Computing Equivalent Transformations for Combinatorial Optimization by Branch-and-Bound Search

Eric I. Hsu and **Sheila A. McIlraith**
University of Toronto
{eihsu,sheila}@cs.toronto.edu

Abstract

Branch-and-Bound search is a basic algorithm for solving combinatorial optimization problems. Here we introduce a new lower-bounding methodology that can be incorporated into any branch-and-bound solver, and demonstraint its use on the MaxSAT constraint optimization problem. The approach is to adapt a minimum-height equivalent transformation framework that was first developed in the context of computer vision. We present efficient algorithms to realize this framework within the MaxSAT domain, and demonstrate their feasibility by implementing them within the state-of-the-art MAXSATZ solver. We evaluate the solver on test sets from the 2009 MaxSAT competition; we observe a basic performance tradeoff whereby the (quadratic) time cost of computing the transformations may or may not be worthwhile in exchange for better bounds and more frequent pruning. For specific test sets, the trade-off does result in significant improvement in both prunings and overall run-time.

1. Introduction

MaxSAT is an optimization problem whose theoretical and practical importance has motivated a growing body of research on exact solvers, *e.g.* (Li, Manyà, and Planes 2007; Heras, Larrosa, and Oliveras 2008; Lin, Su, and Li 2008; Ansótegui, Bonet, and Levy 2009). Solutions are variable assignments that maximize the weight of the clauses that they satisfy–or equivalently, by convention the goal is to *minimize* the weight of unsatisfied clauses. As a discrete optimization problem, MaxSAT is amenable to branch-and-bound search; for this approach to work it is critical to compute tight lower bounds on the weight of clauses that must go unsatisfied upon completing a partial assignment, in order to prune the search space below said assignment whenever the lower bound exceeds an upper bound representing the best solution found so far. Typically, such lower bounds have been produced by applying resolution-like inference rules whenever fixing a variable during search (Li, Manyà, and Planes 2007).

Here we introduce a new framework for constructing MaxSAT lower bounds that can be incorporated into any branch-and-bound solution method. The conceptual basis for this framework is "minimum-height equivalent transformation" ("MHET"), which derives lower bounds by optimistically assuming that we can achieve full problem height, *i.e.*, that we can achieve the maximum score for each clause. Making this bound non-trivial first requires an extension to the language of MaxSAT problems, where clauses can now give varying weights to different configurations of their variables. Then, given a particular basic problem, we can seek a problem in the extended space of problems that is equivalent in how it scores any variable assignment, but has minimal height. The concept of MHET originates from a formal analysis of vision problems that was originally developed in the Soviet Union (in the absence of high-powered computers!), and that was recently reviewed in the context of relating probabilistic reasoning to constraint satisfaction (Schlesinger 1976; Werner 2007).

The primary contribution of this paper is to adapt the MHET framework to MaxSAT, introducing representations and algorithms that make the framework tractable for contemporary problems in clausal normal form. The resulting adaptation can be seen as a generalization of existing inference procedures that aspires to produce tighter bounds. We have also implemented the bounding technique within the state-of-the-art MAXSATZ solver, and assessed its usefulness on nineteen test sets representing a broad variety of MaxSAT challenge areas and applications. In some of these sets, performing equivalent transformations yields an overall improvement in both the number of prunings and the overall runtime. In the remainder, the methodology still increases the number of prunings, but its computational overhead results in longer overall runtimes.

Section 2 motivates and defines the notion of minimum-height equivalent transformation for MaxSAT, while Section 3 presents efficient algorithms for calculating problem height and finding MHET's to local optimality. Section 4 describes the implementation of MHET and presents empirical results. Finally, Section 5 relates the framework to existing research and makes concluding observations.

2. Approach

To describe our approach to computing lower bounds by MHET, we first define the MaxSAT problem conventionally, and then using a graphical interpretation, with an extensional representation of clauses. We can then extend the space of problems, and specify how a problem from this extended space can be equivalent to a given conventional problem. We then present an "equivalent transformation" process that encodes a mapping from a conventional problem to an equivalent extended one. Finally, we define the notion of problem height, which serves as our minimization criterion within the space of equivalent problems, and is the basis for MHET lower bounds.

MaxSAT. A weighted **MaxSAT** problem (in CNF) is a triple $\Phi = (X, F, W)$, where:

- X is a set of n Boolean **variables**. By convention we refer to 0 as the "negative" value and 1 as the "positive." We will typically index members of X with the subscript i, as in "$x_i \in X$." An **assignment** is a set of mappings between variables in X and values in $\{0, 1\}$; an individual mapping is denoted by, for example, "$x_i = 0$."
- F is a set of m **clauses** defined over X. Each clause is a disjunction of positive or negative literals formed from variables in X. We will typically index members of F with the subscript a. Clause f_a is **satisfied** by a given variable assignment A *iff* it contains a literal whose variable is mapped to the corresponding polarity in A.
- $W \in (\mathbb{R}^+ \cup \{0\})^m$ associates a non-negative **weight** with each clause in F. We will denote by "w_a" the weight associated with clause f_a. (The approach to be described can also easily accommodate *partial* weighted MaxSAT problems, which distinguishes between hard and soft weights–this distinction is left out of the description for ease of presentation.)

The **score** of an assignment to all the variables in a MaxSAT problem is the sum of the weights associated with the clauses that it satisfies. A solution to a MaxSAT problem is a complete assignment that achieves the maximum possible score–or rather, minimizes the weight of unsatisfied clauses.

Factor Graph Interpretation. A MaxSAT problem can be represented as a **factor graph**, or a bipartite graph containing two types of nodes: *variables* and *functions*. Associated with each such node is a variable or a function over variables, respectively; usually we will not need to distinguish between variables/functions and their associated nodes in the graph. Edges in the graph connect variable nodes to nodes for functions in which they appear as arguments. To interpret a MaxSAT problem $\Phi = (X, F, W)$ as a factor graph, let X correspond to the variables in the graph, and F correspond to the functions. Each clause f_a should be viewed as a two-valued function that evaluates to 0 if it is not satisfied by the way its variables are assigned, and w_a otherwise. We can call the set of variables appearing as literals in a clause f_a the "**scope**" of f_a, denoted σ_a. Similarly, we call the set of clauses containing literals for a variable x_i its "**neighborhood**," denoted η_i.

Extensional Representation. Under the **extensional**, or dual, representation, we explicitly list the value of a function f_a for each of the $2^{|\sigma_a|}$ possible assignments to the variables in its scope. Such assignments can be called "**extensions**" of f_a. Further, we can **condition** f_a on an individual variable assignment x_i = *val* by discarding those extensions that do not contain the assignment. In so doing, we thereby define a new function with scope $\sigma_a \setminus \{x_i\}$.

Example 1. *Figure 1(a) shows a simple two-variable MaxSAT problem in clausal form. Figure 1(b) depicts the same problem as a factor graph with the clauses represented as functions in extensional form. Conditioning the function f_c on the variable assignment $x_1 = 1$, for example, yields a new function defined over x_2 whose value is 0 if x_2 is assigned negatively, and 18 otherwise.*

Equivalent Problems. Under the extensional representation, we can now extend our definition of MaxSAT problems from using clauses that can be represented exclusively by two-valued functions, to arbitrary functions that can give distinct positive or negative values to each possible extension of their arguments. That is, each function f_a can now map $\{0, 1\}^{\sigma_a}$ to the reals as opposed to $\{0, w_a\}$. Where before the score of a given assignment was the sum of weights for clauses that it satisfied, now the **score** is the sum of function values according to the assignment. We can call such problems "**extended MaxSAT**" problems, of which traditional MaxSAT problems are a subclass. Two extended MaxSAT problems $\Phi = (X, F)$ and $\Phi' = (X, F')$ are **equivalent** *iff* they yield the same score for all possible assignments to all the variables in X.

Equivalent Transformations. Here our interest is in extended MaxSAT problems that are equivalent to traditional ones that we wish to solve. Given a traditional MaxSAT problem $\Phi = (X, F, W)$, we can encode a specific equivalent extended problem $\Phi' = (X, F')$ by means of an **equivalent transformation** $\Psi \in \mathbb{R}^{|X|\,|F|\,|\{+,-\}|}$. Ψ is a vector of individual positive and negative **potentials**, which are defined between variables and functions in their neighborhoods, and are denoted $\psi^+_{i,a}$ and $\psi^-_{i,a}$ respectively. Each such pair of potential acts to "**rescale**" a function's values by instructing f_a to subtract $\psi^+_{i,a}$ from the value of each extension wherein x_i is assigned positively, and to subtract $\psi^-_{i,a}$ from those where it is assigned negatively. Thus, if Φ' is the result of applying equivalent transformation Ψ to Φ, then for each $f_a \in F$ the rescaled $f'_a \in F'$ is defined as follows:

$$f'_a(ext) = f_a(ext) - \sum_{(x_i=val)\in ext} \begin{cases} \psi^-_{i,a} & \text{if } val \text{ is } 0; \\ \psi^+_{i,a} & \text{if } val \text{ us } 1. \end{cases} \quad (1)$$

Recall that because f_a represents a clause that evaluates to either 0 or w_a, the expression above can be recast as the negative sum of all potentials corresponding to the given extension–plus w_a if the extension is any of the $2^{|\sigma_a|} - 1$ that satisfy the original clause. (This perspective will be crucial to the algorithms in Section 3.where we will evade the computational expense of explicit extensional functions representations.)

Subtracting a series of arbitrary real numbers from the values of particular function extensions yields a transformation, but to be sure this is not necessarily an equivalent

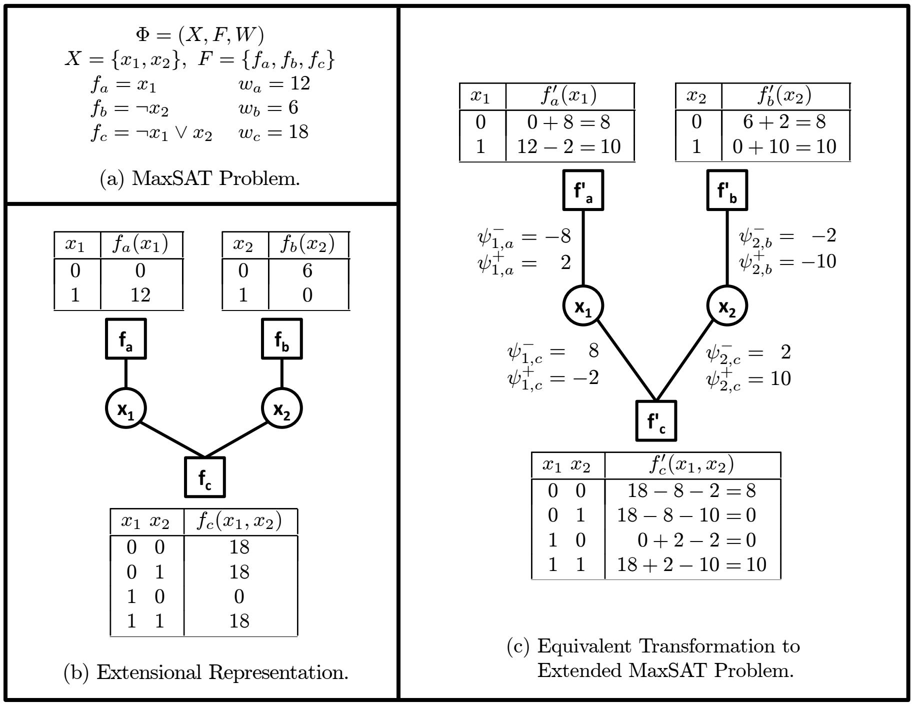

Figure 1: Example MaxSAT problem, represented (a) conventionally as clauses and (b) extensionally as a factor graph. Equivalent extended MaxSAT problem (c) under equivalent transformation Ψ, whose every pair of components $\psi^-_{i,a}$ and $\psi^+_{i,a}$ appears beside the edge connecting variable x_i to function f_a. (The equivalent transformation also introduces two new unary functions over x_1 and x_2, but these both assign a score of 0 to any extension, and are therefore not depicted.)

one. To achieve exactly the same score for Φ' as for Φ under any assignment, we augment F' with a unary function f'_i for each variable $x_i \in X$. These serve to add back in all the positive or negative rescalings associated with the variable, according to how it is assigned:

$$f'_i(\{x_i = val\}) = \begin{cases} \sum_{a \in \eta_i} \psi^-_{i,a} & \text{if } val \text{ is } 0; \\ \sum_{a \in \eta_i} \psi^+_{i,a} & \text{if } val \text{ us } 1. \end{cases} \quad (2)$$

(If F already contains a unary, or "unit" clause for variable x_i it doesn't affect the formalism if we create two functions with the same scope in F', or perform some sort of special handling when implementing the algorithms in Section 3)

In summary, an equivalent transformation Ψ encodes two potentials for every variable/function pair, and thus comprises a rescaling of all the functions (clauses) in a standard MaxSAT problem. Applying Ψ to standard problem $\Phi = (X, F, W)$ yields the equivalent extended problem $\Phi' = (X, F')$ where $F' = \{f'_a : f_a \in F\} \cup \{f'_i : x_i \in X\}$, with each function f'_a and f'_i defined as in (1) and (2). It is simple to see that the original score under Φ equals the new score under Φ' for any assignment to the variables in X: if x_i is assigned positively, for example, then f'_a will subtract $\psi^+_{i,a}$ from the original score for each clause f_a in which x_i appears; f_i will in turn add back each such $\psi^+_{i,a}$.

Example 2. *In Figure 1(c) the example problem undergoes equivalent transformation Ψ, yielding an equivalent extended MaxSAT problem. The individual potentials comprising Ψ appear beside the corresponding edges in the factor graph. Intuitively, the variable x_i has directed function f'_c to decrease its value by 8 if x_i is assigned negatively; this is offset by also instructing f'_a to* increase *its value by 8 if x_i is 0. By such means, (b) and (c) yield the same score for any assignment to x_1 and x_2.*

Height. The **height of a function** f_a is its maximum value across extensions. The **height of an extended MaxSAT problem** Φ is the sum of the heights of its functions. A **minimum-height equivalent transformation** for a given MaxSAT problem is one that produces an equivalent problem whose height is no greater than that of any other equivalent problem. The height of a problem is an upper bound on the maximum score that can be achieved by any assignment to its variables; so the motivation for finding minimum-height equivalent transformations is to tighten this bound.

Example 3. *The height of the regular MaxSAT problem in Figure 1(b) is $12 + 6 + 18 = 36$; here we optimistically assume that we can accrue the maximum score for each clause, though in reality this cannot be done consistently by a single assignment to all the variables. In contrast, the height of the extended problem (c) is $10 + 10 + 10 = 30$. In fact, 30 is the minimum height across all problems equivalent to (b), so Ψ is a minimum-height equivalent transformation. (In fact, the height of (c) is a* tight *upper bound on the maximum score achievable for (b): the score of 30 corresponding to*

$\{x_1 = 1, x_2 = 1\}$ *is maximal across assignments.)*

In summary, we use the height of a problem to bound the maximum achievable score for a problem from above. Given a particular standard MaxSAT problem, then, we can consider the space of equivalent extended problems, and aspire to find one yielding the tightest possible upper bound, by minimizing height. The difference between the sum of all clause weights in the original problem and the height of the chosen equivalent problem thus comprises a *lower* bound on the amount of weight that will have to be left unsatisfied by any assignment to the problem's variables. This constitutes a new lower-bounding technique that can be applied directly to any branch-and-bound backtracking MaxSAT solver.

3. Algorithms

Our overall goal of providing height-based lower bounds to branch-and-bound MaxSAT solvers requires efficient algorithms for calculating function heights, and for pursuing minimum height equivalent transformations. To calculate heights in extended problems, we sidestep the inherently exponential cost of representing functions extensionally. For finding MHET's we apply an approximate algorithm that circumvents the inherent NP-completeness of the task (if optimal height-minimization could be done in polynomial time, then so could SAT, because any satisfiable problem has a minimum-height equivalent problem whose height is the number of clauses.)

Algorithm 1: LOWER-BOUND-BY-MIN-HEIGHT

Input : Weighted MaxSAT problem $\Phi = (X, F, W)$.
Output: Lower bound on weight of unsatisfied clauses.

1 $\Psi \leftarrow$ MIN-HEIGHT-EQUIV-TRANSFORM (Φ).
2 *lb* $\leftarrow 0$.
3 **foreach** $x_i \in X$ **do**
4 | *lb* $\leftarrow$ *lb* $-$ VARIABLE-HEIGHT (x_i, Ψ).
5 **end**
6 **foreach** $f_a \in F$ **do**
7 | *lb* $\leftarrow$ *lb* $+ w_a -$ CLAUSE-HEIGHT (f_a, w_a, Ψ).
8 **end**
9 **return** *lb*.

Algorithm 1 depicts the overall process of producing lower bounds. Given the subproblem induced by the series of variable assignments that we have made up to a certain point during branch-and-bound search, we first find a height-minimizing equivalent transformation Ψ. We then calculate the difference between the total weight available from the problem, and its height under the equivalent transformation, producing a lower bound on the weight of clauses that will have to be unsatisfied should we continue down this branch of search. Aside from the step of actually pursuing a minimum-height equivalent transformation, the complexity of Algorithm 1 and its subroutines is $\mathcal{O}(d(n+m))$, where d is the largest degree of any node in the problem's factor graph representation.

The "variable-height" function at Line 4 of the algorithm indicates the height of the unary function that the equivalent transformation process introduces for each variable, as in Eq. (2). Algorithm 2 performs this fairly straightforward calculation. (This depiction and the following are written for clarity rather than efficiency–many of the calculations can be cached and re-used within and between procedures.)

Algorithm 2: VARIABLE-HEIGHT

Input : Variable x_i, potentials comprising equivalent transformation Ψ.
Output: Height of x_i under equivalent transformation.

// *Variable's positive/negative rescaled weight is sum of*
// *the pos/neg potentials it distributes across its clauses.*
neg $\leftarrow \sum_{a \in \eta_i} \psi^-_{i,a}$, *pos* $\leftarrow \sum_{a \in \eta_i} \psi^+_{i,a}$.
// *Height is maximum rescaled weight.*
return max (*pos*, *neg*).

To calculate the height of a transformed problem clause at Line 7, we cannot rely on any explicit representation of extension values akin to the tables in Figure 1(c). Instead, we rely on the fixed structure of equivalent transformations by variable-clause potentials, and observe that with one exception, the highest-scoring extension for a clause is always that with the least associated potentials. The one exception occurs when the extension with least rescaling happens to be the unique extension that does not satisfy the original clause–then we do not accrue the original clause weight. In this case, we must determine whether gaining this weight is worth the extra rescaling that our score will suffer should we flip the assignment of a single variable. This is the basis for calculating a transformed clause's height under a given equivalent transformation, in Algorithm 3.

Line 15 of Algorithm 3 determines the decrease in score that we suffer on substituting the second-least rescaled extension for the least-rescaled one identified earlier in the algorithm. This is done by flipping a single variable with least difference between its two potentials (recall that all extensions satisfy an original clause, except one.) This allows us to judge whether sacrificing the clause's weight or suffering additional rescaling will yield the function's highest score.

The remaining algorithmic task is to find equivalent transformations that minimize problem height. In Algorithm 4 we adapt the *max-sum diffusion* algorithm (Kovalevsky and Koval approx 1975) developed from an older line of research in the Soviet Union (Schlesinger 1976) and recently reviewed in the context of probabilistic inference and computer vision (Werner 2007). The algorithm is guaranteed to converge, but only to a local minimum in problem height, rather than a global minimum. Lines 6 though 14 comprise the core of the process, effecting a "height-smoothing" that rescales the factors surrounding a given variable so that they all have the same height when conditioned on either assignment to the variable. For a given positive or negative assignment, this is realized by calculating the height of each function when conditioned on the assignment (by subjecting the clause height algorithm to the simple changes depicted as Algorithm 5.) and forming an average. We then update the variable's potentials for this polarity to the difference between each clause's conditional height and this average. The time complexity for each iteration of the algorithm is $\mathcal{O}(d(n+m))$.

Algorithm 3: CLAUSE-HEIGHT

Input : Factor f_a, weight w_a, potentials comprising equivalent transformation Ψ.
Output: Height of f_a under equivalent transformation.

// Get minimum-potential assignment to variables in f_a.
1 *assignment* $\leftarrow \{\}$.
2 *potentials* $\leftarrow 0$.
3 **foreach** $x_i \in \sigma_a$ **do**
4   **if** $\psi^-_{i,a} < \psi^+_{i,a}$ **then**
5   *assignment* $\leftarrow$ *assignment* $\cup \{x_i = 0\}$.
6   *potentials* $\leftarrow$ *potentials* $+ \psi^-_{i,a}$.
7   **else**
8   *assignment* $\leftarrow$ *assignment* $\cup \{x_i = 1\}$.
9   *potentials* $\leftarrow$ *potentials* $+ \psi^+_{i,a}$.
10   **end**
11 **end**
12 **if** *assignment* **satisfies** f_a **then**
  // Height is factor's weight after minimum rescaling.
13   **return** $w_a -$ *potentials*.
14 **else**
  // Height depends on whether factor's weight is
  // more valuable than difference between minimum
  // and next-most-minimum rescaling.
15   *difference* $\leftarrow \infty$.
16   **foreach** $x_i \in \sigma_a$ **do**
17   **if** $|\psi^-_{i,a} - \psi^+_{i,a}| <$ *difference* **then**
      // Flipping this variable yields least increase
      // in minimum rescaling (and satisfies f_a).
18   *difference* $\leftarrow |\psi^-_{i,a} - \psi^+_{i,a}|$.
19   **end**
20   **end**
21   **if** $w_a >$ *difference* **then**
22   **return** $w_a -$ *potentials* $-$ *difference*.
23   **else**
24   **return** $-$*potentials*.
25   **end**
26 **end**

Convergence to a local minimum in problem height is readily demonstrated by directly adapting a theorem from the review of the originating MHET research (Werner 2007). (Technically the theorem allows the algorithm to stagnate at the same non-minimal height, so long as the height never increases, but this has never been observed in practice. Also, the number of iterations is unbounded, but convergence can be enforced by an epsilon-valued threshold.)

Algorithm 4: MIN-HEIGHT-EQUIV-TRANSFORM

Input : Weighted MaxSAT problem $\Phi = (X, F, W)$.
Output: Height-minimizing equivalent transformation $\Psi \in \mathbb{R}^{|X|\,|F|\,|\{+,-\}|}$.

// Initialize all variable-factor potentials to zero.
1 **foreach** $x_i \in X,\ f_a \in F$ **do**
2   $\psi^+_{i,a} \leftarrow 0, \quad \psi^-_{i,a} \leftarrow 0.$
3 **end**
4 **repeat**
5   **foreach** $x_i \in X$ **do**
    // Find max rescaled score for each of variable's
    // functions, conditioned on each polarity.
6   **foreach** $f_a \in \eta_i$ **do**
7       $\alpha^+_{i,a} \leftarrow$ COND-HEIGHT $(f_a, w_a, x_i = 1, \Psi)$.
8       $\alpha^-_{i,a} \leftarrow$ COND-HEIGHT $(f_a, w_a, x_i = 0, \Psi)$.
9   **end**
    // Average max rescaled scores across functions.
10     $\mu^+ \leftarrow \sum_{a \in \eta_i} \alpha^+_{i,a} / |\eta_i|$.
11     $\mu^- \leftarrow \sum_{a \in \eta_i} \alpha^-_{i,a} / |\eta_i|$.
    // Set the variable's potentials to rescale each
    // function's former max score to the average.
12   **foreach** $f_a \in \eta_i$ **do**
13       $\psi^+_{i,a} \leftarrow \alpha^+_{i,a} - \mu^+$.
14       $\psi^-_{i,a} \leftarrow \alpha^-_{i,a} - \mu^-$.
15   **end**
16   **end**
17 **until** *convergence*
18 **return** Ψ.

Algorithm 5: COND-HEIGHT

// Algorithm is same as CLAUSE-HEIGHT, but changed
// to reflect assumption that x_i is already fixed to *val*.
1 **Change** Line 1 **to** "*assignment* $\leftarrow \{x_i = \mathit{val}\}$."
2 **Change** Line 3 **to** "**foreach** $x_i \in \sigma_a \setminus \{x_i\}$ **do**."
3 **Change** Line 16 **to** "**foreach** $x_i \in \sigma_a \setminus \{x_i\}$ **do**."

Theorem 1 ("Convergence"). *After each iteration of Line 5 in Algorithm 4, the height of Φ cannot increase.*

Proof. At the beginning of the loop, variable x_i and its neighborhood η_i contribute quantity $\max\{\sum_{a\in\eta_i}\psi^-_{i,a}, \sum_{a\in\eta_i}\psi^+_{i,a}\} + \sum_{a\in\eta_i}\max\{\alpha^+_{i,a} - \psi^+_{i,a}, \alpha^-_{i,a} - \psi^-_{i,a}\}$ (the single-variable height of x_i plus the height of each clause) to the height of Φ. At the end of the loop, they contribute $\max\{\sum_{a\in\eta_i}\psi^+_{i,a} + \alpha^+_{i,a} - \psi^+_{i,a}, \sum_{a\in\eta_i}\psi^-_{i,a} + \alpha^-_{i,a} - \psi^-_{i,a}\}$. The second quantity cannot exceed the first. $\square$

Intuitively, the proof observes that in first calculating problem height, we are free to cast x_i positively in choosing the highest-scoring extension for one of its functions, and to still cast it negatively when taking the height of another. After iterating on x_i, we are still free to do so, but all of x_i's functions will yield the same height when it is set positively, and they will all share another common height when

x_i is set negatively–we only consider which of these two is greater. As for correctness of the lower bound, it is easiest to appeal directly to the *equivalence* property of equivalent transformations: no matter what values Algorithm 4 assigns to its components, Ψ will by construction define a problem that yields the same score as Φ, for any variable assignment.

4. Implementation and Results

We have implemented the algorithms described above within the state-of-the-art 2009 version of the MAXSATZ solver(Li, Manyà, and Planes 2007)[1]. We thus evaluate the usefulness of MHET as a lower-bounding framework on nineteen test sets comprising the most general divisions (weighted, and weighted partial MaxSAT) of the 2009 MaxSAT Evaluation (Argelich et al. 2009). The experiments were run on machines with 2.66 GHz CPU's using 512 MB of memory.

Table 1 compares the performance of MAXSATZ with and without the computation of minimum-height equivalent transformations. The basic tradeoff is between the increased computational cost of computing the transformations, versus the opportunity to search a reduced space due the extra prunings triggered by such transformations.

4.1 Problems for which MHET is Beneficial, Neutral, or Harmful Overall

The rows of the table group the nineteen problem sets into three categories, by comparing the number of problems solved using the two versions of MAXSATZ within a 30-minute timeout. The categories can be understood as containing problems sets for which adding MHET improves overall performance, makes no significant difference, or degrades overall performance. The principal statistics appear in the first two groupings of three columns each. The first column in each such grouping displays the number of problems from a given test set that were solved before timeout. The next columns show, for problems that were solved within the timeout period, the average number of backtracks and overall average runtimes. (Thus, a version may show a lower average runtime even if it solved fewer problems.)

For the first six problem sets in the table, MAXSATZ with MHET embedded solves a greater or equal number of problems without timing out when compared to regular MAXSATZ. Looking ahead to the third grouping of columns, entitled "Prunings Triggered," we can characterize these problems as ones where MHET allows for a significant number of prunings, while the original version of MAXSATZ could find few or none. Here the difference is enough to outweigh the added computational cost of performing MHET. The problems in this category are diverse in their specific characteristics; one general property that they have is common relative to the other entries in the table is that they are fairly difficult to solve, meaning that fewer problems within a set can be solved in the time allotted, and that these problems themselves require more time. This yields a simple explanation for why the greater power of MHET is worth the computational expense. A secondary phenomenon that remains to be explained is that these problems also tend to have large weights; at this point it is unclear to the authors whether this is actually beneficial to the MHET formalism, and why this might be.

For the middle grouping of problem sets, adding MHET slows MAXSATZ overall for all but two test sets; but the difference in runtime is negligible in the sense that the two versions still solve the same number of problems within the 30-minute cutoff period. This grouping is characterized by problems that are already extremely easy or extremely hard for both versions. Still, it is worth noting that for all but the (extremely easy) random problems, MHET continues the trend of finding additional prunings and drastically reducing the search space, while regular MAXSATZ performs far more branches but without the overhead that tends to double overall runtime. So, while the two versions of the solver are comparable here, particularly in terms of the number of problems solved before the cutoff time, such parity is achieved by divergent means.

For the final grouping of three problem sets, adding MHET prevents MAXSATZ from solving problems that it was originally able to solve, within the 30-minute timeout. While MHET is still able to find a significant number of new prunings, for these problem its overhead cost is prohibitive. Such problems are characterized by the largest numbers of variables across the entire evaluation (thousands, in the case of the Mancoosi configuration problems.) Thus, the greater cost of doing MHET on such problems makes it not worthwhile. Still, it is important to note that proportionally to the existing heuristics, MHET is not especially more costly, as such heuristics also tend to have low-order polynomial complexity on the number of variables. Rather, the problems are just large in general and present a challenge to any inference-based heuristic.

4.2 Direct Measurement of Pruning Capability, and Proportion of Runtime

Turning to the third ("Prunings Triggered") grouping of columns, we can corroborate some overall trends across all categories of problems. Here we have re-run each test suite with a new configuration of MAXSATZ that runs both "MHET" and the "original" MAXSATZ inference rules (Rules 1 and 2 from (Li, Manyà, and Planes 2007)) each time a variable is fixed, regardless of whether one or the other has already triggered a pruning. This way, we can identify pruning opportunities that could be identified exclusively by one technique or the other, or by both. Additionally, if neither technique triggers a pruning, we identify a third class of "other" prunings–those that are triggered by additional mechanisms built into the solver like unit propagation, the pure literal rule, and most prominently, look-ahead. Here we see that although MHET always triggers more prunings than the original MAXSATZ inference rules, the test sets for which it causes longer runtimes correspond almost exactly to those where an even greater number of prunings could have been triggered by the third class of built-in mechanisms.

[1] Upon publication of its description, the source code for the solver, along with example problems, will be available at `http://www.cs.toronto.edu/~eihsu/MAXVARSAT/`.

	Solved w/ MHET			Solved w/o MHET			Prunings Triggered				Runtime %	
Test Suite	*#*	*branches*	*time*	*#*	*branches*	*time*	*MHET*	*orig.*	*both*	*other*	*MHET*	*rest*
auction-paths	***87***	***201K***	***74***	76	14087K	192	57	0	0	0	82	4
random-frb	***34***	***1K***	***187***	9	789K	12	1K	0	0	0	82	5
ramsey	***39***	***40K***	***99***	37	28K	4	4K	2	87	11K	12	1
warehouse-place	***6***	***208K***	***520***	1	6K	0	636	0	0	0	42	6
satellite-dir	***3***	***632K***	***105***	2	593K	7	33K	0	0	0	75	5
satellite-log	***3***	***1012K***	***475***	2	578K	9	95K	0	0	89	59	4
factor	186	***38***	***0***	186	32K	13	3	0	0	17	67	26
part. rand. 2-SAT	90	***511***	15	90	***511***	***8***	143	0	0	333	57	40
part. rand. 3-SAT	60	***35K***	72	60	***35K***	***20***	473	0	4	33K	57	8
rand. 2-SAT	80	***236***	***1***	80	***236***	***1***	5	0	0	229	66	29
rand. 3-SAT	80	***61K***	171	80	62K	***56***	645	0	1	52K	76	13
maxcut-d	57	***15K***	89	57	37145	***82***	5K	0	4	13K	70	23
planning	50	***9K***	147	50	97K	***128***	17K	149	162	27K	76	13
quasigroup	13	***110K***	156	13	564K	***93***	4K	1K	87	185	87	7
maxcut-s	4	***3K***	35	4	4K	***22***	5	0	2	3K	41	33
miplib	3	***24K***	325	3	30K	***192***	25	0	0	10	92	8
mancoosi-config	5	1107K	1445	***40***	***5424K***	***1271***	4K	23	3	95	78	13
bayes-mpe	10	5K	2	***21***	***787K***	***152***	299K	0	0	6K	82	14
auction-schedule	82	549K	164	***84***	***1504K***	***38***	33	0	0	0	83	7

Table 1: **Performance of MHET on weighted problems from MaxSAT Evaluation 2009.**
The first two groups of columns depict the performance of MAXSATZ with and without MHET-derived lower bounds. Each entry lists the number of problems from a given test set that were solved within a 30-minute cutoff period, and the average number of backtracks and seconds of CPU time required for each *successful* run. (All three entries appear in bold when a configuration solves more problems than the other; otherwise backtracks and CPU time are highlighted to indicate the more efficient configuration.) The third column tracks the number of prunings that the various lower bounding mechanisms are able to generate, as described in the text. The final column depicts the average percentage of total runtime devoted to performing the MHET lower-bounding technique, versus the rest ("orig." and "other").

This observation is amplified by the final column of the results table, which shows the percentage of overall runtime used to calculate MHET versus that of all other pruning mechanisms (both "original" and "other"). Here we see that while MHET computations are extremely valuable, they are also very expensive in terms of time. For the four test suites where adding MHET increases overall runtime, the "other" built-in pruning mechanisms are able to find a large number of prunings, demotivating the use of MHET. Still, the low-order polynomial complexity of performing MHET suggests that its percentage cost will always be within a constant factor of the other mechanisms'.

4.3 Practical Considerations: Blending the Two Pruning Methods

The experiments demonstrate that MHET is able to generate unique lower bounds and trigger prunings that were not possible using the existing methods. For at least some of the problems, this ability is strictly beneficial; and for the majority it does not do any harm in terms of the number of problems solved within the timeout period. For practical considerations, though, it is compelling to seek a finer-grained approach to using MHET versus the existing rules. For in the experiments above, recall that the new and existing lower bounding techniques are run at every single node of the search space. Instead, it may be worthwhile to attempt to prune at only a portion of the nodes in the search space, and to choose between MHET and the old techniques in some way that balances computational cost with greater pruning power.

Thus, an area of ongoing research has been to formulate a blend of using either no lower-bounding techniques, old techniques alone, MHET alone, or both, at selected nodes in the search space. On the one hand, we are generally more likely to prune toward the bottom of the search tree, so running a bounding technique here is more likely to pay off with a pruning. On the other hand, if we do manage to trigger a pruning higher in the search tree, then the pruned subtree will be exponentially larger. In this light, a simple heuristic is to run the old bounding techniques at every node, and run MHET once every 10^a node expansions, where a is a parameter. Such a scheme shows an exponential preference for running the method lower in the search tree, because there are exponentially more such nodes. Naturally, using low values of a will be effective on problems from Table 1 where MHET was more successful, while increasingly high values of a will simulate using the old techniques alone, producing better results on the corresponding problems. In practice, we have found that this method produces a reasonable improvement on the test sets considered, with a set to about 3.

Also under development is a more sophisticated approach to blending four choices of pruning strategy (*nothing*, *old*, *MHET*, *old+MHET*), while learning from experience and consulting the current state of the search process. Here we use reinforcement learning to balance between exploring the effectiveness of the various actions in different situations within search, while also exploiting our experiences to prefer those actions that have done well when applied to comparable situations in the past. We measure "doing well" by penalizing decisions for the runtimes they incurr, while rewarding them according to how many nodes they are able to prune. And "situations" are realized by state variables that represent our current depth within the search tree and the current gap between the upper bound and the most recently computed lower bound.

In this paper, though, we have isolated the effect of MHET by running it at every node, and demonstrated its usefulness in triggering unique prunings and minimizing backtracks, while in some cases improving overall runtime. Methods for blending the various pruning methods are described in a separate presentation.

5. Related Work and Conclusions

We have demonstrated a Minimum-Height Equivalent Transformation framework for computing a new type of lower bound that can improve the performance of branch-and-bound search for MaxSAT. Interestingly, the notion of equivalent transformation defined over an expanded space of extensionally represented functions can be seen as a "soft" generalization of existing inference rules within the current state-of-the-art. More precisely, just as a linear programming relaxation drives the formal apparatus in the original account of MHET (Schlesinger 1976), so can we view the algorithms of Section 3as sound relaxations of traditional inference rules like those of MAXSATZ, whose proofs of soundness incidentally appeal to an integer programming formulation (Li, Manyà, and Planes 2007). For instance, the problem in Figure 1 is simple enough to have been solved one such rule (essentially, weighted resolution or arc-consistency.) But by softening the rules to work over fractional weights and functions with more than two values, and by allowing the consequences of a rescaling operation to propagate throughout the factor graph over multiple iterations, we achieve a finer granularity and more complete process to reduce problem height on real problems (a goal that is not motivated by existing approaches that seek exclusively to create empty clauses.)

Similarly, within the research area of weighted constraint satisfaction problems (a generalization of MaxSAT), a growing body of work has also sought to process weighted constraint problems into equivalent ones that produce tighter bounds (Cooper et al. 2008; Cooper, DeGivry, and Schiex 2007; Cooper and Schiex 2004). Indeed, many of the representations and optimizations found in the algorithms presented here can independently be found in generalized form within this literature. In keeping with the previous discussion of related MaxSAT techniques, such efforts originated from the translation of hard (consistency-based) inference procedures from regular constraint problems to weighted ones. Again the work presented here differs primarily in using fractional weights and multiple rescaling iterations, thus allowing weight to be re-distributed further along chains of interacting variables, and at a finer granularity.

In conclusion, the MaxSAT solver presented here reflects an MHET apparatus that is decades-old (Schlesinger 1976), but that requires a good deal of adaptation and algorithmic design to achieve the gains depicted in Table 1. Two untapped features of that foundation suggest directions of future work. First, the authors link the height-minimization framework to the entire field of research in performing statistical inference on probablistic models (like Markov Random Fields.) This suggests algorithmic alternatives to the "diffusion" method presented here, and an opportunity to improve the generated bounds or (more importantly) reduce their computational cost. Secondly, the tightness of a height-minimization bound can be tested (in binary domains like SAT) by arc consistency alone. In this work, though, we have adapted an existing formulation motivated by computer vision to introduce a new bounding technique to the MaxSAT research area. We have developed efficient algorithms to make the framework feasible for problems in conjunctive normal form, and tested its utility on a variety of contemporary problems.

References

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT '09*.

Argelich, J.; Li, C. M.; Manyà, F.; and Planes, J. 2009. Fourth Max-SAT evaluation. `http://www.maxsat.udl.cat/09/`.

Cooper, M. C., and Schiex, T. 2004. Arc consistency for soft constraints. *Artificial Intelligence* 154(1-2):199–227.

Cooper, M. C.; DeGivry, S.; Sanchez, M.; Schiex, T.; and Zytnicki, M. 2008. Virtual arc consistency for weighted CSP. In *Proc. of 23rd National Conference on A.I. (AAAI '08), Chicago IL*, 253–258.

Cooper, M.; DeGivry, S.; and Schiex, T. 2007. Optimal soft arc consistency. In *Proc. of 20th International Joint Conference on A.I. (IJCAI '07), Hyderabad, India*, 68–73.

Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient weighted Max-SAT solver. *JAIR* 31.

Kovalevsky, V. A., and Koval, V. K. (approx.) 1975. A diffusion algorithm for decreasing energy of max-sum labeling problem. *Glushkov Institute of Cybernetics, Kiev, U.S.S.R.*

Li, C. M.; Manyà, F.; and Planes, J. 2007. New inference rules for Max-SAT. *JAIR* 30.

Lin, H.; Su, K.; and Li, C. M. 2008. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *AAAI '08*.

Schlesinger, M. I. 1976. Sintaksicheskiy analiz dvumernykh zritelnikh signalov v usloviyakh pomekh (syntactic analysis of two-dimensional visual signals in noisy conditions). *Kibernetika* 4.

Werner, T. 2007. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29(7).