus$ for MPE here, a direct translation to PBSAT is adding a linear constraint where the weighted sum of all positive literals is at least a given bound. In principle we should run PB2 iteratively to optimize this bound, but we just set the bound as opt (for proving SAT) and $opt + 1$ (for proving UNSAT), because PB2 is not efficient enough on this domain. (The value of opt was that found by other solvers.) For MPE-SAT, we used the weight-based dynamic heuristic, where variable selections with good unit-propagated weights are preferred before a solution is found (to get a good bound early) and bad unit-propagated weights are preferred afterwords (to get prunings early).

In Figure 1, each point represents the median runtime on 100 instances. On the PB2 curve, the data point of ratio 2.6 is actually a timeout (median > 600 seconds). Compared to the other two, PB2 is very inefficient at low ratios, frequently timed out. However, as the problem gets more and more constrained with ratio increasing towards 4.2, PB2 improves significantly. MPE-SAT has a similar trend: it is about two times slower than Toolbar at low ratios, but gains a dramatic speedup at high ratios. Apparently at high ratios MPE-SAT and PB2 benefit a lot from clause learning by the underlying SAT engine, which prunes most infeasible search space. The curve of Toolbar is rather flat before ratio 4.2. For low ratio under-constrained problems, Toolbar prunes the search space very effectively using advanced bounding techniques; however, without nogood learning, it is not effective at pruning using the constraints and so does poorly on well-constrained high ratio problems. This effect is amplified with 200 variables and ratio 4.2: Toolbar often cannot solve an instance in an hour but MPE-SAT can solve it in a few seconds.

Figure 2 shows the results of MPE on structured circuit CNF formulas with random variable weights. Clearly most problems are easy to MPE-SAT but hard or non-solvable to Toolbar. We also ran PB2 with a naïve binary search for the optimal value. PB2 works fine on all problems but one. It appears that CNF-based MPE suits MPE-SAT and PB2 better than Toolbar, which is optimized for weighted CSP.

In Figure 3, we compared with AND/OR tree search on

Problems	Toolbar	PB2	MaxSatz	UP	MPE-SAT
Pret60_60	40	0.001	11	49	0.05
Pret60_75	41	0.001	11	48	0.06
Pret150_60	X	0.003	X	X	0.09
Pret150_75	X	0.003	X	X	0.1
dubois22	130	0.001	37	248	0.04
dubois23	245	0.001	74	168	0.04
dubois24	496	0.001	152	X	0.04
dubois25	X	0.001	311	X	0.04
dubois26	X	0.001	X	X	0.04
dubois30	X	0.001	X	X	0.05
dubois100	X	0.006	X	X	0.17
aim-100-1_6-n1	318	0.001	7.8	41	81
aim-100-1_6-n2	2	0.005	2.3	24	139
aim-100-1_6-n3	X	0.003	13.8	194	55
aim-100-1_6-n4	X	0.005	10.0	29	78
aim-100-2_0-n1	158	0.003	23	24	2.2
aim-100-2_0-n2	20	0.008	13	24	1.5
aim-100-2_0-n3	145	0.001	5.9	20	16
aim-100-2_0-n4	192	0.04	15	23	4
hole07	0.19	0.001	0.04	0.03	0.31
hole08	1.8	0.002	0.4	0.4	1.9
hole09	20	0.002	4.1	5.8	8.9
hole10	249	0.002	46	96	35

Figure 4: MAX-SAT problems from DIMACS UNSAT instances and runtime in seconds (X = time > 600).

grid MPE problems on Bayesian networks. An $n \times n$ grid network has binary-valued nodes indexed by pairs (i, j), with node $(1, 1)$ as a source and (n, n) as a sink. Each node (i, j) has parents indexed by $(i-1, j)$ and $(i, j-1)$, and a fraction of nodes have deterministic CPTs [Sang *et al.*, 2005b]. The sink is set to true as the evidence for MPE. The induced width of a Bayesian network is a measure of its density. It is clear that MPE-SAT dominates on all these highly deterministic problems by a large margin and AoTree has difficulties on grid networks with large induced width. MPE-SAT is not very sensitive to the induced width because the underlying SAT engine can explore the local structures (introduced by deterministic entries) efficiently.

The remaining experiments were on structured MAX-SAT problems: examples considered in [de Givry *et al.*, 2003; Li *et al.*, 2005] and fault diagnosis problems. Our MPE solver is competitive with the other solvers on most classes. (We do not show results for unstructured random MAX-SAT problems where MPE-SAT is not at all competitive because MPE-SAT spends significant time on dynamic decomposition checking and caching, which is wasted because the problems have very dense constraints and hardly decompose.) For each problem, we gave all solvers the initial bound found by Borchers' local search program for MAX-SAT [Borchers and Furman, 1999], and the optimal value is defined as the minimum number of clauses violated. In fact, local search can quickly find the optimal bound of 1 for every problem in Figure 4. All the solvers will find a solution value at least as good as the given bound.

In Figure 4, PB2 dominates on all problems, and MPE-SAT is also quite good, outperforming Toolbar and UP on most problems. Unlike other branch-and-bound solvers, PB2 proves the optimality of a given bound that becomes a single linear constraint. When the upper bound is 1, the problem is probably under-constrained (note that the encoding for PB2 is trivially satisfiable without the linear constraint), so PB2 can easily find a solution. It is also easy for PB2 to prove that all problems in Figure 4 are UNSAT using SAT techniques

Problems.	OPT	Bound	Toolbar	PB2	MaxSatz	UP	MPE-SAT
c432-1	1	4	16.9	0.02	0.25	0.17	0.7
c432-2	1	8	X	0.01	543	2.3	0.1
c432-3	2	6	X	0.16	37	0.2	0.25
c432-4	2	6	X	0.06	4.1	0.4	0.75
c499-1	3	4	55	4.58	0.22	0.01	0.4
c499-2	4	5	28	3.58	3.33	0.01	0.37
c499-3	7	9	X	X	2.96	0.03	0.66
c499-4	8	9	X	X	7.94	0.02	0.75
c880-1	4	9	11.6	0.3	23	0.3	2.2
c880-2	5	6	X	X	X	0.1	2.7
c880-3	6	8	X	X	X	0.2	7.8
c880-4	7	8	X	X	X	0.11	5.9
c1355-1	5	9	X	X	X	1	1.5
c1355-2	6	9	X	X	X	1	3.4
c1355-3	7	12	X	X	X	1.7	4
c1355-4	8	13	X	X	X	1.9	6.7

Figure 5: MAX-SAT problems from fault diagnosis and runtime in seconds (X = time > 600).

such as clause learning and non-chronological backtracking, which are unavailable for other MAX-SAT solvers including MPE-SAT (because of the MAX-SAT-to-MPE encoding).

MPE-SAT works very well on pret and dubois problems, because they decompose quickly—they usually have small separator sets by dtree based heuristics. After those critical variables are instantiated, the problem decomposes into independent sub-problems and MPE-SAT can solve them efficiently. For the same reason, MPE-SAT works less well on some aim-100 and hole problems—the sizes of root separator sets vary from 17 to 20, large enough to be hard.

Toolbar and UP have somewhat similar behaviors over problems in Figure 4, in that they can solve all aim (except one) and hole problems and all fail on large pret and dubois problems, though UP is faster up to a factor. For large pret and dubois problems they seem to have difficulty in finding a good solution matching the given upper bound, but for hole problems finding a good solution is easy: removing any clause makes the formula satisfiable.

Next, we show that PB2 is indeed not universally good for all MAX-SAT problems. In Figure 5, the fault diagnosis problems were generated from ISCAS-85 combinational circuits [Brglez and Fujiwara, 1985]. In a circuit we make some gates have stuck-at faults such that the output is inconsistent with the given input, so a MPE solution is a sound explanation with the least number of faulty gates. This problem can be solved by a SAT based method that enumerates possible explanations [Smith *et al.*, 2005]. An optimal solution often falsifies quite a few clauses. Each entry in the /Bound column is the initial upper bound (for solvers) returned by local search. It is interesting that local search cannot find any optimal value. PB2 at least takes one run to prove UNSAT (given $bound = opt - 1$) and another run to prove SAT (given $bound = opt$). So each number for PB2 is the sum of these two runtimes, and we ignore the insignificant portion of time for other runs.

Only UP and MPE-SAT can successfully solve all fault diagnosis problems, and UP is faster by a constant factor. UP is extremely efficient probably because of its lower bound computation: there are many binary clauses in these formulas and UP may get a very tight lower bound by applying the unit literal rule iteratively. It is a little surprising that MaxSatz, which is supposed to use more sophisticated lower bounding

techniques than UP, has serious difficulties on many problems. MPE-SAT is efficient because it takes advantage of decomposition: the problems often decompose after a few instantiations when dtree heuristic is used. Toolbar fails on most problems, while PB2 can solve problems with small optimal values (≤ 4) but fails on most problems with large optimal values (≥ 5). We guess that in general pseudo-Boolean solvers become very inefficient for MAX-SAT when the optimal value is reasonably large, where it is hard to prove both SAT and UNSAT.

5 Conclusion

MPE and weighted MAX-SAT are complementary representations for problems with soft constraints and we have described natural reductions between them. To solve these optimization tasks efficiently, we have presented an algorithm MPE-SAT which incorporates various techniques with a dynamic flavor: including dynamic problem decomposition, dynamic bounding, caching (a form of dynamic programming), and clause-learning. As a result, our approach is quite competitive with other solvers on a wide range of problem domains from MPE and MAX-SAT and significantly outperforms each on at least one of the domains.

References

[Aloul *et al.*, 2002] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack search pseudo-boolean solver. In *Proc. 5th International Conference on Theory and Applications of Satisfiability Testing*, 2002.

[Bacchus *et al.*, 2003] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Uncertainty in Artificial Intelligence UAI-2003*, pages 20–28, 2003.

[Bistarelli *et al.*, 1997] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[Borchers and Furman, 1999] B. Borchers and J. Furman. A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT. *Journal of Combinatorial Optimization*, 2(4):299-306, 1999.

[Brglez and Fujiwara, 1985] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark designs and a special translator in fortran. In *Proc. ISCAS*, 1985.

[Chavira and Darwiche, 2005] M. Chavira and A. Darwiche. Compiling Bayesian networks with local structures. In *Proc. 19th IJCAI*, 2005.

[Darwiche, 2002] A. Darwiche. A compiler for deterministic decomposable negation normal form. In *Proc. 18th AAAI*, 2002.

[Darwiche, 2004] A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. 16th European Conference on Artificial Intelligence*, 2004.

[de Givry *et al.*, 2003] S. de Givry, J. Larrosa, P. Meseguer, and T. Schiex. Solving max-sat as weighted csp. In *9th Principles and Practice of Constraint Programming*, 2003.

[de Givry *et al.*, 2005] S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *Proc. 19th IJCAI*, 2005.

[Huang and Darwiche, 2003] J. Huang and A. Darwiche. A structure-based variable ordering heuristic for sat. In *Proc. 18th IJCAI*, 2003.

[Jensen *et al.*, 1990] F. V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[Li *et al.*, 2005] C. M. Li, F. Manya, and J. Planes. Exploiting unit propagation to compute lower bounds in branch and bound max-sat solvers. In *Proc. 11th Principles and Practice of Constraint Programming*, 2005.

[Li *et al.*, 2006] C. Li, F. Manya, and J. Planes. Detecting Disjoint Inconsistent Subformulas for Computing Lower Bounds for Max-SAT. In *Proc. 21st AAAI*, 2006.

[Marinescu and Dechter, 2005] R. Marinescu and R. Dechter. And/or branch-and-bound for graphical models. In *Proc. 19th IJCAI*, 2005.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. 38th Design Automation Conference*, 2001.

[Park, 2002] J. Park. Using weighted MAX-SAT engines to solve MPE. In *Proc. 18th AAAI*, 2002.

[Sang *et al.*, 2004] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. 7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[Sang *et al.*, 2005a] T. Sang, P. Beame, and H. Kautz. Heuristics for fast exact model counting. In *Proc. 8th International Conference on Theory and Applications of Satisfiability Testing*, 2005.

[Sang *et al.*, 2005b] T. Sang, P. Beame, and H. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. 20th AAAI*, 2005.

[Smith *et al.*, 2005] A. Smith, A. Veneris, M. Ali, and A. Viglas. Fault diagnosis and logic debugging using boolean satisfiability. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 24, 2005.

[Zhang *et al.*, 2001] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *International Conference on Computer Aided Design*, 2001.