Hyperplane Initialized Local Search for MAXSAT

Doug Hains and Darrell Whitley and Adele Howe and Wenxiang Chen
Department of Computer Science
Colorado State University
Fort Collins, CO, U.S.A. 80523
{dhains,whitley,howe,chenwx}@cs.colostate.edu

ABSTRACT

By converting the MAXSAT problem to Walsh polynomials, we can efficiently and exactly compute the hyperplane averages of fixed order k. We use this fact to construct initial solutions based on variable configurations that maximize the sampling of hyperplanes with good average evaluations. The Walsh coefficients can also be used to implement a constant time neighborhood update which is integral to a fast next descent local search for MAXSAT (and for all bounded pseudo-Boolean optimization problems.) We evaluate the effect of initializing local search with hyperplane averages on both the first local optima found by the search and the final solutions found after a fixed number of bit flips. Hyperplane initialization not only provides better evaluations, but also finds local optima closer to the globally optimal solution in fewer bit flips than search initialized with random solutions. A next descent search initialized with hyperplane averages is able to outperform several state-of-the art stochastic local search algorithms on both random and industrial instances of MAXSAT.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Theory, Algorithms

Keywords

Local Search, Initialization, MAXSAT

1. INTRODUCTION

The early theoretical analysis of genetic algorithms emphasized the potential for populations to explicitly estimate hyperplane averages and to use this information to guide search [9, 7]. While this line of research has been criticized [17], a similar idea is at the foundation of estimation of distribution algorithms: information about the interaction between variables can be used to guide search [14, 8].

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

For all k-bounded pseudo-Boolean optimization (PBO) problems, we can convert the evaluation functions into a polynomial form in $O(n)$ time. This allows us to quickly and exactly compute low order hyperplane averages. We can then explicitly determine which combination of variable assignments will lead to the highest overall combined hyperplane average. By using this information to initialize search we achieve two results: 1) search starts at a solution that *must* be better than average, reducing the number of steps needed to reach a local optimum, and 2) the initial solution is in a region of the search space that is also better than average relative to other regions. Thus, not only is the initial solution better than average, solutions that are nearby in Hamming space must also be better than average.

A wide range of important optimization problems are naturally expressed as k-bounded PBO problems. In computing, this includes hardware verification, combinatorial auctions, design debugging, software testing and graph coloring[13] as well as classic NP-hard problems such as MAXSAT, vertex cover, maximum cut, and maximum independent set [1]. In biology, NK-landscapes have been developed as a general model for interacting sets of components (alleles, proteins, amino-acids) with applications in RNA-folding [19, 18] and the study of viruses [5]. In physics, Ising spin glasses correspond to PBO problems [4].

In this paper, we focus on MAXSAT for three reasons. First, local search algorithms for MAXSAT have been widely studied for more than 20 years. Thus, improving local search algorithms for MAXSAT is very challenging. Second, there are many well studied and widely available benchmark problems as well as real world problems that have been reduced to MAXSAT problems. Third, other methods for initializing search have been studied for MAXSAT. In particular, other researchers have attempted to identify "backbone variables" as a way to initialize search [26].

We show how the Walsh transform can be used to efficiently calculate the hyperplane averages of MAXSAT instances and then describe a method of using these averages to construct a solution. This method consistently produces solutions with better evaluations than those constructed with a uniform random distribution, the standard practice for local search algorithms. We conjecture that this method also finds solutions closer to the backbone. We empirically test this conjecture on random instances of MAX-3SAT and find that our hyperplane construction algorithm sets variables in the backbone to their correct assignments in the majority of cases.

The calculation of hyperplane averages requires Walsh coefficients; we show how the coefficients can also be used to implement a constant time neighborhood update in local search. We use this update to develop a constant time next descent local search algorithm. This algorithm helps assess the effect of hyperplane initial-

ization on two factors: the first local optimum encountered and the final solution after n bit flips.

We find that the first local optima found by next descent search when initialized with hyperplane averages have better evaluations than those found by randomly initialized search. These results are consistent across both real world and random instances of MAXSAT. Furthermore, the first local optima are found more quickly and are closer to an optimal solution on an industrial MAXSAT problem.

To assess the longer term impact of the initialization, the algorithms are run for n bit flips, well past the first local optimum. Again, the solutions found by hyperplane initialized search are always significantly better than those found by randomly initialized search. When our Walsh-based next descent search algorithm is initialized with hyperplane averages, it outperforms several state-of-the-art stochastic local search algorithms for MAXSAT.

Finally, the speedup induced by exploiting Walsh coefficients is greater for industrial problems, as opposed to randomly generated problems. Industrial problems tend to have a very high rate of co-occurring variables due to the algorithms used to convert general satisfiability expressions into CNF-SAT. These co-occurrences translate into overlapping Walsh coefficients and more compact Walsh polynomials, which translates into faster updates per move.

2. THEORETICAL FOUNDATIONS

A discrete function $f : \{0,1\}^n \mapsto \mathbb{R}$ can be decomposed into an orthogonal basis

$$f(x) = \sum_{i=0}^{2^n-1} w_i \psi_i(x)$$

where w_i is a real-valued weight known as a *Walsh coefficient* and ψ_i is a *Walsh function*. The index i and vector x can be represented as binary strings, and standard binary operations can be applied. The Walsh function

$$\psi_i(x) = -1^{i^T x}(-1)^{bitcount(i \wedge x)}$$

generates a sign: if $i^T x$ is odd $\psi_i(x) = -1$ and if $i^T x$ is even $\psi_i(x) = 1$.

The MAXSAT objective function is given by

$$f(x) = \sum_{j=1}^{m} f_j(x, mask_j)$$

where each subfunction f_j corresponds to a clause and $mask_j$ selects the bits used by f_j. Since MAXSAT is a linear combination of subfunctions, we can apply the Walsh transform to each clause:

$$w = \sum_{j=1}^{m} \mathbf{W} f_j$$

where w is a vector of polynomial coefficients and $\mathbf{W}$ is a discrete Fourier transform known as the Walsh transform. This generates the Walsh coefficients associated with each clause, and then adds them together as needed. We will use the Walsh transform without normalization, since this results in all of the Walsh coefficients being integer values. Rana et al. [16] show that we can dispense with matrix $\mathbf{W}$ and directly compute the Walsh coefficient associated with each clause. Each subfunction f_j contributes at most 2^k nonzero Walsh coefficients to vector w.

A single Max-3SAT clause generates eight coefficients: the constant w_0, three linear coefficients and four nonlinear (3 pairwise and 1 order-3). Thus, over m clauses, there are at most $4m$ nonlinear coefficients.

We assume the Walsh coefficients and their signs have been computed for some initial solution x and will use b to index all of the Walsh coefficients in vector $w'(x)$. Let $p \subseteq b$ denote that every bit in string p that has value 1 must also have value 1 in string b. For the moment, assume p has only one bit set to value 1, and all others are 0. We can then compute the sum of all of the Walsh coefficients that are affected when local search flips bit p.

The Walsh function, $bitcount(i \wedge x)$, counts how many 1-bits are in the bit string produced by the logical operation $i \wedge x$. If the bit flip in string x does not interact with a bit in string i, it has no impact on Walsh coefficient contribution $\psi_i(x)w_i$. If the bit flip in x does interact with a bit in string i, it flips the sign generated by $\psi_i(x)$ since $bitcount$ changes by exactly 1.

To exploit this idea, the vector w' will store the Walsh coefficients; this will include the sign relative to the current solution x.

$$w'_i(x) = w_i \psi_i(x)$$

We next accumulate all of the Walsh coefficients that include a single bit p.

$$S_p(x) = \sum_{\forall b,\ p \subseteq b} w'_b(x) \tag{1}$$

When a flip occurs, only k Walsh coefficients per clause change. For example (indexing from right to left), assume the current solution (clause assignment) is $x = 000$ and we flip one bit such that $x_1 = 001$ then:

$$f(001) = f(000) - 2w'_1(x) - 2w'_3(x) - 2w'_7\psi_i(x)$$

$$f(001) = f(000) - 2S_1(x)$$

The indices 1, 3, and 7 act as bit masks that select variables. This leads to Lemma 1.

Lemma 1.
Let function $N(x)$ generate the set of neighbors of solution x. Let $y_p \in N(x)$ be the neighbor of string x generated by flipping bit p. Then $f(y_p) = f(x) - 2(S_p(x))$.

Whitley and Chen provide a detailed proof [25]. Assume bit p is flipped; we must update sums in S that include Walsh coefficients that are jointly indexed by p. Let b represent a bit string such that w'_b is a nonzero Walsh coefficient.

$$S_q(y_p) = S_q(x) - 2 \sum_{\forall b, (p \wedge q) \subseteq b} w'_b(x) \tag{2}$$

The vector $w'(x)$ must also be updated. The sign is flipped for those coefficients affected by flipping bit p.

$$\begin{array}{lll} \forall b, & if\ \ p \subseteq b, & w'_b(y_p) = -w'_b(x) \\ & otherwise & w'_b(y_p) = w'_b(x) \end{array}$$

For each clause that is affected, when 1 bit flips, there are $k-1$ other sums in S that must be updated. Assume kc clauses are affected: then there are at most $kc(k-1)$ updates to sums in S for each bit flip. If bit p is flipped we must also update $S_p(y_p) = -S(x)$. This yields a total of $kc(k-1)+1$ updates to vector S.

Whitley [24] translates this special case result into an O(1) bound on the average case computation. Whitley and Chen extend that result to cover NK-landscapes [25]. This proof depends on holding bits that appear in a large number of clauses tabu after they are

flipped to amortize cost; this can be particularly important on industrial MAXSAT problems.

If an element of S does not change, the bit flip cannot yield a new improving move. Therefore we can identify any new improving move in $O(1)$ time. For next descent, we simply maintain a list of improving moves. As improving moves are created or destroyed by updates to the vector S, the list is updated as well. Then, we can select any improving move randomly from the list.

Note that if $S_j(x)$ is negative, then flipping bit j must yield an improving move. Thus, $S_j(x)$ can be used as a proxy for $f(y_j)$ because $f(x)$ is constant as j is varied.

The detailed proof of $O(1)$ complexity per move is an amortized average case result [25], which is beyond the scope of the current paper. However, if every bit is flipped exactly once, the number of updates to w' is at most $ck2^{k-2}$. As a special case, when $k = 3$ we obtain $ck(k-1) = ck2^{k-2} = 6c$ which is independent of number of variables or clauses.

2.1 Compact Walsh Polynomials for MAXSAT

In the general case, the Walsh polynomials associated with random Boolean functions are exponentially large. However, this ignores the fact that most evaluation functions are decomposable.

There are two ways to obtain MAXSAT problems: they can be randomly generated, or they can be produced by reducing another problem to a MAXSAT problem. Assume we wish to reduce the following Boolean satisfiability problem (from Cormen et al. 2001) into a MAX-3SAT problem.

$$\phi = ((x_1 \longrightarrow x_2) \vee \neg((\neg x_1 \iff x_3) \vee \neg x_4)) \wedge x_2.$$

This would be converted to a binary "parse" tree (which might double the number of variables). An **intermediate form** is then produced which "and's" together expressions over triples of variables of the form $y_1 \iff y_2 \wedge \neg x_2$.

To be more concise, assume we just want to convert the Boolean expression $x_2 \wedge \neg x_3$ to MAX-3SAT and we use the **intermediate form** $x_1 \iff x_2 \wedge \neg x_3$ where x_1 is an introduced variable. Using a truth table (see Cormen et al. 2001) to extract the DNF formulas and then applying DeMorgan's laws to obtain the CNF formulas yields the following literal clauses.

$$\begin{aligned}(x_2 \wedge \neg x_3) \equiv\ & ((\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \\ & \wedge(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3))\end{aligned}$$

This reduction yields four clauses that reference exactly the same three variables. It is simple to prove by enumeration of cases that all "if and only if" expressions over three variables will yield exactly four literal clauses unless the expression reduces to a simpler form.

Why is this important? If four clauses share exactly the same variables x_1, x_2 and x_3 then the Walsh coefficients $w_{1,2}, w_{2,3}, w_{1,3}$ and $w_{1,2,3}$ will capture all of the nonlinear interactions from all four clauses simultaneously using just these four numbers. This, in effect, compresses the representation back to a size that reflects the underlying intermediate form.

Large uniform random MAXSAT problems have surprisingly few duplicate nonlinear interactions. The maximum number of nonlinear terms in MAX-3SAT is four. Randomly generated MAXSAT problems with 100,000 variables typically have 3.999 unique Walsh coefficients per clause; this number approaches 4.0 as n increases.

We analyzed 14 large industrial problems with variable k from the MAXSAT 2012 challenge (http://maxsat.ia.udl.cat/); the problems range in size from 247,943 to 2,766,036 variables. As shown in Table 1, these real world problems have a median of just 1.17 Walsh coefficients per clause (mem-ctrl problem), just slightly more than 1 integer number per clause. Some applications have less than one Walsh coefficient per clause (e.g., wb-46 and rsd-KES as the minimum).

	Maximum		Median	Minimum	
Instance	b15	fpu	mem-ctrl	wb-46	rsd-KES
WPC	1.69	1.26	1.17	0.97	0.73

Table 1: The number of Walsh coefficients Per Clause (WPC). We report the 2 maximum and 2 minimum WPC, as well as the median WPC over 14 industrial problems.

Real world industrial problems have fewer Walsh coefficients on average than random MAX-3SAT problems. As the update cost during search is a function of the number of Walsh coefficients, the update cost for industrial MAXSAT problems will be much faster than updates on uniform random problems.

2.2 Computing Hyperplane Averages

The search space of a MAXSAT instance with n variables and m clauses corresponds to a n-dimensional hypercube. If we 'fix' the truth values of j variables to 1 or 0, the search space is reduced to a $(n-j)$-dimensional hyperplane.

The Walsh coefficients can be used to efficiently compute the average evaluation of solutions contained in any $(n-j)$-dimensional hyperplane [16] [6]. Let h denote a $(n-j)$ dimensional hyperplane where j variables have preassigned bit values. Let $\alpha(h)$ be a mask with 1 bits marking the locations where the j variables appear in the problem encoding, and 0 bits elsewhere. Let solution x assign values to the j variables. Let $\beta(h) = \alpha(h) \wedge x$. This means $\beta(h)$ has value 0 in all of the positions where the j bits do not appear, and has the assigned values of the relevant j bits in the appropriate bit locations. Then the average fitness of hyperplane h is

$$Avg(h) = f_{\text{avg}} + \sum_{\forall b, b \subseteq \alpha(h)} w_b \psi_b(\beta(h))$$

where $f_{\text{avg}} = w_o$ is the average over the entire MAXSAT search space, i.e. $f_{\text{avg}} = (2^k - 1)/(2^k) * m$.

Although we can find the averages of any number of hyperplanes using this method, for the current study we compute the averages of the $2^k m$ hyperplanes that exactly correspond to the m clauses. For example, in a MAX-3SAT problem, there are seven assignments that can make a clause true. For each of these assignments, we can compute the hyperplane averages: this tells us how, on average, this assignment will impact the evaluation function over the remainder of the search space. Note that we only need to compute the hyperplane averages once. Thus, we not only have local information (whether the assignment makes a clause true or not), we also have global information about how the assignment affects the rest of the search space. The computational complexity to do this is $O(n)$ assuming $m = cn$ and c is a constant.

3. HYPERPLANE VOTING

We now describe a method of exploiting hyperplane averages to construct solutions to MAX-SAT that we call *hyperplane voting*. While we use MAX-3SAT to describe the method, hyperplane voting can be applied to any MAX-SAT problem.

In MAX-3SAT, given a clause v_i containing the variables p, q, and r, there are eight possible assignments of these variables: 000, 001, 010, 011, 100, 101, 110, and 111. We compute the averages of the eight hyperplanes formed by fixing p, q and r to each of these partial assignments and leaving the remaining variables free. This process is repeated for each clause in the instance. Thus we

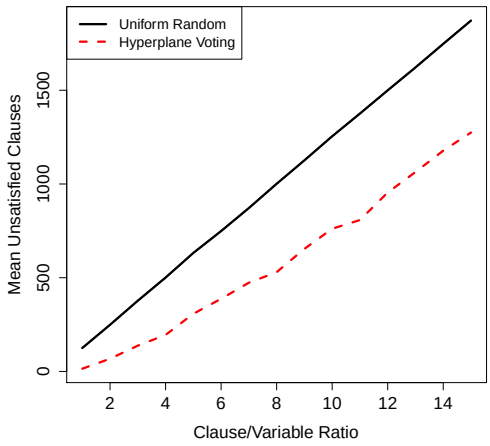

Figure 1: Mean number of unsatisfied clauses in randomly generated MAX-3SAT instances found by initializing solutions using a uniform random probability distribution and hyperplane voting. $n = 1000$; in expectation, there should be $1/8m$ unsatisfied clauses.

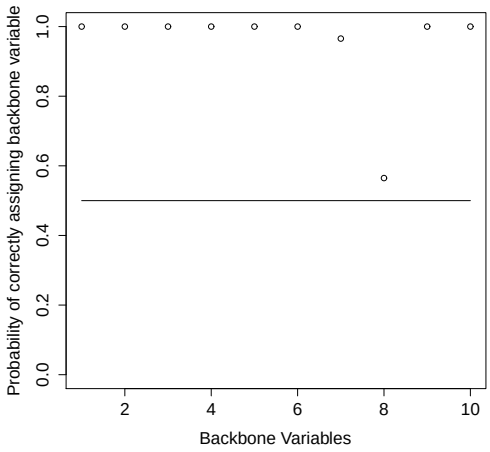

Figure 2: Probability of correctly setting backbone variables based on the hyperplane voting method of a randomly generated MAX-3SAT instance. The horizontal line marks a probability of .5; points over this line indicate a bias towards the correct assignment of a backbone variable.

compute eight hyperplane averages for each clause for a total of $8m$ hyperplane averages. We then use the hyperplane with the best average from each clause to calculate a probability distribution over all n variables as follows.

Let $v_i = \{p, q, r\}$ be the three variables in clause i. Let $A_i : v_i \mapsto \{0, 1\}$ be the partial assignment of the variables in clause i that correspond to the hyperplane with the highest average for clause v_i. Let true_j count the number of times that variable j is set to 1 across all partial assignments A and let total_j count the total number of times that variable j appears across all clauses (assignments):

$$\text{true}_j = \sum_{\forall i : j \in v_i} A_i(j)$$

$$\text{total}_\text{j} = true_j + \sum_{\forall i : j \in v_i} (1 \oplus A_i(j))$$

where $(1 \oplus A_i(j)) = 1$ when $A_i(j) = 0$. We define the following probability distribution over truth assignments based on the hyperplane voting:

$$P(j = 1) = \frac{\text{true}_j}{\text{total}_\text{j}}$$

Solutions are then constructed by generating a random value in the range of $(0, 1)$ for each variable. If the random value generated for variable j is greater than $P(j = 1)$, j is set to 1, otherwise j is set to 0.

We generated uniform random MAX-3SAT instances with $n = 1000$ and c/v varying from 1-15. 100 solutions were constructed using each of hyperplane voting and uniform random initialization. Figure 1 shows the mean unsatisfied clauses of the initial solutions constructed for each instance. Hyperplane voting significantly decreases the number of unsatisfied clauses in the initial tour.

3.1 Estimating Backbone Variables

We are aware of only two other initialization methods for MAXSAT that are able to improve over a uniform random solution, due to Zhang et al. [26] and Qasem et. al [15]. In both of these cases, local search must first be run multiple times with a uniform random initialization to construct a set of local optima. The frequency of assignments for each bit found in the set of local optima are then used to initialize subsequent runs of local search. It is hypothesized that these frequencies can provide a good estimation of the *backbone*, a subset of variables that are consistently assigned either true or false across all global optima [26, 15].

We conjecture that hyperplane voting is also able to produce a good estimate of the backbone. To determine how well our probability distribution estimates the backbone, we found the backbone to 10 randomly generated MAX-3SAT instances with $n = 20$ and $m = 85$ by enumerating the search space to find all optimal solutions. We then generated the probability distribution for initializing variables for each of these instances using the hyperplane voting method described above.

Figure 2 depicts the probability of assigning a backbone variable the correct truth assignment using hyperplane voting. Any point over .5 denotes a bias towards correctly assigning a variable to the backbone. We counted the number of correct assignments based on hyperplane voting on all instances and found that hyperplane voting biases the initial assignment towards the backbone in 121 of the 163 backbone variables found across these 10 instances. We can consider the probability distribution found by hyperplane voting as a prediction of the backbone variable assignments. The mean absolute error of these predictions against the correct assignments of the backbone variables averaged 0.27 with a standard deviation of .08 across the 10 instances. Hyperplane voting provides a remarkably good estimation of the backbone variables considering that no prior sampling of the space is required.

4. INITIALIZING LOCAL SEARCH

Hyperplane voting can improve the initial solution and provide an estimation of the backbone variables on random instances of MAX-3SAT, but how do the solutions produced by hyperplane voting influence subsequent search? We investigate the effect of hy-

perplane voting on two factors: the first local optimum encountered and the final solution after n bit flips. In these experiments, we have two sets of benchmark problems: random and industrial instances.

4.1 Random Instances

We generated random MAX-3SAT problems by assigning three variables to each of m clauses from n variables with equal probability. To examine the effect of problem size, we chose n =500,000, 1,000,000, 1,500,000 and 2,000,000. The number of clauses was set to $m = 4.27n$ as this is the standard value used in many SAT and MAXSAT studies [10]. The bottom rows in Table 2 show the exact n and m values of these problems as well as the mean evaluation of 50 solutions constructed by hyperplane voting and random initialization.

4.2 Industrial Benchmarks

To address the effectiveness on application problems, we selected a subset of the 52 industrial instances from the MAXSAT 2012 challenge. The number of variables (n) and number of clauses (m) are shown in the top rows of Table 2. These problems were used in a circuit-board testing method utilizing MAXSAT solvers [2, 20]. These problems therefore represent circuits converted to a Boolean formula in conjunctive normal form as previously described.

Instance	n	m	Hyperplane	Random
div-8	246943	810105	**90076**	146660
fpu	257168	928310	**87601**	159129
wb-problem-46	300846	789283	**97079**	165097
wb-conmax1	277950	1221020	**89124**	182860
c2-1	400085	1121810	**130552**	231442
i2c-25	521672	1581471	**193869**	309250
b15	581064	1712690	**277864**	368333
mrisc	844900	2905976	**233463**	509772
rsd-41	1186710	3829036	**292403**	685756
rsd-37	1513544	4909231	**382916**	876443
mem-ctrl2	1974822	6795573	**610751**	1182789
wb-4m8s-48	2766036	8774655	769608	1594813
3sat-.1m	100000	427000	**24343**	53416
3sat-.5m	500000	2135000	**122078**	266942
3sat-1m	1000000	4270000	**244557**	533762
3sat-1.5m	1500000	6405000	**367019**	800636
3sat-2m	2000000	8540000	**489144**	1067515

Table 2: The left most columns respectively indicate the instance, the number of variables n and number of clauses m found in selected industrial instances from the 2012 MAXSAT challenge (top) and in random instances (bottom). The right two columns are the mean number of unsatsified clauses in 50 solutions constructed by hyperplane voting and uniform random initialization, respectively. P-values of standardized T-tests between the evaluations were $< 1 \times 10^{-100}$ in all cases.

4.3 Effect on Local Optima

Does initializing local search with hyperplane voting improve the solutions found? We use the fast Walsh update to the S vector to implement a next descent search algorithm (see Algorithm 1) which we call *WalshSAT-N* (N=Next Improving Moves). If at least one improving move exists, it will be taken. If no improving moves exist, an equal move is taken; otherwise a random move is made.

We first examine the effect of initialization on the first local optimum found. We ran WalshSAT-N on both sets of instances until a local optimum was found. (When the improving move buffer is empty, the search halted, and steps 9-16 are not executed.) We ran two versions of WalshSAT-N: hyperplane voting and uniform random initialization for step 2. Each was executed in 50 trials recording both the evaluations of the local optima found and the number of bit flips required to reach a local optimum. The means and standard deviations of evaluations are shown in Table 3; the means and standard deviations of the number of bit flips to a local optimum are shown in Table 4. Welch paired T-tests confirm that the local optima found by hyperplane initialized search are significantly better than those found by randomly initialized search ($p < .000001$ in all tests).

Not only are the local optima found by hyperplane initialized search better than those found by random initialized search, but they are also found in less bit flips. Thus the initial solutions constructed by hyperplane voting are closer to better local optima than random solutions.

Although hyperplane initialization finds better solutions in less bit flips, this could be detrimental to the overall search if the local optima are further away from a *globally* optimal solution. Is hyperplane initialization guiding the search closer to globally optimal solutions or away from them? To answer this question, we calculated the Hamming distance of the local optima found by hyperplane initialized search to a known globally optimal solution on the i2c-25 industrial instance. The Hamming distances and the evaluation differences are shown in Figure 3 for instance i2c-25. The local optima found by hyperplane initialized search are closer to the global optima than those found by random initialized search. Thus, in this case hyperplane initialization does lead the search to better parts of the space.

4.4 Effect on Longer Runs

Of course stochastic local search algorithms are typically run beyond the first local optimum. To evaluate the effect of hyperplane voting on longer runs, we ran both our hyperplane and random initialized WalshSAT-N for n bit flips, where n is the number of variables in the instance. We did 50 runs per instance and recorded the evaluation of the final solution found by each run. To determine how well WalshSAT-N is doing relative to other algorithms, we also ran several algorithms from the UBCSAT search package [23]: GSAT, IRoTS, and AdaptG2WSAT.

GSAT [21] was chosen as it is a well-studied algorithm for SAT and MAXSAT that is very similar to our Walsh-based next descent. The main difference is that GSAT always takes the move that results

Algorithm 1: Next Descent Search using Walsh Polynomials

```
1  Compute Walsh coefficients; Generate initial solution x.
2  Initialize S(x).
3  Let ImprovingMoves = {i|S_i(x) < 0}.
4  Let EqualMoves = {i|S_i(x) = 0}.
5  while Termination Criteria not Met do
6      if ImprovingMoves ≠ ∅ then
7          Select j with uniform probability from
           ImprovingMoves.
8      else
9          if EqualMoves ≠ ∅ then
10             Select j with uniform probability from
               EqualMoves.
11         else
12             Select j with uniform probability from all
               possible moves.
13     Flip bit j in x.
14     Update S(x), ImprovingMoves and EqualMoves.
15     if j ∈ ImprovingMoves then
16         Remove j From ImprovingMoves
```

Initialization	Hyperplane	Random
div-8	**23303 ± 323**	34057 ± 337
fpu	**25378 ± 161**	36148 ± 158
wb-46	**35263 ± 165**	44543 ± 158
wb-conmax1	**23905 ± 226**	41334 ± 205
c2-1	**32019 ± 153**	56237 ± 284
i2c-25	**49448 ± 320**	75279 ± 329
b15	**60289 ± 252**	74245 ± 216
mrisc	**46095 ± 1447**	118353 ± 1172
rsd-41	**110032 ± 284**	191260 ± 388
rsd-37	**140915 ± 422**	244059 ± 412
mem-ctrl2	**143141 ± 717**	254937 ± 752
wb-4m8s-48	**261290 ± 437**	423034 ± 538
3sat-.1m	**7179 ± 66**	10379 ± 72
3sat-.5m	**35969 ± 126**	51974 ± 171
3sat-1m	**72101 ± 214**	104206 ± 328
3sat-1.5m	**108148 ± 213**	156103 ± 356
3sat-2m	**143891 ± 253**	208178 ± 386

Table 3: Mean and standard deviations of the evaluation of 10 local optima found by WalshSAT-N. Industrial benchmarks are on the top and random are on the bottom. P-values from t-tests comparing the means of both methods were $< 1 \times 10^{-100}$ in all cases.

Initialization	Hyperplane	Random
div-8	**47397 ± 130**	70286 ± 466
fpu	**44939 ± 204**	76344 ± 229
wb-46	**41872 ± 204**	77691 ± 221
wb-conmax1	**43333 ± 201**	87359 ± 272
c2-1	**68563 ± 161**	109088 ± 516
i2c-25	**97916 ± 523**	151806 ± 494
b15	**138701 ± 274**	182851 ± 364
mrisc	**108900 ± 208**	248026 ± 1429
rsd-41	**120934 ± 604**	296584 ± 530
rsd-37	**158945 ± 724**	378430 ± 712
mem-ctrl2	**313428 ± 1018**	591649 ± 1209
wb-4m8s-48	**361978 ± 519**	730023 ± 644
3sat-.1m	**14143 ± 87**	29358 ± 164
3sat-.5m	**70913 ± 180**	146720 ± 346
3sat-1m	**142196 ± 290**	293265 ± 417
3sat-1.5m	**213439 ± 381**	440030 ± 590
3sat-2.0m	**284533 ± 409**	586619 ± 720

Table 4: Mean and standard deviations of the number of bit flips to a local optimum by next descent search using both random and hyperplane initialization. The p-values from t-tests comparing the means of the two methods were $< 1 \times 10^{-100}$.

in the *best* change to the evaluation, whether it be improving or disimproving. Like GSAT, our Walsh-based next descent will take an improving move if one exists, otherwise it will take an equal move and in the last case it will select a disimproving move. Unlike GSAT, our algorithm randomly selects from each of these cases and is not guaranteed to always take the best move.

IRoTS, or Iterated Robust Tabu Search [22], was chosen as it was the best performing incomplete solver in the MAXSAT 2012 competition. It will always take the move that satisfies the largest number of unsatisfied clauses with respect to the current solution, given that the move is not Tabu. IRoTS also incorporates a perturbation stage if a new best improving move has not been found for some number of bit flips.

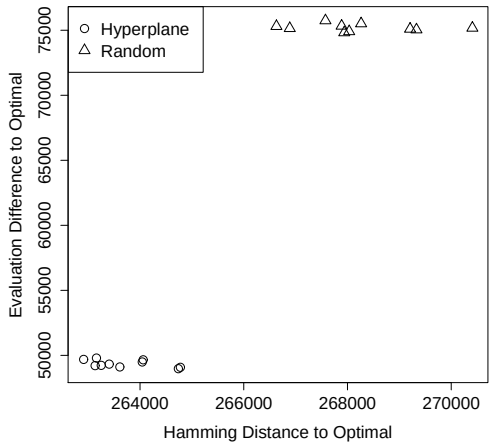

Figure 3: The Hamming distances and evaluation differences to a known globally optimal solution from local optima found by WalshSAT-N on the i2c-25 instance. 10 runs were initialized with hyperplane voting, and 10 runs were initialized with uniform random solutions.

AdaptG2WSAT [12] was chosen as it has been shown to be the best performing algorithm in UBCSAT on industrial instances [11]. It has performed exceptionally well in several SAT competitions [12]. AdaptG2WSAT works in two phases. The first phase is a steepest descent search that lasts until a local optimum is reached. The second phase uses a heuristic based on the Novelty family of algorithms to choose the bit to flip, although the steepest descent search is invoked if improving moves are found.

We ran the UBCSAT algorithms, using the default parameters, for the same number of bit flips and runs as WalshSAT-N. The mean and standard deviations of the final solutions are shown in Table 5.

As our Walsh-based search algorithm is very similar to GSAT, we expected that it would find similar quality solutions when initialized with random solutions and that it would outperform GSAT when initialized with hyperplane voting. Table 5 confirms this. Surprisingly, we see that our simple Walsh-based next descent method outperforms both IRoTS and AdaptG2WSAT when initialized with hyperplane voting. This result highlights the power of hyperplane initialization: our search algorithm is able to outperform more sophisticated stochastic local search algorithms when leveraging hyperplane information.

The last issue is cost: how much time is required to execute n bit flips by each of the tested algorithms? Table 6 shows means and standard deviations of the time in seconds of each of the 10 runs for each algorithm in the right-hand columns. The left column of Table 6 times shows the initialization time required to Walsh coefficients and hyperplane averages. The median time required to compute the coefficients on industrial instances is 4.06 seconds. Not only does hyperplane initialized WalshSAT-N find better solutions in most cases, but it is generally faster than the algorithms in UBCSAT, in some cases by an order of magnitude or more.

Furthermore, there is the hidden cost of converting industrial problems, such as the circuit board debugging problems used here, from their original SAT formulation to CNF-SAT. The Walsh coefficients can be more efficiently computed from the intermediate form than from the CNF-SAT form, making it unnecessary to convert to CNF-SAT.

Instance	WalshSAT-N (HP)	WalshSAT-N (RND)	GSAT	ADAPTG2WSAT	IROTS
div-8	5467 ± 197	13761 ± 196	13066 ± 187	10795 ± 93	12314 ± 481
fpu	10917 ± 77	13044 ± 108	12838 ± 125	10247 ± 111	11022 ± 136
wb-problem-46	8532 ± 168	13379 ± 185	13178 ± 164	8794 ± 131	10939 ± 532
wb-conmax1	11554 ± 160	19130 ± 210	17071 ± 173	12212 ± 127	13537 ± 120
c2-1	12819 ± 80	19524 ± 148	19595 ± 116	16056 ± 251	13473 ± 90
i2c-25	12146 ± 131	21586 ± 209	20352 ± 193	18288 ± 318	21951 ± 145
b15	24517 ± 149	30803 ± 172	31509 ± 149	27920 ± 176	30604 ± 153
mrisc	6435 ± 258	40851 ± 690	39628 ± 588	34301 ± 768	41244 ± 114
rsd-41	17256 ± 138	72363 ± 612	68708 ± 494	50298 ± 228	55212 ± 271
rsd-37	22976 ± 167	92361 ± 688	87911 ± 533	64162 ± 355	70568 ± 325
mem-ctrl2	29649 ± 368	75729 ± 714	73071 ± 584	38277 ± 535	58833 ± 533
wb-4m8s-48	87829 ± 698	151520 ± 407	148248 ± 372	122306 ± 375	122962 ± 475
3sat-.1m	2912 ± 35	4124 ± 49	4034 ± 52	3052 ± 61	2959 ± 50
3sat-.5m	14605 ± 70	20718 ± 120	20210 ± 107	15396 ± 153	14875 ± 77
3sat-1m	29249 ± 125	41511 ± 164	40418 ± 165	30856 ± 134	29638 ± 146
3sat-1.5m	43951 ± 156	62218 ± 187	60515 ± 203	46180 ± 209	44349 ± 141
3sat-2m	58415 ± 186	82898 ± 225	80696 ± 220	61466 ± 182	59123 ± 178

Table 5: Means and standard deviations of evaluations of solutions found after n bit flips by several algorithms. Our Walsh based algorithm (left two columns) was initialized using hyperplane averages (HP) and random solutions (RND).

It should also be noted that a fast descent search with constant time updates could also be applied to the UBCSAT algorithms to make them faster during the first phase. However, the second phase still accounts for the vast majority of these runs. Although it is likely possible to further optimize the second stage, it would require a non-trivial amount of engineering to optimize the algorithms and it is not clear if this could be accomplished without significantly changing their behavior.

5. CONCLUSIONS

We have investigated the use of hyperplane averages as a means to initialize stochastic local search for MAXSAT. This method uses configurations of variables that correspond to hyperplanes with good averages to construct a probability distribution over all variables.

Solutions constructed by hyperplane initialization have better evaluations than those found by randomly assigning solutions. For small enumerable instances, these initial solutions were shown to have variable assignments consistent with the correct assignment for backbone variables in the majority of cases.

We evaluated the use of hyperplane initialization on local search on both industrial and random instances of MAXSAT. We examined two factors of the search: the first local optimum encountered by the search and the final solution after n bit flips.

We found that the first local optima found by next descent search when initialized by hyperplane averages had better evaluations than those found by search initialized with random solutions. Furthermore, the hyperplane initialized search required less bit flips to find the first local optimum. Comparing the Hamming distance of local optima to a globally optimal solution on industrial instances, we found that search initialized with hyperplane averages found local optima closer to the global optimum.

Finally, we ran our search using both initializations on the instances along with several algorithms from UBCSAT. WalshSAT-N with hyperplane initialization consistently found better solutions than search initialized with random solutions. Surprisingly WalshSAT-N was able to outperform the UBCSAT algorithms in the vast majority of cases. This includes AdaptG2WSAT which was shown to be the best performing algorithm in UBCSAT on industrial instances [11]. WalshSAT-N was also much faster than these algorithms, but this is largely due to the use of next-improving moves instead of greedy best-improving moves used by AdaptG2WSAT and GSAT.

This work has demonstrated the effectiveness of using hyperplane averages to initialize the search. Although our results are on MAXSAT, this method can be applied to any pseudo-boolean function, such as NK-landscapes or spin glass problems. Our future work will explore other methods of guiding local search based on the variable interaction information contained within the Walsh coefficients.

6. ACKNOWLEDGMENTS

This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0088. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. This research utilized the CSU ISTeC Cray HPC System supported by NSF Grant CNS-0923386.

7. REFERENCES

[1] E. Boros and P.L. Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1):155–225, 2002.

[2] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris. Automated design debugging with maximum satisfiability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(11):1804–1817, 2010.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

[4] C. DeSimone, M. Diehl, M. Juenger, P. Mutzel, G. Reinelt, and G. Rinaldi. *Exact Ground States of Two-Dimensional J Ising Spin Glasses*. Max-Planck-Inst. für Informatik, Bibliothek & Dokumentation, 1996.

[5] S.F. Elena, R.V. Solé, J. Sardanyés, et al. Simple genomes, complex interactions: Epistasis in RNA virus. *Chaos*, 20, 2010.

[6] D. Goldberg. Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction. *Complex Systems*, 3:129–152, 1989.

Instance	Init	WalshSAT-N (HP)	WalshSAT-N (RND)	GSAT	ADAPTG2WSAT	IROTS
div-8	4.1	0.89 ± 0.03	0.79 ± 0.02	9.29 ± 0.15	15.39 ± 0.11	223.02 ± 1.49
fpu	3.04	1.06 ± 0.02	0.93 ± 0.02	9.28 ± 0.09	16.72 ± 0.19	256.14 ± 0.56
wb-problem-46	1.05	0.83 ± 0.01	0.72 ± 0.01	16.49 ± 0.13	19.56 ± 0.18	330.99 ± 0.84
wb-conmax1	3.94	1.33 ± 0.01	1.15 ± 0.01	8.5 ± 0.12	22.12 ± 0.32	302.75 ± 1.16
c2-1	2.36	1.49 ± 0.02	1.42 ± 0.02	28.06 ± 0.16	36.14 ± 0.36	591.01 ± 0.73
i2c-25	2.51	1.86 ± 0.05	1.6 ± 0.03	49.29 ± 0.37	68.35 ± 0.93	1060.14 ± 2.58
b15	2.49	3 ± 0.05	2.74 ± 0.05	43.3 ± 0.29	92.57 ± 0.97	1276.63 ± 2.34
mrisc	19.72	2.96 ± 0.03	2.74 ± 0.04	107.18 ± 2.03	186.58 ± 2.27	2655.03 ± 48.91
rsd-41	12.96	4.33 ± 0.04	3.68 ± 0.04	236.4 ± 1.97	330.39 ± 5.14	5264.28 ± 16.95
rsd-37	17.25	5.57 ± 0.03	4.74 ± 0.03	388.23 ± 2.83	590.85 ± 6.71	8644.43 ± 9.58
mem-ctrl2	46.24	6.11 ± 0.48	5.71 ± 0.06	616.32 ± 4.25	1293.1 ± 9.19	13804.98 ± 70.77
wb-4m8s-48	17.39	10.06 ± 0.3	8.57 ± 0.22	1282.47 ± 6.4	2563.23 ± 33.95	29413.65 ± 56.36
3sat-.1m	2.16	0.79 ± 0.02	1.03 ± 0.03	2.13 ± 0.02	2.27 ± 0.03	39.85 ± 0.09
3sat-.5m	11.42	6.83 ± 0.37	4.99 ± 0.09	45.45 ± 0.26	51.94 ± 0.92	987.82 ± 1.04
3sat-1m	23.35	12.06 ± 1.68	13.18 ± 1.03	184.18 ± 3.47	216.62 ± 2.92	4002.1 ± 11.81
3sat-1.5m	36.19	16.48 ± 0.21	20.75 ± 0.5	408.69 ± 1.43	567.89 ± 6.49	9245.38 ± 52.67
3sat-2m	48.57	22.4 ± 0.22	26.45 ± 2.97	735.17 ± 4.49	1132.09 ± 7.91	16407.38 ± 17.97

Table 6: The left column shows the average time to compute the Walsh coefficients and hyperplane averages required by WalshSAT-N. The remaining columns report the mean and standard deviation of time in seconds to execute n bit flips for the various algorithms.

[7] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[8] D. Goldberg, B. Korb, and K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 4:415–444, 1989.

[9] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[10] H. Hoos, K. Smyth, and T. Stützle. Search space features underlying the performance of stochastic local search algorithms for MAX-SAT. In *Proc. Parallel Problem Solving from Nature (PPSN VIII)*, pages 51–60, 2004.

[11] L. Kroc, A. Sabharwal, C.P. Gomes, and B. Selman. Integrating systematic and local search paradigms: A new strategy for maxsat. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 544–551, 2009.

[12] C. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for sat. *Theory and Applications of Satisfiability Testing–SAT 2007*, pages 121–133, 2007.

[13] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proceedings of the conference on design, automation and test in Europe*, pages 408–413. ACM, 2008.

[14] M. Pelikan, D. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic model. Technical Report 99018, IlliGAL, Septemper 1999.

[15] M. Qasem and A. Prugel-Bennett. Learning the large-scale structure of the MAX-SAT landscape using populations. *IEEE Transactions on Evolutionary Computation*, 14(4):518–529, 2010.

[16] S. Rana, R.B. Heckendorn, and D. Whitley. A tractable walsh analysis of SAT and its implications for genetic algorithms. In *Proceedings of the National Conference on Artificial Intelligence*, pages 392–397, 1998.

[17] C. R. Reeves and J. E. Rowe. Landscapes. In *Genetic Algorithms – Principles and Perspectives: A guide to GA theory*, pages 231–263. Springer, 2002.

[18] C.M. Reidys and P.F. Stadler. Combinatorial landscapes. *SIAM Review*, 44:3–54, 2002.

[19] C.M. Reidys, P.F. Stadler, and P.K. Schuster. Generic properties of combinatory maps and neutral networks of RNA secondary structures. *Bull. Math. Biol.*, 59:339–397, 1997.

[20] S. Safarpour, H. Mangassarian, A. Veneris, M.H. Liffiton, and K.A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer Aided Design, 2007. FMCAD'07*, pages 13–19. IEEE, 2007.

[21] Bart Selman, Hector Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, San Jose, CA, 1992.

[22] K. Smyth, H.H. Hoos, and T. Stützle. Iterated robust tabu search for MAX-SAT. In *In Proc. of the 16th Conf. of the Canadian Society for Computational Studies of Intelligence*, pages 129–144, 2003.

[23] D.A.D. Tompkins and H.H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In Holger H. Hoos and David G. Mitchell, editors, *Theory and Applications of Satisfiability Testing: Revised Selected Papers of the Seventh International Conference (SAT 2004, Vancouver, BC, Canada, May 10–13, 2004)*, volume 3542 of *Lecture Notes in Computer Science*, pages 306–320, Berlin, Germany, 2005. Springer Verlag.

[24] D. Whitley. Defying gravity: constant time steepest ascent for MAX-kSAT. Technical report, Department of Computer Science, Colorado State University, December 2011.

[25] D. Whitley and W. Chen. Constant Time Steepest Ascent Local Search with Statistical Lookahead for NK-Landscapes. In *GECCO '12: Proc. of the annual conference on Genetic and Evolutionary Computation Conference*, 2012.

[26] W. Zhang, A. Rangan, and M. Looks. Backbone guided local search for maximum satisfiability. In *Proc. International Joint Conference on Artificial Intelligence*, volume 18, pages 1179–1186, 2003.