

The Evergreen Game: The Promise of Polynomials to Boost Boolean MAX-CSP Solvers

Ahmed Abdelmegeed Christine Hang Daniel Rinehart Karl Lieberherr

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

Abstract. We invent a simple game, called the Evergreen Game, which is about generating and solving Boolean MAX-CSP problems. The fall-outs from the Evergreen Game are surprising: (1) Although the game is about solving MAX-CSP problems, a simple, efficient algorithm is sufficient to play the game well. The best game-playing strategy leads to a significant reduction of the huge search space for both formula generation and solving. (2) The Evergreen Game shows us how to systematically translate a CSP formula into a polynomial that is fundamental in playing the game well. (3) We show evidence that those polynomials are useful for efficient MAX-CSP as well as MAX-SAT solvers.

1 Introduction

Do we really need backtracking search to solve MAX-CSP problems? We introduce a simple MAX-CSP-based game, called the Evergreen Game, that involves quickly finding assignments that satisfy many constraints. We show, surprisingly, that in the context of this game, although it involves an NP-hard optimization problem, backtracking search is not needed to play the game well¹.

A Boolean CSP formula is a non-empty bag of constraints. From now on, we simply use CSP formula. A constraint consists of an integer multiplicity, a Boolean relation of some rank r and a set of r variables. The multiplicity indicates how often the constraint appears in the bag. A Boolean relation of rank r is defined by a Boolean formula involving r variables. 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 . A constraint is satisfied under assignment J if the Boolean formula evaluates to true under assignment J . Let Γ be a set of relations, other works call Γ a constraint language. A $CSP(\Gamma)$ -formula is a CSP formula that only contains relations in Γ . A $MAX-CSP(\Gamma)$ problem has as input a $CSP(\Gamma)$ -formula F and as output an assignment J such that $fsat(F, J)$ is maximized.

¹ Although we present our results for Boolean MAX-CSP, the techniques also generalize to MAX-CSP, see section 8 of [1].

The Evergreen Game is played by two players, Anna and Bob, that take turns creating and solving CSP formulae and paying each other a percentage of a wager based on the fraction of constraints satisfied. Let the wager w be 1 million dollars and the constraints limited to $\Gamma = \{OR(x, y), NOT(x)\}$. Anna starts by constructing $F_{Initial} = \{100 : NOT(x), 150 : NOT(y), 200 : OR(x, y)\}$. Bob tries to find an assignment that satisfies the largest possible fraction of constraints. For example, the assignment $\{x = true, y = false\}$ will satisfy $(150 + 200)/450 \approx 0.78$. Anna then pays Bob 0.78 million dollars ($w * 0.78$). Bob now constructs a formula that Anna solves and pays Anna the percentage of the wager that she solved.

In this case Anna's first move ($F_{Initial}$) was irrational. She could have chosen another formula in which a smaller percentage could have been satisfied by Bob, saving her money. Some important questions about the Evergreen Game are: How can Anna limit the amount she has to pay to Bob? How should Anna construct the formula? How can Bob quickly satisfy many constraints?

The reader is encouraged to play the game with the two given relations a few rounds to develop a feeling for degrees of freedom. This is a great opportunity for experiential learning and discovery. Hint: Bernstein polynomials play an important role both for formula generation as well as for solving the CSP formula.

A key idea behind the game is the insight that it is sufficient for Anna to search among a subset S of CSP formulae. This drastically cuts the search space and the search in this subset turns out to be tractable by using polynomial maximization and minimization techniques. The set S has the property that if Anna selects a CSP formula F outside S we can construct a new formula in S with the same number of variables as F , but with a lower satisfaction fraction using the maximum assignment. Therefore, it is not in the best interest of Anna to choose a formula outside S . Bob's MAX-CSP solver is also based on the polynomials that Anna used to construct her formula.

1.1 Contributions

The paper makes the following contributions:

- **Surprise: No backtracking needed.** Although the game involves an NP-hard optimization problem, we present a polynomial algorithm to play it well. The game is a draw when both players act rationally and we show that the draw can be achieved by a polynomial-time algorithm. The algorithm might win money if the other player acts irrationally. Bob can always efficiently find an assignment for the formula that Anna gives him so that it is as good as if Anna would have used her best answer.
 Existing MAX-SAT algorithms (like yices or the algorithms in UBC-SAT) don't offer guarantees to play the game without losing money.
- **Technology derivation with potential MAX-CSP solver impact.** The simple Evergreen Game acts as the generator of technology that might be useful for MAX-CSP solvers in general. The Evergreen Game relies on polynomials that abstractly represent CSP

formulae. These polynomials lead to P-optimal algorithms [1]: An algorithm for $MAX-CSP(I)$ is P-optimal if it guarantees to satisfy a fraction τ_I of all constraints and the set of formulae in which the fraction $\tau_I + \epsilon$ ($\epsilon > 0$) can be satisfied, is NP-complete.

- **New P-optimal thresholds.** We present new P-optimal thresholds that limit the loss a player faces in the worst case. Those thresholds extend the results in [2–4]. The threshold computation is automated using Mathematica for all pairs of relations up to rank 3.

1.2 Paper Organization

The rest of the paper is organized as follows: In section 2, we present the Evergreen Game. In section 3, we analyze the game from Anna’s perspective. We also derive the optimal strategy for Anna. In section 4, we analyze the game from Bob’s perspective. We also provide two algorithms for Bob to play the game optimally. In section 5, we compare the performance of a number of MAX-SAT solvers playing the Evergreen Game. In section 6, we discuss the implications of the Evergreen Game on future MAX-CSP solvers. Sections 7 and 8 conclude this paper.

2 The Evergreen Game

The Evergreen Game is a two-player game focused on constructing and solving Boolean MAX-CSP problems. The players win and lose money based on how good they solve those problems. Let $Evergreen(r, s)$ define a class of games where r is the maximum rank of relations used in the formulae and s is the number of distinct relations used in a formula.

The game consists of a setup phase and two rounds. In the setup phase the players jointly decide: the set of s distinct relations to use (observing the r maximum rank restriction), the total number of variables that can be used (denoted as n), which player will go first, the wager, and an action time-limit. Each round consists of a formula construction action, an assignment production action, and a money settlement phase. During the formula construction action the first player constructs a CSP formula that uses no more than n variables, that contains at least one constraint, and that consists only of relations from the agreed upon set. Duplication of a variable in a relation is not permitted. At or before the action time-limit the first player gives the constructed formula to the second player. During the assignment production action the second player finds an assignment that satisfies as many constraints as possible in the received formula. At or before the action time-limit the second player gives the assignment to the first player. In the money settlement phase the first player pays the second player the percentage of the wager equal to the percentage of constraints satisfied by the second player’s assignment when applied to the formula created by the first player. Should either player fail to construct a valid formula or assignment the entire wager is forfeit to the other player and the game ends immediately. The second round is played the same as the first, but with the player order reversed.

As an example Anna and Bob are playing the game with a wager of \$100 and Anna is going first. In the first round Anna constructs a formula, Bob finds an assignment that satisfies 75% of Anna's formula, and Anna pays Bob \$75. In the second round Bob constructs a formula, Anna finds an assignment that satisfies 80% of Bob's formula, and Bob pays Anna \$80, resulting in a net loss of \$5 for Bob.

3 Anna's Perspective

Anna's objective is to minimize her loss. Therefore, she is looking for a CSP formula for which the best assignment Bob can find is minimum.

3.1 Symmetric Formulae

Let π_n be the full permutation group on the n variables of 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 1. *A CSP formula is called symmetric if any permutation of the variables in the formula returns the same formula up to a permutation of the constraints.*

According to this definition, $F_{Initial}$ is not symmetric.

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

Now, we can formally state Anna's objective as:

$$\tau_\Gamma = \inf_{CSP\ formula:F} \max_{Assignment:J} fsat(F, J)$$

Anna is searching for a CSP formula where Bob cannot satisfy more than τ_Γ of the constraints. Anna can cut her search space drastically by searching only for symmetric formulae.

Lemma 1. *Let F be an asymmetric CSP 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 asymmetric CSP formula F , the symmetrized formula $sym(F)$ satisfies:*

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

Proof. We first prove \leq for all CSP formulae and later $<$ for asymmetric formula. The proof is best explained in terms of a two-dimensional matrix for a given CSP 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 $fsat(\sigma(F), a_i) = fsat(F, \sigma_j^{-1}(a_i))$) 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).

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

The above argument shows that

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

where F is a CSP formula and J and J' are assignments for $sym(F)$ and F respectively. Now we choose the maximum assignment J_{maxsym} for $sym(F)$. Also for this assignment the inequality holds. Therefore, F must have an assignment that is at least as good as J_{maxsym} for $sym(F)$. Hence, the theorem follows with the \leq sign substituted for $<$. Now if F is asymmetric, then by definition not all its assignments have the same satisfaction ratio. Hence the theorem follows.

Now, Anna's objective is simplified and can be restated as:

$$\tau_T = \inf_{Symmetric\ CSP\ formulae:F} \max_{Assignment:J} fsat(F, J)$$

This is correct because theorem 1 states that for every asymmetric formula there exists a symmetric one which is worse. It is sufficient to minimize among the symmetric formulae because in the asymmetric ones larger fractions can be satisfied which do not influence the minimum.

3.2 Anna's Best Strategy:

Anna generates symmetric formulae by filling in parameters in a template $T \langle n, m_1 \cdots m_s \rangle$ where n is the number of variables, s is the number of relations in Γ , and m_i is the multiplicity for relation R_i in Γ . The template has for every relation R_i in Γ , $C_n^{\text{rank}(R_i)}$ constraints with multiplicity m_i .

Now, Anna's objective can be restated as:

$$\inf_{\langle n, m_1 \cdots m_s \rangle} \max_{\text{Assignment: } J} \text{fsat}(T \langle n, m_1 \cdots m_s \rangle, J)$$

3.3 Mean Polynomials

According to theorem 2, Anna's search space can be reduced further using *mean* polynomials. This reduces the search space exponentially from size 2^n to n and it can be reduced even further by using calculus. We define $\text{mean}_F(n, k)$ to be the average fraction of satisfied constraints in F among all assignments to the n variables of F that set exactly k variables to true. By elementary combinatorial analysis we get the following:

$$\begin{aligned} \text{mean}_F(n, k) &= \sum_{i=1}^s t_{R_i}(F) \cdot \text{SAT}_{R_i}(n, k) \\ \text{SAT}_{R_i}(n, k) &= \frac{\sum_{j=0}^{\text{rank}(R_i)} \frac{q_j(R_i)}{C_j^{\text{rank}(R_i)}} \cdot C_j^k \cdot C_{\text{rank}(R_i)-j}^{n-k}}{C_{\text{rank}(R_i)}^n} \end{aligned}$$

Where $t_{R_i}(F)$ is the fraction of constraints in F that contain relation R_i , $t_{R_i}(F) = \frac{m_i}{(m_1 + \dots + m_s)}$. s is the number of relations in Γ . $q_j(R)$ is the number of rows with j variables set to true in the truth table of relation R .

Theorem 2. *If F is a symmetric CSP formula then:*

$$\max_{\text{all assignments } J \text{ for } F} \text{fsat}(F, J) = \max_{0 \leq k \leq n} \text{mean}_F(n, k)$$

Proof. According to corollary 1, permuting an assignment doesn't change the fraction of satisfied constraints in a symmetric formula. In other words, in a symmetric formula all that matters is the number of true variables.

This allows us to restate Anna's problem as:

$$\tau_\Gamma = \inf_{\langle n, m_1 \cdots m_s \rangle} \max_{0 \leq k \leq n} \text{mean}_{T \langle n, m_1 \cdots m_s \rangle}(n, k)$$

3.4 Approximate Mean Polynomials

We approximate the *mean* polynomials using *appmean* polynomials. The $appmean_F(x)$ polynomials predict the expected fraction of satisfied constraints of F when each variable in F is set to true with probability x .

In [4] it is shown that the *appmean* polynomials give the correct P-optimal thresholds. Therefore, Anna may use the *appmean* polynomials to predict her loss in the worst case.

$$appmean_F(x) = \sum_{i=1}^s t_{R_i}(F) \cdot appSAT_{R_i}(x)$$

$$appSAT_{R_i}(x) = \sum_{j=0}^{rank(R_i)} q_j(R_i) \cdot x^j \cdot (1-x)^{rank(R_i)-j}$$

3.5 Automated P-optimal Threshold Computation for Evergreen(3,2) using Mathematica

When Anna is constructing her $CSP(\Gamma)$ -formula, she wants to minimize the number of constraints that can be easily satisfied by Bob, even if he is using *mean* polynomials or any other clever and efficient technique. Once the Γ for the game is chosen it is possible to compute the fraction that each R_i should exist in the $CSP(\Gamma)$ -formula to produce the worst best case scenario. Anna wants to find the fraction t_{R_i} for each R_i such that:

$$\inf_{0 \leq t_{R_i} \leq 1, \sum_{i=1}^n t_{R_i} = 1} \max_{0 \leq x \leq 1} \sum_{i=1}^n t_{R_i} \cdot appSAT_{R_i}(x)$$

To easily refer to all possible R_i constraints used in $appSAT_{R_i}(x)$, a shorthand notation is used. If each constraint is fixed at 3 variable positions (with some variable positions playing no role in satisfiability) the truth table for a given constraint will contain 8 rows. Each row consists of a boolean value for each variable position and are ordered $\{(f, f, f), (f, f, t), (f, t, f), \dots\}$. Each row has a corresponding 2^n value associated with it, $\{1, 2, 4, \dots\}$ respectively. The shorthand notation for a constraint is the sum of values for the rows that its truth table it true. R_{22} is true for rows $\{(f, f, t), (f, t, f), (t, f, f)\}$ which correspond to $\{2, 4, 16\}$, for a sum of 22.

With $rank(R)$ fixed at 3, the $appSAT_{R_i}(x)$ computation is coded as:

```
includeRow[Ri_, r_] := Mod[Floor[Ri/r], 2];
satisfyingRows[Ri_, s_] := Switch[s,
  0, includeRow[Ri, 1],
  1, includeRow[Ri, 2] + includeRow[Ri, 4] + includeRow[Ri, 16],
  2, includeRow[Ri, 8] + includeRow[Ri, 32] + includeRow[Ri, 64],
  3, includeRow[Ri, 128]];
appSAT[Ri_] :=
  Sum[satisfyingRows[Ri, s] · xs · (1-x)(3-s), {s, 0, 3}];
```

If F is fixed at 2 relations, the *mean* polynomial for the $CSP(F)$ -formula is represented as:

$$combineTwo[R1_, R2_] := t1 \cdot appSAT[R1] + t2 \cdot appSAT[R2];$$

The maximum bias for a *mean* polynomial will occur at $x = 0$, $x = 1$, or a stationary point in $0 \leq x \leq 1$. Based on the degree of the *mean* polynomial i.r.t. x there maybe 0, 1, or 2 stationary points to consider. In the case of a *mean* polynomial of degree 2, the maximum is modled as:

$$\begin{aligned} maxX2[F_] := & \\ & With[\{xSP = Root[(D[F, x]/.x \to \#)\&, 1]\}, \\ & With[\{f0 = F/.x \to 0, f1 = F/.x \to 1, fSP = F/.x \to xSP\}, \\ & Piecewise[\{\{xSP, fSP > f0 \&\& fSP > f1\}, \\ & \{1, f1 > f0 \&\& f1 \geq fSP\}, 0]]]; \end{aligned}$$

The *Piecewise* function models the maximum in light of unknown values for $t1$ and $t2$, by symbolically determining the stationary point and including it in the calculation. $Piecewise[\{\{val_1, cond_1\}, \{val_2, cond_2\}, \dots\}, val_d]$ represents a piecewise function with values val_i in the regions defined by the conditions $cond_i$ and uses default value val_d if none of the $cond_i$ apply. To support an arbitrary pair of relations a branching function named *whichMaxX* is used to select the correct *Piecewise* function based on the degree of the *mean* polynomial. Using the constraints defined above that $t1 + t2 = 1$ and $0 \leq t1 \leq 1$ the *Minimize* function is used to calculate the worst case scenario:

$$\begin{aligned} minMax[R1_, R2_] := & With[\{F = combineTwo[R1, R2]\}, \\ & Minimize[\{F/.x \to whichMaxX[F], 0 \leq t1 \leq 1, t1 + t2 = 1\}, \\ & \{t1, t2\}]]; \end{aligned}$$

The use of *Root* in the *Piecewise* function to model the value of a stationary point can, for certain values of $t1$, lead to the denominator of the stationary point becoming 0. Such values represent the degree of the *mean* polynomial changing. This usually occurs at either $t1 = 0$ or $t2 = 0$. In such cases Mathematica reports a warning that there is no minimum in the region described by the constraints and returns the proper boundary. If the critical point occurs elsewhere, such as in the case of 1 and 128 which has an undefined stationary point value at $t1 = \frac{1}{2}$, additional calculations must be used to include that value in the minimization calculation. The complete Mathematica program that incorporates these additional calculations is available at [5].

P-optimal Thresholds Table 1 lists P-optimal thresholds obtained from the Mathematica program described above. Let $F(p, q)$ be a constraint that is only satisfied when it has exactly p positive and exactly q

Table 1. P-Optimal Thresholds

	1	3	22	60	104	128
1	1					
3	1	1				
22	$\frac{27}{64}$	$\frac{4}{9}$	$\frac{4}{9}$			
60	$-\frac{2}{27}(135 - 81\sqrt{3})$	$\frac{4}{9}$	$\frac{4}{9}$	$\frac{1}{2}$		
104	$\frac{1}{4}(3 - \sqrt{3})$	$\frac{3}{14}(3 - \sqrt{2})$	$\frac{3}{8}$	$\frac{4}{9}$	$\frac{4}{9}$	
128	$\frac{1}{2}$	$\frac{1}{4}(1 + \sqrt{3})$	$\frac{1}{4}(3 - \sqrt{3})$	$-\frac{2}{27}(135 - 81\sqrt{3})$	$\frac{27}{64}$	1

negative literals. The $F(p, q)$ and corresponding relation numbers used in the table are: $F(0, 3) = 1$, $F(0, 2) = 3$, $F(1, 2) = 22$, $F(1, 1) = 60$, $F(2, 1) = 104$, and $F(3, 0) = 128$.

4 Bob’s Perspective

Bob has an ideal gambling strategy by which he never loses money. The key of such a strategy lies in both constructing an efficient algorithm A which guarantees to satisfy τ_Γ of any formula and giving back to Anna the hardest formula to solve.

Let the formula Bob receives from Anna be F_{Anna} , and let the fraction of F_{Anna} satisfied in polynomial time by applying algorithm A be $f_{Anna} = A(F_{Anna})$, and let the fraction of the formula Bob gives back to Anna satisfiable in polynomial time be f_{Bob} . Therefore, we can represent Bob’s gain as $(f_{Anna} - f_{Bob}) \cdot w$, where w is the wager. f_{Bob} is the minimum over all formulae, therefore $f_{Bob} \leq f_{Anna}$. If Bob can construct a polynomial time algorithm which guarantees to satisfy the fraction f_{Anna} of the formula that he receives from Anna, he will never lose money. We introduce two gambling tools for Bob, a randomized algorithm by which with high probability Bob will not lose money and a derandomized algorithm which guarantees Bob will not lose money.

4.1 Randomized Algorithm

Let F be the formula Bob receives from Anna. The randomized algorithm $\text{RANDOMIZED-GAMBLER}(b)$ goes through all the variables and sets each of them to *true* with probability b . If we set b to the x in $\text{appmean}_F(x)$ defined in section 3.4, $\text{RANDOMIZED-GAMBLER}(b)$ will find with high probability an assignment that satisfies a fraction no less than what the appmean polynomial predicts. Intuitively, the more iterations we run this algorithm, the higher the probability will be. We first introduce this randomized algorithm and then bound the number of iterations we need in order to achieve a given probability.

```

RANDOMIZED-GAMBLER( $b$ )
1  bend a coin according to  $b$ 
2   $J \leftarrow \emptyset$ 
3  for each variable  $x \in F$ 
4      do flip the bent coin
5      if it is heads
6          then  $J \leftarrow J \cup x$ 
7          else  $J \leftarrow J \cup \neg x$ 
8  return  $J$ 

```

For an arbitrary formula F containing c constraints, we denote by f_{avg} the satisfiable fraction that the *appmean* polynomial predicts, i.e. $f_{avg} = \text{appmean}_F(x)$ for some $0 \leq x \leq 1$. We denote by p the probability that after a single iteration the RANDOMIZED-GAMBLER finds an assignment satisfying a fraction greater than f_{avg} .

We first compute a lower bound of p . Consider the worst senario 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 probability in this worst senario by p_w . Thus, $p \geq p_w$.

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

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

The probability that the RANDOMIZED-GAMBLER finds after n iteration an assignment satisfying a fraction greater than f_{avg} is $\delta = 1 - (1 - p)^n$. Since $1 - (1 - p)^n \geq 1 - (1 - p_w)^n \geq 1 - e^{-np_w}$, therefore, one needs to make the number of iterations satisfy the following condition in order to achieve a given probability δ .

$$1 - e^{-np_w} \geq \delta$$

$$n \geq -\frac{1}{p_w} \cdot \ln(1 - \delta)$$

$$n \geq -(1 + c \cdot (1 - f_{avg})) \cdot \ln(1 - \delta)$$

A special case is when we set b to the x that maximizes $\text{appmean}_F(x)$, then the RANDOMIZED-GAMBLER will find with high probability an assignment that satisfies a fraction no less than the maximum of what the *appmean* polynomial predicts. We name the randomized algorithm in this situation as EVERGREEN-GAMBLER and introduce it as follows.

```

EVERGREEN-GAMBLER( $F$ )
1   $b \leftarrow \arg \max_{0 \leq x \leq 1} \{\text{appmean}_F(x)\}$ 
2  RANDOMIZED-GAMBLER( $b$ )

```

4.2 Derandomized Algorithm

We define $\text{REDUCE}(l, F)$ 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. The derandomized algorithm is a deterministic algorithm which guarantees to satisfy the maximum fraction the mean polynomial predicts in polynomial time. We reconcile this algorithm from [1, 6]. Let F be the formula Bob receives from Anna.

```

EVERGREEN-PLAYER( $F$ )
1   $k \leftarrow 0, tm \leftarrow \text{mean}_F(n, t)$ 
2  for  $t \leftarrow 1$  to  $n$ 
3      do if  $\text{mean}_F(n, t) > tm$ 
4          then  $k \leftarrow t, tm \leftarrow \text{mean}_F(n, t)$ 
5   $J \leftarrow \emptyset$ 
6  for each variable  $x \in F$ 
7      do
8           $F_1 \leftarrow \text{REDUCE}(x, F)$ 
9           $F_0 \leftarrow \text{REDUCE}(\neg x, F)$ 
10     if  $\text{mean}_{F_1}(n-1, k-1) > \text{mean}_{F_0}(n-1, k)$ 
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  $J$ 

```

Intuitively, $\text{mean}_F(n, k)$ can be decomposed into $\text{mean}_{F_1}(n-1, k-1)$ and $\text{mean}_{F_0}(n-1, k)$. Among the $\binom{n}{k}$ cases in which exactly k variables are set to *true*, $\binom{n-1}{k-1}$ cases belong to the former, whereas $\binom{n-1}{k}$ cases belong to the latter. Therefore, we can derive the following recurrence relation for all $0 < k \leq n$.

$$\text{mean}_F(n, k) = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} \cdot \text{mean}_{F_1}(n-1, k-1) + \frac{\binom{n-1}{k}}{\binom{n}{k}} \cdot \text{mean}_{F_0}(n-1, k)$$

for $k = 0$,

$$\text{mean}_F(n, 0) = \text{mean}_{F_0}(n, 0)$$

for $k = -1$,

$$\text{mean}_F(n, -1) = 0$$

Observe that $\frac{\binom{n-1}{k-1}}{\binom{n}{k}}$ and $\frac{\binom{n-1}{k}}{\binom{n}{k}}$ sum up to 1 by Pascal's rule, so for all $0 < k \leq n$, $\text{mean}_F(n, k) \leq \max\{\text{mean}_{F_1}(n-1, k-1), \text{mean}_{F_0}(n-1, k)\}$. Therefore, the assignment J , which algorithm EVERGREEN-PLAYER produces, satisfies $\text{fsat}(F, J) \geq \max_{0 \leq t \leq n} \{\text{mean}_F(n, t)\}$.

5 Evaluation of MAX-SAT Solvers on Evergreen

We play *Evergreen*(2,2), described in section 2, with some MAX-SAT solvers that performed well in competitions. We used three solvers: Yices and Toolbar (complete solvers) and UBCSAT a weighted novelty+ algorithm (incomplete solver). Assume that Anna acts rationally and gives Bob the hardest formulae (symmetric formulae) containing 5, 100 and 1000 variables respectively, and Bob employs both our algorithm EVERGREEN-GAMBLER and EVERGREEN-PLAYER (EG-Gambler and EG-Player) and the three other competition-winning solvers above to solve these formulae. We ran the experiments on a Linux machine (Ubuntu Dapper Drake 6.06) with a 2.0 GHz Intel Pentium M processor and 512MB of RAM, and illustrate the corresponding results (in seconds) in table 2. *X* represents “Program failed to terminate within 20 minutes”, and *Y* represents “Insufficient memory. Aborting Program.”.

Player	5 variables		100 variables		1000 variables	
	time	<i>fsat</i>	time	<i>fsat</i>	time	<i>fsat</i>
EG-Gambler	0.336	0.63902	0.992	0.61771	30.138	0.61815
EG-Player	0.332	0.64390	1.020	0.61911	57.356	0.61816
UBCSAT	0.096	0.64390	1.052	0.61911	152.782	0.61728
Yices	0.002999	0.64390		X		X
Toolbar	0	0.64390		X		Y

Table 2. Performance of MAX-SAT Solvers

6 Implications of the Evergreen Game on MAX-CSP Solvers

The insights from the Evergreen Game offer opportunities to improve MAX-CSP solvers, both complete solvers that provide a proof, and incomplete solvers, like stochastic local search solvers. The Evergreen Game shows that a non-trivial level of satisfaction can be reached in polynomial time. We would like to postulate two properties that future MAX-CSP solvers should have and that the designers of the solvers should be able to prove. If a MAX-CSP solver satisfies these laws it not only becomes useful for playing the Evergreen Game but more importantly, it should also have better performance on practically useful formulae.

6.1 Evergreen Law: Loss-Free

We postulate that future MAX-CSP solvers should guarantee, on their first try to construct an assignment, at least at the level of satisfaction that bounds Anna’s loss. This level of satisfaction must be reached

quickly, i.e., in quadratic time in the size of the CSP formula. This can be achieved either by the probabilistic algorithm Evergreen-Gambler or its derandomized version Evergreen-Player.

6.2 Evergreen Law: Maximal

This is an iterative application of the Loss-Free law. We postulate that future MAX-CSP solvers should guarantee to find a maximal assignment (defined below) after constructing at most *total* assignments, where *total* is the total number of constraints.

An assignment M is considered maximal for a given CSP formula F , if

$$\max_{0 \leq k \leq n} \text{mean}_{n\text{-map}(F,M)}(n, k) = \text{mean}_{n\text{map}(F,M)}(n, 0)$$

where $n\text{-map}(F, M)$ is a function that takes a CSP 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 [7]. Note that if an assignment is not maximal, it cannot be a maximum assignment. 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 formula. In each iteration of the main loop, the number of satisfied constraints increases by at least one.

AGGRESSIVE-PLAYER(F)

```

1   $M \leftarrow A \leftarrow$  all false
2  repeat
3       $F \leftarrow n\text{-map}(F, M)$ 
4       $M \leftarrow$  EVERGREEN-PLAYER( $F$ )
5       $Mean_0 \leftarrow \text{mean}_{n\text{map}(F,M)}(n, 0)$ 
6       $MaxMean \leftarrow \max_{0 \leq k \leq n} \text{mean}_{n\text{-map}(F,M)}(n, k)$ 
7       $A \leftarrow A$  xor  $M$ 
8  until  $MaxMean = Mean_0$ 
9  return  $A$ 

```

It is in Bob's best interest to find a maximal assignment for the formula he gets from Anna. If Anna acts rationally, algorithm Evergreen-Player will return a maximal assignment. But if Anna acts irrationally, Bob will make her pay for her risky behavior by returning a maximal assignment considerably better than what algorithm Evergreen-Player produced in one step.

7 Related and Future Work

Our Boolean MAX-CSP is a special case of Weighted Constraint Satisfaction (WCSP). We believe that the techniques presented in this paper could be applied to full WCSP. The Weighted Constraint Satisfaction Problem is a well known soft constraint framework for modelling over-constrained problems with several practical applications. 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: 0 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 [8] and several Branch and Bound algorithms (see [9] for an enumeration). None of those WCSP algorithms is using polynomials as abstract representations of WCSP problems.

We propose to put the Evergreen law Maximal into action by using algorithm Aggressive-Player to create a maximal assignment J for an input formula F and to feed $n\text{-map}(F, J)$ to a fast solver (any MAX-SAT, WCSP, MAX-CSP, or SAT solver). The hope of this preprocessing experiment is that the fast solver will notice that the assignment *all false* is pretty good and it 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 mean polynomials without having to modify the solver. This preprocessing experiment exploits the fact that finding a good assignment is the same as finding a good n-map. Preprocessing for SAT solvers is currently an active topic of research, e.g., [10]. The kind of preprocessing we propose is novel, very different from resolution-based techniques.

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 [11] 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 Evergreen Game is Boolean MAX-CSP in game form. Finding optimal strategies for the players reveals useful algorithms for enhancing MAX-CSP solvers. We believe that the the Evergreen laws Loss-Free and Maximal 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.

We would like to thank Novartis Institutes for Biomedical Research, Inc. for supporting this work. Karl Lieberherr spent his 2006 sabbatical at Novartis. 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. Huang, M., Lieberherr, K.J.: Implications of forbidden structures for extremal algorithmic problems. *Theoretical Computer Science* **40** (1985) 195–210
4. Lieberherr, K., Specker, E.: Complexity of Partial Satisfaction II. Technical Report 293, Princeton University, Dept. of EECS (1982) <http://www.ccs.neu.edu/research/demeter/biblio/partial-sat-II.html>.
5. Abdelmegeed, A., Hang, C., Rinehart, D., Lieberherr, K.J.: The Evergreen Game: The Promise of Polynomials to Boost Boolean MAX-CSP Solvers. Technical Report NU-CCIS-07-03, Northeastern University (2007) <http://www.ccs.neu.edu/research/demeter/biblio/evergreen.html>.
6. Williamson, D.P.: Lecture notes on approximation algorithms. Technical Report RC 21409, IBM Research (1999)
7. 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.
8. Dechter, R.: Bucket elimination: a unifying framework for processing hard and soft constraints. *ACM Comput. Surv.* **28**(4es) (1996) 61
9. Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: The Logic Behind Weighted CSP. In Veloso, M.M., ed.: *IJCAI*. (2007) 32–37
10. Anbulagan, Slaney, J.: Multiple Preprocessing for Systematic SAT Solvers. In: *IWIL-6*, as part of *LPAR-2006*, Phnom Penh, Cambodia (2006)
11. 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)