The Promise of Polynomial-based Local Search to Boost Boolean MAX-CSP Solvers

Christine D. Hang Ahmed Abdelmeged Daniel Rinehart Karl J. Lieberherr

Northeastern University,
College of Computer and Information Science,
360 Avenue of the Arts, Boston, MA 02115, USA
{christine,mohsen,danielr,lieber}@ccs.neu.edu

Abstract. We propose a novel family of polynomial-based local search algorithms for MAX-CSP. From this family, we present an optimal, fast algorithm, called Evergreen local search (*ELS*). We evaluate ELS as a preprocessor for state-of-the-art MAX-CSP and MAX-SAT solvers which shows promising improvement in speed and quality.

1 Introduction

Is backtracking search necessary to boost the performance of MAX-CSP solvers? To answer this question with a no, we introduce a novel polynomial-based local search algorithm and use it as a preprocessor to boost the performance of Boolean MAX-CSP solvers.

We define the following terminologies: A Boolean CSP formula is a non-empty bag of constraints, each of which consists of an integer multiplicity and a Boolean relation of some rank r. The multiplicity indicates how often the constraint appears in the bag, and the Boolean relation of rank r represents a Boolean formula involving r variables. Henceforth, we refer to a Boolean CSP formula as a CSP formula. An assignment for a CSP formula F maps the variables of F to Boolean values. $fsat(F, J)$ is the fraction of satisfied constraints in formula F under assignment J, where a satisfied constraint is one whose corresponding Boolean formula evaluates to true under J.

We further define a constraint language Γ as a set of relations, and a $CSP(\Gamma)$ formula as a CSP formula that only contains relations in Γ. Then, a $MAX\text{-}CSP(\Gamma)$ problem can be formalized as one that has as input a $CSP(\Gamma)$ formula F and as output an assignment J such that $fsat(F, J)$ is maximized. Boolean MAX-CSP solvers are used to solve $MAX\text{-}CSP(\Gamma)$ problems.

We reactivate the golden ratio technique used in solving MAX-CSP problems from [1][2] in the 1980s. The golden ratio technique is about approximating MAX-CSP problems in a "P-optimal" way. The approximation is formalized as an **infimum-maximum** problem (defined in section 3) in terms of a fraction of all constraints. The golden ratio technique solves the infimum-maximum problem based on a reduction called symmetrization which reduces the problem to a much simpler one involving

polynomials. The polynomials lead to P-optimal algorithms [1]. An algorithm for solving $MAX\text{-}CSP(\Gamma)$ problems is said to be P-optimal with respect to Γ, if:

1. for any $MAX\text{-}CSP(\Gamma)$ formula, the algorithm is guaranteed to satisfy a fraction of τ_Γ of its constraints
2. the problem of solving the set of $MAX\text{-}CSP(\Gamma)$ formulae in which the fraction $\tau_\Gamma + \epsilon$ ($\epsilon > 0$) can be satisfied is NP-complete.

We call τ_Γ the P-optimal threshold with respect to Γ. It can be efficiently computed using polynomials.

This paper follows figure 1. It derives τ_Γ from $MAX\text{-}CSP(\Gamma)$ problems and develops efficient algorithms to achieve it. The process of deriving and achieving τ_Γ reveals a novel local search algorithm, which we call Evergreen Local Search (ELS). This algorithm can be used as a preprocessor to boost the performance of MAC-CSP solvers.

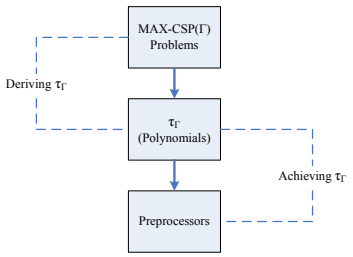

Fig. 1. Presentation Scheme

1.1 Contributions

This paper builds on and partially redevelops the golden ratio technique. It makes the following contributions:

- We show how the algorithms behind the golden ratio technique can be reinterpreted from a local search perspective.
- We impose new requirements on MAX-CSP solvers: they should achieve the P-optimal threshold.
- We show encouraging experimental results in which the golden ratio technique improved the performance of some state-of-the-art MAX-SAT and MAX-CSP solvers.

1.2 Paper Organization

The rest of the paper is organized as follows: Section 2 introduces our local search algorithm ELS. Section 3 derives τ_Γ and shows how to compute it using polynomials. Section 4 achieves τ_Γ and completes ELS. Section 5 postulates two laws to which future MAX-CSP solvers should conform, and introduces ELS' role of enforcing them. Section 6 illustrates the boosting effect of ELS on MAX-CSP and MAX-SAT solvers. Section 7 discusses related and future work, and 8 concludes.

2 Evergreen Local Search

In local search, a neighborhood relation is used to define the set of assignments that are considered neighbors of a given assignment. We introduce the notion of flipping a variable as setting it to the negation of its current value. An assignment J_2 is a k-flip of assignment J_1, if k variables in J_2 are flipped with respect to J_1. For a given formula F and a given assignment J_1, we consider an assignment J_2 to be in the Evergreen-neighborhood(EN) of J_1 if J_2 has a satisfaction ratio that is no less than the average of the satisfaction ratios of all k-flips of J_1. It is important to recognize that J_2 does not have to be a k-flip of J_1.
For a given assignment J, traditional k-*opt* local search algorithms consider all the k-flips of J to be its neighbors. What distinguishes our local search algorithm from the traditional algorithms is that we only pick the assignments that are at least as good as the average of the k-flips of J to be its neighbors. Specifically, let $lap(F, J, k)$ be the mean fraction of satisfied constraints over the $\binom{n}{k}$ assignments for F where among the n variables of J exactly k of them are flipped. We will introduce how to compute $lap(F, J, k)$ in section 3.

Definition 1. *For formula F, J_2 is a k-Evergreen-neighbor of J_1, written as $EN(F, J_1, J_2, k)$, if $fsat(F, J_2) \geq lap(F, J_1, k)$.*

We define k-Evergreen-neighbor finder, $ENF(F, J_1, k)$, as an algorithm that takes as inputs a formula F, an assignment J_1 and an integer $k(0 \leq k \leq n)$, and returns as output an assignment J_2 such that $EN(F, J_1, J_2, k)$ holds. For a given formula, a given assignment and a given integer k, there exists at least one k-Evergreen-neighbor. Note that $ENF(F, J_1, k)$ specifies one local search step.
Let k_{max} be an integer $k(0 \leq k \leq n)$ that maximizes $lap(F, J, k)$ for a given formula F and a given assignment J. If we fix the k of $ENF(F, J_1, k)$ to be k_{max}, we will derive a maximal Evergreen-neighbor finder, $mENF(F, J_1)$, which takes as inputs a formula F and an assignment J_1, and returns as output an assignment J_2 such that $EN(F, J_1, J_2, k_{max})$ holds. We will specify this maximal local search step in section 4.
We construct our Evergreen local search algorithm, ELS, employing a maximal Evergreen-neighbor finder at each step until the fraction of satisfied constraints stops increasing. Note that the number of loops in this algorithm is bounded by the total number of constraints, because at least one additional constraint will be satisfied through each iteration of the loop.

```
ELS(F, J_1)
1  new ← fsat(F, J_1)
2  repeat
3          J_2 ← mENF(F, J_1)
4          old ← new
5          new ← fsat(F, J_2)
6          J_1 ← J_2
7    until old = new
```

3 Deriving τ_Γ

Given a constraint language Γ, what is the fraction of the constraints that can be satisfied in any $CSP(\Gamma)$ formula? We call this fraction τ_Γ and formalize this question as the following **infimum-maximum** problem. We denote the class of all $CSP(\Gamma)$ formulae as $\phi(\Gamma)$, and the set of all assignments of some $CSP(\Gamma)$ formula F as $\alpha(F)$.

$$\tau_\Gamma = \inf_{F \in \phi(\Gamma)} \max_{J \in \alpha(F)} fsat(F, J)$$

At first glance, this problem involves searching through all $CSP(\Gamma)$ formulae. However, we show that the search space can be reduced drastically by searching only within symmetric formulae. We will also show how to compute τ_Γ using polynomials.

3.1 Symmetric Formulae

Let π_n be the full permutation group on the n variables of some $CSP(\Gamma)$ formula F. For every $\sigma \in \pi_n$ let $\sigma(F)$ be the permuted formula, which is the result of substituting $\sigma(x)$ for all variables x in F.

Definition 2. *A $CSP(\Gamma)$ formula is called* symmetric *if any permutation of the variables in the formula returns the same formula up to a permutation of the constraints.*

Corollary 1. *If F is a symmetric $CSP(\Gamma)$ formula then for all permutations σ in π_n and all assignments J of F: $fsat(F, J) = fsat(\sigma(F), J) = fsat(F, \sigma^{-1}(J))$*

Lemma 1. *Let F be an asymmetric $CSP(\Gamma)$ formula. Symmetrize F by using the full permutation group on the n variables of F. Call these permutations $\sigma_1 \dots \sigma_{n!}$. $sym(F)$ is the concatenation of $\sigma_1(F) \dots \sigma_{n!}(F)$ and has $n! \cdot constraints(F)$ constraints. For every assignment J to F the following holds:*

$$fsat(sym(F), J) = \frac{1}{n!} \cdot \sum_{i=1}^{n!} fsat(\sigma_i(F), J)$$

Theorem 1. *For every $CSP(\Gamma)$ formula F, the symmetrized formula $sym(F)$ satisfies:*

$$\max_{Assignment:J} fsat(sym(F), J) \;\leq\; \max_{Assignment:J} fsat(F, J)$$

Proof. The proof is best explained in terms of a two-dimensional matrix (shown in table 3.1) for a given $CSP(\Gamma)$ formula F with n variables like the one shown below. The rows of the matrix correspond to the 2^n assignments for F and the columns correspond to formulae, namely to the $n!$ permutations all applied to F. The first permutation is the identity and we add one more column (the last one) to the matrix for the symmetrized formula $sym(F)$. An entry in the matrix gives the fraction of satisfied constraints by the assignment (row) to the permuted formula (column).

According to lemma 1 and the fact that if the mean of a set of numbers is f then at least one number is greater than or equal to f, the last entry of every row is less than or equal to one of the entries x in the same row. Let the column where this entry resides correspond to permutation σ_j. We construct the inverse of σ_j (because $fsat(\sigma(F), a_i) = fsat(F, \sigma^{-1}(a_i)$) and apply it to a_i. This gives us a new row corresponding to $\sigma_j^{-1}(a_i)$ and the claim is that in that row there is also the same entry x, namely in column 1 (identity permutation).

	$\sigma_1(F)$	$\ldots$	$\sigma_j(F)$	$\ldots$	$\sigma_{n!}(F)$	$sym(F)$
a_1						
$\vdots$						
a_i			x			$\leq x$
$\vdots$						
$\sigma_j^{-1}(a_i)$	x					
$\vdots$						
a_{2^n}						

Table 1. Matrix

The above argument shows that

$$\forall F \forall J \exists J' : fsat(sym(F), J) \leq fsat(F, J'),$$

where F is a $CSP(\Gamma)$ formula and J and J' are assignments for $sym(F)$ and F respectively. Now we choose the maximum assignment J_{maxsym} for $sym(F)$. The inequality also holds for this assignment. F then must have an assignment that is at least as good as J_{maxsym} for $sym(F)$. Hence, the theorem follows.

We denote the class of all symmetric $CSP(\Gamma)$ formulae as $SYM(\Gamma)$, and simplify the **infimum-maximum** problem to

$$\tau_\Gamma = \inf_{F \in SYM(\Gamma)} \max_{J \in \alpha(F)} fsat(F, J)$$

This simplification is correct because theorem 1 states that for every asymmetric formula there exists a symmetric one whose satisfaction ratio is less. It is sufficient to only minimize over symmetric formulae.

3.2 Computing τ_Γ

Mean Polynomials Given a $CSP(\Gamma)$ formula F that contains n variables, we define $mean_F(n, k)$ to be the average fraction of satisfied constraints over all assignments of which exactly k variables are assigned *true* . We reactivate the approach of computing $mean_F(n, k)$ from [1]. Let $\Gamma = \{R_1, R_2, \ldots, R_s\}$ and let $t_{R_i} (1 \leq i \leq s)$ be the fraction of constraints in F that contain relation R_i.

Lemma 2. *$mean_F(n,k)$ is a polynomial in k. Its coefficients are functions of n and $t_{R_i}(1 \leq i \leq s)$ which are linear in t_{R_i}. The degree of the polynomial is bounded by the highest rank of a relation in Γ.*

Proof. By elementary combinatorial analysis,

$$mean_F(n,k) = \sum_{i=1}^{s} t_{Ri}(F) \cdot SAT_{R_i}(n,k)$$

$$SAT_{R_i}(n,k) = \frac{\sum_{j=0}^{r(R_i)} \frac{q_j(R_i)}{\binom{r(R_i)}{j}} \cdot \binom{k}{j} \cdot \binom{n-k}{r(R_i)-j}}{\binom{n}{r(R_i)}}$$

where $r(R_i)$ is the rank of relation R_i, and $q_j(R_i)$ is the number of satisfied rows in the truth table of relation R_i when exactly j variables are set to true.

Theorem 2. *If F is a* symmetric *$CSP(\Gamma)$ formula then*

$$\max_{J \in \alpha(F)} fsat(F,J) = \max_{0 \leq k \leq n} mean_F(n,k)$$

Proof. According to corollary 1, permuting an assignment doesn't change the fraction of satisfied constraints in a symmetric formula. In other words, for a symmetric formula all that matters in an assignment is the number of true variables. Since the mean polynomial averages over assignments that set only k variables to true, the fraction of satisfied constraints predicted by $mean_F(n,k)$ is exact.

Theorem 2 allows us to further simplify the **infimum-maximum** problem to

$$\tau_\Gamma = \inf_{F \in SYM(\Gamma)} \max_{0 \leq k \leq n} mean_F(n,k)$$

This reduces the search space exponentially from size 2^n to n and it can be reduced even further using calculus.

Look-ahead Polynomials Given a $CSP(\Gamma)$ formula F and a complete assignment J of its n variables, we define a *look-ahead* polynomial, denoted as $lap(F,J,k)$, to be the average fraction of satisfied constraints over all variations of J in which exactly k variables are flipped. We call this polynomial the *look-ahead* polynomial because it looks ahead into the search space.
We define *n-map*(F,M) as a function that takes a $CSP(\Gamma)$ formula F and an assignment M and replaces each variable in F with its complement only if the variable is assigned to *true* in M. The name *n-map* comes from [3]. We assume that Γ is closed under *n-mapping*. If this is not the case, we use its closure under *n-mapping*. One can easily derive the following correspondence between *mean* and *look-ahead* polynomials.

$$lap(F,J,k) = mean_{n\text{-}map(F,J)}(n,k)$$

4 Achieving τ_Γ

We introduce two algorithms, a randomized one which achieves τ_Γ with high probability and a derandomized one which is guaranteed to achieve τ_Γ. We will use the latter to generate Evergreen local search steps of the *ELS* algorithm.

4.1 Randomized Algorithm

Given a $CSP(\Gamma)$ formula F, the randomized algorithm iterates over its variables, setting each to *true* with a probability of b. We call this algorithm RANDOMIZED-GAMBLER.

```
RANDOMIZED-GAMBLER(F, b)
1  bias a coin with respect to b
2  J ← ∅
3  for each variable x ∈ F
4        do flip the biased coin
5           if the coin lands Head
6              then J ← J ∪ x
7              else  J ← J ∪ ¬x
8  return J
```

Given a $CSP(\Gamma)$ formula F that contains c constraints, we denote by f_{avg} the average satisfaction ratio of all the assignments of F. We denote by p the probability that we find an assignment whose satisfaction ratio is no less than f_{avg} after a single iteration of RANDOMIZED-GAMBLER. We first compute a lower bound of p. Consider the worst scenario in which among the 2^n possible assignments of F, all whose satisfaction ratio is above f_{avg} satisfies exactly c constraints, whereas all whose satisfaction ratio is below f_{avg} satisfies exactly $c \cdot f_{avg} - 1$ constraints. We denote the corresponding probability by p_w.

$$c \cdot f_{avg} = p_w \cdot c + (1 - p_w) \cdot (c \cdot f_{avg} - 1)$$
$$p_w = \frac{1}{1 + c \cdot (1 - f_{avg})}$$

Intuitively, the more iterations of RANDOMIZED-GAMBLER we run, the more confident we are that the satisfaction ratio of at least one of the resulting assignments is no less than f_{avg}. We denote by δ the probability that we find an assignment whose satisfaction ratio is no less than f_{avg} after n iterations of RANDOMIZED-GAMBLER.

$$\delta = 1 - (1 - p)^n$$

We bound the number of iterations that we need in order to achieve a given probability of δ'. Since $1 - (1 - p)^n \geq 1 - (1 - p_w)^n \geq 1 - e^{-np_w}$,

we have

$$\begin{aligned} 1 - e^{-np_w} &\geq \delta' \\ n &\geq -\frac{1}{p_w} \cdot \ln(1-\delta') \\ n &\geq (1 + c \cdot (1 - f_{avg})) \cdot \ln \frac{1}{1-\delta'} \end{aligned}$$

A special case is when we set b to k_{max}/n, where k_{max} is a k that maximizes the polynomial $mean_F(n,k)$. Then, using RANDOMIZED-GAMBLER, we will find with high probability an assignment whose satisfaction ratio is no less than the maximum of what the mean polynomial predicts. We call the randomized algorithm in this scenario EVERGREEN-GAMBLER.

EVERGREEN-GAMBLER(F)
1 $b \leftarrow k_{max}/n$
2 RANDOMIZED-GAMBLER(F, b)

4.2 Derandomized Algorithm

The derandomized algorithm is a deterministic polynomial time algorithm that guarantees to return an assignment whose satisfaction ratio is no less than the maximum of what the mean polynomial predicts. We reactivate this algorithm from [1, 4], and call it EVERGREEN-PLAYER. We define REDUCE(l, F) as a function that takes a literal, l, and a formula, F, and produces a new formula which is the same as F with the variable corresponding to l assigned *true* if l is positive and assigned *false* otherwise.

EVERGREEN-PLAYER(F)
1 $k \leftarrow 0$, $tm \leftarrow mean_F(n,t)$
2 **for** $t \leftarrow 1$ **to** n
3 **do if** $mean_F(n,t) > tm$
4 **then** $k \leftarrow t$, $tm \leftarrow mean_F(n,t)$
5 $J \leftarrow \emptyset$
6 **for** each variable $x \in F$
7 **do**
8 $F_1 \leftarrow$ REDUCE(x, F)
9 $F_0 \leftarrow$ REDUCE($\neg x, F$)
10 **if** $mean_{F_1}(n-1,k-1) > mean_{F_0}(n-1,k)$
11 **then** $J \leftarrow J \cup x$, $k \leftarrow k-1$, $F \leftarrow F_1$
12 **else** $J \leftarrow J \cup \neg x$, $F \leftarrow F_0$
13 **return** J

Now we prove the correctness of EVERGREEN-PLAYER. Note that the $\binom{n}{k}$ cases of $mean_F(n,k)$, in which exactly k variables are set to *true* , can be divided into two groups: the former consists of $\binom{n-1}{k-1}$ cases that

correspond to $mean_{F_1}(n-1,k-1)$; the latter consists of $\binom{n-1}{k}$ cases that correspond to $mean_{F_0}(n-1,k)$. This implies the following recurrence relation for all k $(0 < k \leq n)$,

$$mean_F(n,k) = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} \cdot mean_{F_1}(n-1,k-1) + \frac{\binom{n-1}{k}}{\binom{n}{k}} \cdot mean_{F_0}(n-1,k) \quad (1)$$

and the following relation for $k = 0$,

$$mean_F(n,0) = mean_{F_0}(n,0)$$

We define the corner case as

$$mean_F(n,-1) = 0 \quad (2)$$

By Pascal's rule,

$$\frac{\binom{n-1}{k-1}}{\binom{n}{k}} + \frac{\binom{n-1}{k}}{\binom{n}{k}} = 1 \quad (3)$$

By equations (1) and (3), the following holds for all k $(0 \leq k \leq n)$.

$$mean_F(n,k) \leq \max\{mean_{F_1}(n-1,k-1), mean_{F_0}(n-1,k)\}$$

This means the assignment J that EVERGREEN-PLAYER returns has the following property

$$max_{0 \leq t \leq n}\{mean_F(n,t)\} \leq fsat(F,J) \quad (4)$$

4.3 Generating Evergreen Local Search Steps

It is important to recognize that if we *n-map* a $CSP(\Gamma)$ formula F with respect to the *all false* assignment, we get back F itself, i.e., $F = n\text{-}map(F, all\ false)$. According to the definition of look-ahead polynomials in section 3, we have

$$lap(F, all\ false, k) = mean_{n\text{-}map(F,all\ false)}(n,k) = mean_F(n,k)$$

If we fix k to be k_{max}, which maximizes the polynomial $mean_F(n,k)$, we get

$$lap(F, all\ false, k_{max}) = mean_F(n,k_{max}) = max_{0 \leq t \leq n}\{mean_F(n,t)\}$$

Therefore, by the inequality (4), the satisfaction ratio that EVERGREEN-PLAYER achieves has the following property

$$lap(F, all\ false, k_{max}) \leq fsat(F,J)$$

This means that what EVERGREEN-PLAYER produces is indeed a maximal Evergreen-neighbor of the *all false* assignment. We formalize this notion as:

$$\text{EVERGREEN-PLAYER}(F) = mENF(F, all\ false)$$

Conversely, we can also generate each maximal local search step ($mENF$) by composing *n-mapping* and EVERGREEN-PLAYER, thus completing our local search algorithm *ELS*.

$mENF(F, J_1)$
1 $F' \leftarrow n\text{-}map(F, J_1)$
2 $J_{aux} \leftarrow$ EVERGREEN-PLAYER(F')
3 $J_2 \leftarrow J_1$ *xor* J_{aux}
4 **return** J_2

5 Implications of τ_Γ on MAX-CSP Solvers

The insights from τ_Γ offer opportunities to improve MAX-CSP solvers. These opportunities apply to both complete solvers that provide a proof, and incomplete solvers, like stochastic local search solvers. The process of deriving and achieving τ_Γ shows that a non-trivial level of satisfaction can be reached in polynomial time. We postulate two properties that future MAX-CSP solvers will have and that the designers of these solvers will be able to prove. If a MAX-CSP solver possesses these two properties, it will have better performance on practically useful formulae.

5.1 Evergreen Law: P-optimal

We postulate that future MAX-CSP solvers will be guaranteed to construct an assignment with a satisfaction ratio no less than τ_Γ on their first try. In fact, this level of satisfaction will be obtained in time quadratic in the size of the CSP formula. This can be achieved by either the probabilistic algorithm EVERGREEN-GAMBLER or its derandomized counterpart EVERGREEN-PLAYER.

5.2 Evergreen Law: Maximal

As an iterative application of the P-optimal law, we postulate that future MAX-CSP solvers will be guaranteed to find a maximal assignment after constructing at most c assignments, where c is the total number of constraints. We consider an assignment M as maximal for a given $CSP(\Gamma)$ formula F, if

$$\max_{0 \le k \le n} mean_{n\text{-}map(F,M)}(n, k) = mean_{n\text{-}map(F,M)}(n, 0)$$

Note that if an assignment is not maximal, it cannot be maximum. A maximal assignment is not globally maximum. It is locally maximum in the sense that changing it with a maximum bias probability will not give a better assignment. Depending on Γ, finding a maximum assignment for a $CSP(\Gamma)$ formula can be NP-hard. On the other hand, finding a maximal assignment is always in P. The following algorithm finds a maximal assignment for a given $CSP(\Gamma)$ formula.

```
AGGRESSIVE-EVERGREEN-PLAYER(F)
 1  A ← all false
 2  newratio ← fsat(F, A)
 3  repeat
 4          M ← EVERGREEN-PLAYER(F)
 5          oldratio ← newratio
 6          newratio ← fsat(F, M)
 7          F ← n-map(F, M)
 8          A ← A xor M
 9     until oldratio = newratio
10  return A
```

Claim. The loop invariant of AGGRESSIVE-EVERGREEN-PLAYER is

$$oldratio \leq newratio$$

In order to prove this loop invariant, we start by proving the following property of our EVERGREEN-PLAYER.

Property 1. EVERGREEN-PLAYER returns an assignment which is at least as good as the *all false* assignment, i.e.,

$$fsat(F, \textit{all false}\ ) \leq fsat(F, \text{EVERGREEN-PLAYER}(F))$$

Proof. By the definition of EVERGREEN-PLAYER,

$$\max_{0 \leq t \leq n} mean_F(n, t) \leq fsat(F, \text{EVERGREEN-PLAYER}(F))$$

By the definition of the maximum of $mean_F(n, t)$,

$$mean_F(n, 0) \leq \max_{0 \leq t \leq n} mean_F(n, t)$$

Note that $fsat(F, \textit{all false}\ ) = mean_F(n, 0)$. Therefore, property 1 holds.

We now prove that the loop invariant of AGGRESSIVE-EVERGREEN-PLAYER holds.

Proof. Observe that *oldratio* and *newratio* correspond to the satisfaction ratios of two consecutive formulae, the latter being the *n-mapped* version of the former. We denote the former formula by F and the latter by F'.

$$oldratio = fsat(F, \text{EVERGREEN-PLAYER}(F))$$
$$newratio = fsat(F', \text{EVERGREEN-PLAYER}(F'))$$

By property 1 of EVERGREEN-PLAYER,

$$fsat(F', \textit{all false}\ ) \leq fsat(F', \text{EVERGREEN-PLAYER}(F')) = newratio$$

By the definition of *n-mapping*,

$$oldratio = fsat(F, \text{EVERGREEN-PLAYER}(F)) = fsat(F', \textit{all false}\ )$$

Thus, $oldratio \leq newratio$ holds throughout the loop.

5.3 *ELS*' Enforcement of the Evergreen Laws

Interestingly, if we apply our local search algorithm *ELS* to a given $CSP(\Gamma)$ formula F and the *all false* assignment, it expands exactly to AGGRESSIVE-EVERGREEN-PLAYER(F). Formally,

$$ELS(F, \textit{all false}) = \text{AGGRESSIVE-EVERGREEN-PLAYER}(F)$$

This implies that *ELS* is a natural enforcer of the Evergreen laws. Those MAX-CSP solvers that do not conform to the Evergreen laws can easily enforce them by employing *ELS* as their preprocessors. Given a $CSP(\Gamma)$ formula F and a MAX-CSP solver S, the preprocessing phase involves finding a maximal assignment A for F and *n-mapping* F with respect to A. Following the preprocessing, we have S solve the *n-mapped* formula and postprocess the result with respect to the original formula F. The rationale behind preprocessing is that finding a maximum assignment has the same complexity as finding an *n-map* so that *all false* is the maximum assignment.

6 Boosting MAX-CSP Solvers

We have implemented two preprocessors, one written in Scheme and the other written in Java. The Scheme implementation confines the constraint language Γ to relation OR, relation NOT and their closures under *n-mapping*. Note that this restriction simplifies a $MAX\text{-}CSP(\Gamma)$ problem to a MAX-SAT problem. We used the Scheme implementation as a preprocessor to boost the performance of an award-winning MAX-SAT solver, Toolbar[5].

The benchmark we chose is from MAX-SAT Evaluation 2007. It contains eight formulae, each of which is composed of constraints of rank 3. We allowed Toolbar twenty minutes to solve each formula and twenty minutes to solve each formula's preprocessed counterpart. The results can be divided into two categories: the four formulae (and their preprocessed counterparts) for which Toolbar succeeded in finding optimum assignments, the other four formulae for which Toolbar failed. We compare the performance of Toolbar on the original formulae with that on their preprocessed counterparts in terms of running time in the case of the former and satisfaction ratio in the case of the latter. Figure 2 illustrates the boosting effect of preprocessing as a reduction of running time and figure 3 illustrates the effect as an improvement of satisfaction ratio. Figure 4 shows time spent in preprocessing in seconds.

There is no restriction on the constraint language Γ in the Java implementation, so it was used as a preprocessor to boost the performance of an award-winning MAX-SMT solver, Yices[6]. We chose a formula containing 2000 variables and 8400 constraints (submitted by Oliver Kullman) from the SAT competition 2005 as our benchmark. The hope of this preprocessing experiment is that the fast solver will notice that the assignment *all false* is pretty good and will try to improve on it, which should lead to good results faster. Such a preprocessing experiment is a cheap way of blending the fast solvers with polynomials without having to modify the solver. Table 2 shows promising results.

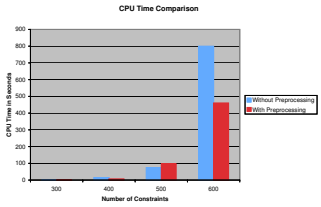

Fig. 2. Running Time

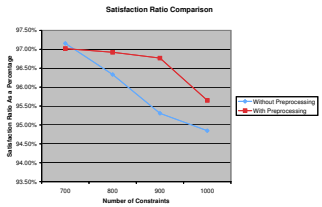

Fig. 3. Satisfaction Ratio

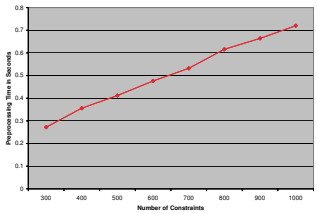

Fig. 4. Preprocessing Time

7 Related and Future Work

Our Boolean MAX-CSP is a special case of Weighted Constraint Satisfaction Problem (WCSP). WCSP is an optimization version of the CSP framework in which constraints are extended by associating costs to tuples. Solving a WCSP formula consists of finding a complete assignment of minimal cost. Our Boolean MAX-CSP is a special case of WCSP because our domain is only Boolean and because the tuples can have only two costs: zero when the relation is satisfied and a positive cost when the relation is unsatisfied. Several kinds of algorithms have been proposed to solve WCSP: Bucket Elimination [7] and several Branch and Bound algorithms (see [8] for an enumeration). None of those WCSP algorithms is using polynomials as abstract representations of WCSP problems. Although we present our results for Boolean MAX-CSP, the techniques also generalize to MAX-CSP, see section 8 of [1].

[9] discusses local search algorithms both for SAT and MAX-SAT and shows that local search outperforms complete algorithms on certain formulae. While traditional local search algorithms have a very simple neighborhood relation, our neighborhood relation is more complex but also efficiently computable. For example, Selman and Kautz have studied local search for SAT [10]. They use a traditional neighborhood notion: specifically, they explore the set of assignments that differ from the current one on only one variable. We use a more refined notion of neighborhood for MAX-SAT and MAX-CSP and prove an optimality result for our neighborhood concept. In addition, while traditional local search algorithms look for a largest increase or decrease within the neighborhood, we only find one point in the neighborhood. It should be noted, however, that we can also generate a large number of different assignments in the

Yices	Running Time (s)	Satisfaction Ratio
Without Preprocessing	888.048	94.7143%
With Preprocessing	0.0342615	100%

Table 2. Boosting Effect on Yices

neighborhood. A random permutation of the variables in the formula is likely to lead to a different assignment when *mENF* or *ELS* is applied to the formula. Our basic neighborhood relation *EN* can be used in different ways to create local search algorithms, as suggested in [9] (e.g. a random walk strategy).

Hoos and Stutzle [11] have studied automata-based local search approaches. On one hand, we can reformulate our local search algorithms in terms of their generalized local search machines and we plan to do so in future work. On the other hand, the golden ratio technique can be used to create several basic search strategies for generalized local search machines.

Preprocessing for SAT solvers is currently an active topic of research, e.g., [12]. Stochastic local search solvers may also benefit from a preprocessing phase borrowed from systematic SAT solving [13]. The kind of preprocessing we propose is novel, very different from resolution-based techniques. Not only can the polynomials be applied to a preprocessor, they can also be used in the Decide rule of a transition system for MAX-CSP problems, as described in [14]. The transition system is based on clause learning (superresolution) which was introduced at least as far back as 1975 [15].

Our motivation for working on MAX-CSP is that we think that it has important applications in biology and drug discovery. A recent paper from SRI confirms this conjecture [16] where MAX-SAT is used to analyze biological pathways.

8 Conclusions

Boolean MAX-CSP has numerous practical applications. For example, it serves as a flexible target language for many NP-hard optimization problems. The process of deriving and achieving τ_Γ reveals useful polynomial-based local search algorithms for boosting MAX-CSP solvers. We believe that the the Evergreen laws are beneficial additions to the bag of tricks used in powerful Boolean MAX-CSP and WCSP solvers.

Acknowledgments: We would like to thank Ravi Sundaram for helping with the bound for the randomized algorithm, Bryan Chadwick for helping implement the preprocessor and Leonardo de Moura for his feedback on our work. We would also like to thank Richard M. Conlan for his feedback on the final draft of this paper. Last but not least, we would like to thank Novartis Institutes for Biomedical Research, Inc. for supporting this work. Karl Lieberherr spent his 2006 sabbatical at Novartis and Christine Hang is supported by a Novartis fellowship.

References

1. Lieberherr, K.J.: Algorithmic extremal problems in combinatorial optimization. Journal of Algorithms **3**(3) (1982) 225–244
2. Lieberherr, K.J., Specker, E.: Complexity of Partial Satisfaction. Journal of the Association for Computing Machinery **28**(2) (1981) 411–421
3. Borchert, B., Ranjan, D., Stephan, F.: On the computational complexity of some classical equivalence relations on boolean functions. Theory of Computing Systems **31** (1998) 679–693 http://math.uni-heidelberg.de/logic/berichte.html, Report 18.
4. Williamson, D.P.: Lecture notes on approximation algorithms. Technical Report RC 21409, IBM Research (1999)
5. Toolbar: . (http://mulcyber.toulouse.inra.fr/projects/toolbar/)
6. Yices: . (http://yices.csl.sri.com)
7. Dechter, R.: Bucket elimination: a unifying framework for processing hard and soft constraints. ACM Comput. Surv. **28**(4es) (1996) 61
8. Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: The Logic Behind Weighted CSP. In Veloso, M.M., ed.: IJCAI. (2007) 32–37
9. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In Trick, M., Johnson, D.S., eds.: Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability, Providence RI (1993)
10. Selman, B., Kautz, H.A.: An empirical study of greedy local search for satisfiability testing. In: Proceedings of theEleventhNational Conference on Artificial Intelligence(AAAI-93), Washington DC (1993)
11. Hoos, H.H., Stutzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann Publishers (2004)
12. Anbulagan, Slaney, J.: Multiple Preprocessing for Systematic SAT Solvers. In: IWIL-6, as part of LPAR-2006, Phnom Penh, Cambodia (2006)
13. Anbulagan, Duc Nghia Pham, J.S., Sattar, A.: Boosting sls performance by incorporating resolution-based preprocessor. In: Third International Workshop on Local Search Techniques in Constraint Satisfaction, Springer Verlag (*LNCS* 244) (2006)
14. Abdelmeged, A., Hang, C., Rinehart, D., Lieberherr, K.J.: Superresolution and P-Optimality in Boolean MAX-CSP Solvers. Technical Report NU-CCIS-07-01, Northeastern University (2007)
15. Karl Lieberherr: Information Condensation of Models in the Propositional Calculus and the P=NP Problem. PhD thesis, ETH Zurich (1977) 145 pages, in German.
16. A. Tiwari and C. Talcott and M. Knapp and P. Lincoln and K. Laderoute: Analyzing Pathways using SAT-based Approaches. In: Proc. 2nd Intl. Conf. on Algebraic Biology, AB 2007. LNCS, Springer (2007)