Automated Discovery of Composite SAT Variable-Selection Heuristics

Alex Fukunaga
Computer Science Department
University of California, Los Angeles
fukunaga@cs.ucla.edu

Abstract

Variants of GSAT and Walksat are among the most successful SAT local search algorithms. We show that several well-known SAT local search algorithms are the result of novel combininations of a set of variable selection primitives. We describe CLASS, an automated heuristic discovery system which generates new, effective variable selection heuristic functions using a simple composition operator. New heuristics discovered by CLASS are shown to be competitive with the best Walksat variants, including Novelty+ and R-Novelty+. We also analyze the local search behavior of the learned heuristics using the depth, mobility, and coverage metrics recently proposed by Schuurmans and Southey.

1 Introduction

Local search procedures for satisfiability testing (SAT) have been widely studied since the introduction of GSAT (Selman, Levesque, & Mitchell 1992). It has been shown that for many problem classes, incomplete local search procedures can quickly find solutions (satisfying assignments) to satisfiable CNF formula. Local search heuristics have improved dramatically since the original GSAT algorithm. Some of the most significant improvements have been the result of developing a new variable selection heuristic for the standard GSAT local search framework. These include: GSAT with Random Walk (Selman & Kautz 1993), Walksat (Selman, Kautz, & Cohen 1994), Novelty/R-Novelty (McAllester, Selman, & Kautz 1997), and Novelty+/R-Novelty+ (Hoos & Stutzle 2000).

In this paper, we consider how new, effective variable selection heuristics could be automatically discovered. First, we review the known variable selection heuristics, and identify some common structural elements. We then formulate the problem of designing a variable selection heuristic as a meta-level optimization problem, where the task is to combine various "interesting" variable-selection primitives into an effective composite heuristic function. We describe CLASS, a system that searches for good SAT variable selection heuristics. CLASS is shown to successfully generate a new variable selections heuristic which are competitive with the best known GSAT/Walksat-based algorithms.


```
T:= randomly generated truth assignment
For j:= 1 to cutoff
 If T satisfies formula then return T
 V:= Choose a variable using some
   variable selection heuristic
 T':=T with value of V reversed
Return failure (no satisfying
  assignment found)
```

Figure 1: SAT local search algorithm template

2 Common Structures in Composite Variable Selection Heuristics

Many of the standard SAT local search procedures can be succinctly described as the template of Figure 1 with a particular variable selection heuristic.

We now introduce some terminology to facilitate the discussion of the common structural elements of Walksat-family SAT variable selection heuristics throughout this paper.

Definition 1 (Positive/Negative/Net Gain) *Given a candidate variable assignment T for a CNF formula F, let B_0 be the total number of clauses that are currently unsatisfied in F. Let T' be the state of F if variable V is flipped.*

Let B_1 be the total number of clauses which would be unsatisfied in T'. The net gain *of V is $B_1 - B_0$. The* negative gain *of V is the number of clauses which are currently satisfied in T, but will become unsatisfied in T' if V is flipped. The positive gain of V is the number of clauses which are currently unsatisfied in T, but will become satisfied in T' if V is flipped.*

Definition 2 (Variable Age) *The* age *of a variable is the number of flips since it was last flipped.*

The standard heuristics we refer to in the rest of this paper are the following:

GSAT (Selman, Levesque, & Mitchell 1992): Select variable with highest net gain.

HSAT (Gent & Walsh 1993b) Same GSAT, break ties in favor of maximum age variable.

GWSAT (Selman & Kautz 1993): With probability p, select a variable in a randomly unsatisfied (broken) clause; otherwise same as GSAT.

Walksat (Selman, Kautz, & Cohen 1994): Pick random broken clause BC from F. If any variable in BC has a negative gain of 0, then randomly select one of these to flip. Otherwise, with probability p, select a random variable from BC to flip, and with probability $(1-p)$, select the variable in BC with minimal negative gain (breaking ties randomly).

Novelty (McAllester, Selman, & Kautz 1997): Pick random unsatisfied clause BC. Select the variable v in BC with maximal net gain, unless v has the minimal age in BC. In the latter case, select v with probability $(1-p)$; otherwise, flip v_2 with second highest net gain.

Novelty+ (Hoos & Stutzle 2000): Same as Novelty, but after BC is selected, with probability p_w, select random variable in BC; otherwise continue with Novelty.

R-Novelty (McAllester, Selman, & Kautz 1997) and **R-Novelty+** (Hoos & Stutzle 2000) are similar to Novelty and Novelty+, but more complicated (we omit their description due to space constraints).

Based on the descriptions above, it is clear that these heuristics share some significant structural (syntactic) features:

2.1 Common Primitives

All of the above heuristics combine a number of "selection heuristic primitives" into a single decision procedure. These conceptual primitives are the following:

Scoring of variables with respect to a gain metric: Variables are scored with respect to net gain or negative gain. Walksat uses negative gain, while GSAT and the Novelty variants use net gain.

Restricting the domain of variables: Whereas GSAT allows the selection of any variable in the formula, Walksat and Novelty variants restrict the variable selection to a single, randomly selected unsatisfied clause.

Ranking of variables and greediness: The variables in the domain are ranked with respect to the scoring metric. Of particular significance is the best (greedy) variable, which is considered by all of the heuristics. Novelty also considers the second best variable.

Variable age: The age of a variable is the number of flips since a variable was last flipped. This historical information seems to be useful for avoiding cycles and forcing exploration of the search space. Age used by the Novelty variants, as well as HSAT. Walksat with a tabu list was also evaluated in (McAllester, Selman, & Kautz 1997).

Conditional branching: In most of the heuristics, some simple Boolean condition (either a random coin toss or a function of one or more of the primitives listed above) is evaluated as the basis for a branch in the decision process.

Compact, nonobvious combinatorial structure: All of the heuristics can be implemented as relatively simple functions built by composing the various primitives discussed above. Even R-Novelty, which is the most complex of the above heuristics, can be represented as a 3-level decision diagram (Hoos & Stutzle 2000).

For all except possibly the simplest GSAT variants, it is difficult to determine a priori how effective any given heuristic is. Empirical evaluation is necessary to evaluate complex heuristics. For example, although there are many possible heuristics that combine the elements of Walksat (random walk, some greediness, localization of the variable domain to a single randomly selected broken clause), the performance of Walksat-variants varies significantly depending on the particular choice and structural organization of these "Walksat elements". Furthermore, significant performance differences between superficially similar local search heuristics can not be eliminated by merely tuning control parameters. See, for example, the comparison of Walksat/G, Walksat/B, and Walksat/SKC in (McAllester, Selman, & Kautz 1997).

3 CLASS: A System for Discovering Composite Variable Selection Heuristics

How can we discover new, effective variable selection heuristics? Some existing heuristics were a result of a focused design process, which specifically addressed a weakness in an existing heuristic. GWSAT and Novelty+ added random walk to GSAT and Novelty after observing behavioral deficiencies of the predecessors (Selman & Kautz 1993; Hoos & Stutzle 2000). However, some major structural innovations involve considerable exploratory empirical effort. For example, (McAllester, Selman, & Kautz 1997) notes that over 50 variants of Walksat were evaluated in their study (which introduced Novelty and R-Novelty).

It appears that although human researchers can readily identify interesting primitives (Section 2) that are relevant to variable selection the task of combining of these primitives into composite variable selection heuristics may benefit from automation. We therefore developed a system for automatically discovering new SAT heuristics, **CLASS** (**C**omposite heuristic **L**earning **A**lgorithm for **S**AT **S**earch). The main components of CLASS are:

- A minimal language for expressing variable selection heuristics s-expressions, and
- A population-based search algorithm that searches the space of possible selection heuristics by repeated application of a composition operator.

CLASS represents variable selection heuristics in a Lisp-like s-expressions language. In each iteration of the local search (Figure 1), the s-expression is evaluated in place of a hand-coded variable selection heuristic. To illustrate the primitives built into CLASS, Figure 2 shows some standard heuristics represented as CLASS s-expressions (See Appendix and Section 3.1 for language primitive definitions).

The space of possible s-expressions expressible in our language is obviously enormous, even if we bound the size of the expressions. Furthermore, we currently lack principled, analytical meta-heuristics that can be used to guide a systematic meta-level search algorithm. Therefore, CLASS uses a population-based search algorithm to search for good variable selection heuristics (Figure 3).

The `Initialize` function creates a population of randomly generated s-expressions. The expressions are gener-

GSAT with Random Walk (GWSAT):

```
(If (rand 0.5)
 RandomVarBC0
 VarBestNetGainWFF)
```

Walksat:

```
(IfVarCond == NegGain 0
 VarBestNegativeGainBC0
 (If (rand 0.5)
  VarBestNegativeGainBC0
 RandomVarBC0))
```

Novelty:

```
(IfNotMinAge BC0
 VarBestNetGainBC0
 (If (rand 0.5)
  VarBestNetGainBC0
  VarSecondBestNetGainBC0))
```

Figure 2: Walksat, GSAT with Random Walk, and Novelty represented in the CLASS language.

```
Initialize(population,populationsize)
For I = 1 to MaxIterations
Pick parent1 and parent2 from population
 Children = Composition(parent1,parent2)
 Evaluate(Children)
 Insert(Children,Population)
```

Figure 3: CLASS Meta-Search Algorithm

ated using a context-free grammar as a constraint, so that each s-expression is guaranteed to be a syntactically valid heuristic that returns a variable index when evaluated. The `Pick` function picks two s-expressions from the population, where the probability of selecting a particular s-expression is a function of its rank in the population according to its objective function score (the higher an expression is ranked, the more likely it is to be selected). The `composition` operator (detailed below) is applied to the parents to generate a set of children, which are then inserted into the population. Each child replaces a randomly selected member of the population, so that the population remains constant in size (the lower the ranking, the more likely it is that an s-expression will be replaced by a child). The best heuristic found during the course of the search is returned.

3.1 The composition operator

Recall that GWSAT and Novelty+ were derived by adding random walk to GSAT and Novelty. This can be generalized into a general meta-heuristic for creating new variable selection strategies: Given two heuristics H_1 and H_2, combine the two into a new heuristic that chooses between H_1 and H_2 using the schema:

`If Condition` H_1 `else` H_2

where Condition is a Boolean expression.

We call this the *composition* operator. Intuitively, this is a reasonable meta-heuristic because it "blends" (switches between) the behavior of H_1 and H_2 according to some boolean condition. The special case where Condition is a randomization function (i.e., `If (rnd<`p`) then` ...) is a *probabilistic composition*.

Probabilistic composition has a desirable theoretical property. Hoos (Hoos 1998) defines a SAT local search procedure to be PAC (approximately correct) if with increasing run-time the probability of finding a solution for a satisfiable instance approaches one. An algorithm that is not PAC is called essentially incomplete. GSAT, Novelty, and R-Novelty were shown to be essentially incomplete; however, their performance is significantly improved by adding random walk, which was proven to make these algorithms PAC (Hoos 1998). That is, the historical process by which GWSAT and Novelty+ were generated can be modeled as an application of probabilistic composition. The composition operator has the generalization of this formal property:

Property 1 *Let H_1 and H_2 be two variable selection heuristics. If (without loss of generality) H_1 is PAC, then the composite heuristic* `(If (rnd` p`) then` H_1 `else` H_2`)`*, which is the result of applying the probabilistic composition operator is also PAC for all $p > 0$. [Proof: follows from the fact that as long as $p > 0$, there is a sequence of coin flips which continues to choose H_1]*

Thus, during the discovery process, if we have a heuristic whose major deficiency is essential incompleteness, then probabilistic composition with any PAC heuristic in the population theoretically removes that deficiency.

The full composition operator used by CLASS takes two heuristics s-expressions H_1 and H_2 as input and outputs the 10 new heuristics to be inserted into the population:

- Five probabilistic compositions of the form `(If (rnd p)` H_1 H_2`)`, for p=0.1, p=0.25, p=0.5, p=0.75, and p=0.9
- `(OlderVar` H_1 H_2`)` - evaluates H_1 and H_2, and returns the variable with maximal age.
- `(IfTabu 5` H_1 H_2`)` - Let variable v be the result of H_1. If $age(v)$ is tabu (i.e., less than 5), then evaluate H_2.
- `(IfVarCond == NegativeGain 0` H_1 H_2`)` - Let v_1 be the result of H_1. if $NegativeGain(v_1) = 0$ return v_1, else return v_2, the result of H_2.
- `(IfVarCompare <= NegGain` H_1 H_2`)` - Let v_1 be the results of H_1, v_2 the result of H_2. If $NegativeGain(v_1)$ is less than or equal to $NegativeGain(v_2)$, then return v_1, else v_2.
- `(IfVarCompare <= NetGain` H_1 H_2`)` - same as above, but uses net gain as the comparator.

3.2 Evaluating the utility of a candidate heuristic

The `Evaluate` function in Figure 3 evaluates the utility of a candidate s-expression on a set of training instances. We selected the class of hard, randomly generated 3-SAT problem instances (Mitchell, Selman, & Levesque 1992) as our training set. First, the heuristic was run on 200 satisfiable, 50 variable, 215-clause random 3-SAT instances, with a cutoff of 500 flips. If more than 130 of these descents

```
(If (rand 0.5)
 (If (rand 0.25)
  (IfVarCompare > NetGain
   VarBestNetGainBC0
   (OlderVar VarBestNegativeGainBC1
    VarBestNegativeGainBC0))
  (OlderVar (OlderVar
   VarBestNegativeGainBC0
   (IfVarCond == NegativeGain 0
    VarBestNegativeGainBC0
    RandVarBC0))
   VarBestNegativeGainBC1))
 (If (rand 0.1)
  (If (rand 0.5) VarBestNetGainWFF
   VarRandomWFF)
  (OlderVar VarBestNegativeGainBC1
   VarBestNegativeGainBC0)))
```

Figure 4: CH1, a heuristic learned by CLASS

was successful, then the heuristic was run for 2000 satisfiable, 100-variable, 430-clause random 3-SAT instances with a 4000 flip cutoff. The score of an individual is: (# of 50-var successes) + (5 * (# of 100-var successes)) + ($1/MeanFlipsInSuccessfulRuns$)

Although the purpose of this scoring function design is to train heuristics on 100-variable problems, the 50-variable problems serve as a filter that quickly identifies very poor individuals and saves us from evaluating the large set of 100-variable problems. Nevertheless, this is a relatively expensive objective function, which can require up to a minute of computation for some individuals on a 500-Mhz Pentium III machine (largely due to inefficiencies in our implementation). We have experimented with using only smaller problem instances in the objective function, but in preliminary studies, we found that heuristics generated using only 25 and 50-variable instance training sets did not scale when executed on 100-variable problem instances (this is because on extremely small problems, it is difficult to distinguish the performance between mediocre heuristics and good heuristics, so the discovery algorithm receives insufficient bias).

The heuristic CH1 (Figure 4) was the best heuristic discovered in a CLASS run using a population of 300, after 3000 candidate expressions were generated and evaluated. Another heuristic, CH2 (not shown due to space constraints) was discovered in a run with population 400 and 5000 evaluations. Both CH1 and CH2 will be empirically evaluated below. Heuristics of similar quality can be reliably generated in a 5000 evaluation run. However, since each run of CLASS currently takes several days on a Pentium III-500MHz machine, we have not yet had the resources to perform a statistically meaningful study of learning algorithm performance.

3.3 CLASS-L: Extensions to the algorithm

In order to improve the quality of the heuristics discovered by CLASS, we introduced two enhancements, resulting in the CLASS-L system. Rather than relying on a completely random initial population to serve as the building blocks of

```
(If (rand 0.10)
 (NovSchema BC1 NegativeGain 0.50)
 (OlderVar
  (If (rand 0.10)
   (IfVarCond == NegativeGain 0
    (If (rand 0.25)
     VarBestNetGainBC0
     (RNovschema BC1
      NetGain 0.60))
    (If (rand 0.10)
     VarSecondBestNegativeGainWFF
     (Select2Rank PositiveGain
      NetGain)))
   (IfTabu age5
    (RNovSchema BC1 NetGain 0.40)
    (If (rand 025)
     VarRandomBC0
     VarBestNegGainBC1)))
  (IfTabu age5
   (RNovSchema BC1 Net 0.40)
   (If (rand 0.25)
    (NovSchema BC0 Negative 0.45)
    (If (rand 0.25)
     VarBestNegativeGainBC1
     VarBestNetGainBC0)))))
```

Figure 5: CLH1, a heuristic discovered by CLASS-L

the composed heuristics, it is intuitive to attempt to help the learning system by providing "good" building blocks. We therefore added a library of hand-selected s-expressions. The library is used as follows. CLASS-L still generates a random population, but in addition, it loads the library into a separate array. The new CLASS-L selection function, with probability p_L, picks an individual from the library, instead of the population; otherwise, it picks an individual from the population. Currently, the library consists of 50 s-expressions, including encodings of all of the standard heuristics (GSAT, GWSAT, all the Walksat variants in (McAllester, Selman, & Kautz 1997), Novelty, and R-Novelty), as well as s-expressions which perform poorly by themselves but were believed to be possibly useful as building blocks. In addition, we added two new primitives so that we could compactly represent variants of Novelty and R-Novelty:

`(NovSchema clause gaintype` P_{noise}`)` - executes the Novelty decision procedure on the given clause as defined in (McAllester, Selman, & Kautz 1997), using P_{noise} as the noise parameter. Instead of using only net gain (as done by the version presented in (McAllester, Selman, & Kautz 1997), any gain metric can be used depending on gaintype. For example, the standard Novelty heuristic is `(NovSchema BC0 netgain 0.5)`.

`(RNovSchema clause gaintype` P_{noise}`)` - similar to NovSchema, but executes the R-Novelty decision process schema. Using CLASS-L, we generated a new variable selection heuristic, CLH1 (Figure 5) after a 400-population run which generated 6000 candidate heuristcs.

4 Experimental Results

We empirically evaluated the automatically discovered heuristics CH1, CH2, and CLH1, using a set of standard SAT local search benchmark instances. The experimental design is similar to that of (Schuurmans & Southey 2001). For comparison, the results of Walksat with noise parameter 0.5, Novelty with a noise parameter of 0.5, and Novelty+ with noise parameter 0.5 and random walk probability 0.01) taken from (Schuurmans & Southey 2001) are shown in Table 1

All of the benchmark instances in this paper were obtained from SATLIB (www.satlib.org). Failure % is the percentage of runs which terminated without finding a solution; flips is the mean number of flips used by the runs which succeeded.

4.1 Evaluation of Learned Heuristic vs. Standard Local Search Heuristics

Performance vs. Test instances from Target Distribution Recall that CLASS uses a training set of 50 and 100 variable random 3-SAT instances as the training instances during the learning process. We first evaluated the performance of the learned heuristics on test instances from the same problem problem class as the training instances (the 1000 instances in the uf100-430 benchmark set). To evaluate how the learned heuristics performed relative to the best hand-coded heuristics on the target distribution, we first needed to find the best control parameter values for the best standard heuristics. We tuned R-Novelty by varying the noise parameter at 0.01 increments, and we also tuned R-Novelty+ by varying both the noise parameter and the random walk parameter at 0.01 increments, and measuring their performance on 100 independent runs on the 1000 uf100-430 benchmark instances. We found that R-Novelty with a noise setting of 0.68 had the best performance (with a cutoff of 500,000 flips; mean of 100 runs per instance). As shown on Table 1, the performance of CH1 and CH2, which were generated by CLASS, is competitive with all but the highly tuned R-Novelty(0.68). CLH1, which was generated by CLASS-L, actually outperforms R-Novelty(0.68).

Generalization and Scaling We have already shown that the learned heuristics are capable of some generalization, since they performed well on the uf100-430 instances, which are different problem instances than the problems in the training set (although they are from the same abstract class of 100 variable, 430 clause random generate 3-SAT problems).

Next, we evaluated the learned heuristics (and also the tuned R-Novelty(0.68)) on benchmarks from different problem classes to see how well the heuristics generalized and scaled beyond the test distribution for which they were specifically trained. As shown in Table 1, CH1 CH2, and CLH1 scaled and generalized well on larger hard 3-SAT instances from the phase-transition region (uf150, uf200, and uf250, which are 150-250 variable instances). They generalized fairly well to the graph-coloring and All-Interval-Series instances (flat125, ais6, 8, 10). CLH1 also generalized well on the planning instances (medium, huge, bw_large(a-c), and logistics.c), but CH1 and CH2 significantly degraded on the larger planning instances.

	successes	flips	depth	mobility	coverage
GWSAT	285	4471	7.998	11.6829	0.00007854
Walksat	902	2151	6.965	14.5036	0.0004819
R-Novelty	987	1101	8.232	23.237	0.001183
CLH1	989	1045	7.892	22.468	0.001249
CH1	958	1624	7.30	18.42	0.000789
CH2	952	1771	7.08	20.505	0.000882

Table 2: Local search characteristics of learned and standard heuristics (100 instances, 10 runs, cutoff=10000)

It is important to keep in mind that none of the heuristics (hand-coded or learned) were tuned for these other instances; we present this data in order to show how well the learned heuristic generalizes relative to the generalization of the tuned, hand-coded heuristics.

In order to discover heursitics which perform very well on structured problem instances, it is likely that the training instances need to be tailored to the target class of structured instances. Research in the related area of variable selection heuristics for systematic, Davis-Putnam-Loveland based SAT algorithms has shown that variable selection heuristics have various utilities depending on the class of problems to which they are obtained.[1] Nevertheless, the data shows that traning based on hard random 3-SAT instances is sufficient to generate heuristics with respectable performance on structured problems.

4.2 Local search characteristics of learned heuristics

Schuurmans and Southey (Schuurmans & Southey 2001) recently identified several metrics of local search behavior that were shown to predict the problem solving efficiency of standard SAT heuristics. Depth measures how many clauses remain unsatisfied over the course of a run, Mobility measures how rapidly a search moves in the search space, and Coverage measures how systematically the search explores the search space (see (Schuurmans & Southey 2001) for detailed definitions of the metrics). We measured these characteristics for CH1, CH2, and CLH1 on 100 instances from the uf100-430 benchmarks (uf100-0001 through uf100-0100, 10 runs per instance, 10,000 flips). As shown in Table 2, the depth, mobility, and coverage characteristics of the learned heuristics relative to the standard algorithms are consistent with their performance.

Next, we conducted a large-scale experiment to test the predictive power of the depth, mobility, and coverage metrics on a large sample of heuristics generated by CLASS. 1200 heuristics were chosen as follows: 400 from the population at the end of the CLASS run which produced CH2, 400 from the initial (random) population of the same run, and 400 from the population at the end of the CLASS-L run

[1] Unit propagation-based heuristics have been found to be highly effective for random 3-SAT instances (Li & Anbulagan 1997), while simpler strategies have been found effective for structured instances (Moskewicz *et al.* 2001).

Instance Set*	Walksat(0.5)		Novelty(0.5)		Novelty+(.5,.01)		R-Novelty(0.68)		CLH-1		CH-1		CH-2	
	fail %	flips	fail %	flips	fail %	flips	fail %	flips	fail %	flips	fail %	flips	fail %	flips
uf100(1000)	0	3655	0	3801	0	2298	0	1258	0	1124	0	2112	0	2205
uf150(100)	0.3	14331	0.15	9573	0.03	8331	0	5102	0	4317	0.09	8176	0.02	8777
uf200(100)	2.9	41377	2.5	31794	2.3	28529	2.03	19946	1.67	14533	2.08	19162	2.66	22358
uf250(100)	1.6	41049	2.1	32864	2.2	31560	2.82	23849	1.47	16541	0.96	23578	1.67	28457
medium(1)	0	1167	0	392	0	537	0	332	0	316	0	510	0	525
huge(1)	0	20211	0	11382	0	12419	0	12167	0	6454	0	10901	0	9805
logistics.c(1)	42	332822	2	135382	1	163622	7	152218	0	81306	83	221030	23	203460
bw_large.a(1)	0	20336	0	9695	0	10788	0	10151	0	6656	0	10126	0	11803
bw_large.b(1)	58	377348	48	343078	53	373001	79	245243	24	186508	72	223377	46	192420
ais6(1)	0	1377	92	460007	0	12031	0	7279	0	1413	0	1001	0	660.96
ais8(1)	0	36499	99	495003	8	169626	16	139036	0	43050	0	24952	0	21559
ais10(1)	37	317323	100	n/a	84	451222	73	259861	60	216420	34	203709.8	31	184884.6
flat125(100)	1.5	74517	3.2	91004	0.8	37408	18.21	125946	4.63	71125	0.94	60219	0.97	41220

Table 1: Performance on SATLIB benchmarks. Each entry is the mean of 100 runs on each instance; cutoff=500,000 flips per run.
* Number in parentheses are the # of instances in each instance set; values for Walksat, Novelty, and Novelty+ are from (Schuurmans & Southey 2001)

which generated CH2, and. This way, we sought to sample a wide range of heuristics, ranging from very poor (random) to good (the population with CH2) to very good (the population containing CLH1). Each of these heuristics was executed on 100 instances from the uf50-215 (50 variable, 215 clause) benchmarks from SATLIB (100 runs per instance with cutoff of 1000 flips). Figures 5 shows the depth, mobility, and coverage metrics versus the heuristic's performance (number of successful runs). There is a very high correlation between performance and coverage (r=0.89), and weaker correlations with depth (r= -0.31) and mobility (r=0.138). The weak correlations between performance and the mobility metric is caused by the large number of very bad heuristics in the randomly generated subset of heuristics which exhibit extreme reckless mobility. Without the random heuristics, the correlation increased to r=0.30. On the other hand, the correlation between depth and performance without the random heuristics was 0.22 (note the change in the sign of the coefficient). Figure 6 indicates that high coverage seems to be both a necessary and sufficient characteristic for local search success on these random 3-SAT instances. Similarly, for mobility and depth, there is apparently a "correct" range of values which are necessary, although not sufficient, for good performance.

5 Related work

Several previous systems have learned to improve the performance of constraint satisfaction systems by modifying heuristics using what is essentially a heuristically guided generate-and-test procedure like CLASS. MULTI-TAC (Minton 1996) adapts generalized constraint-satisfaction heuristics for specific problem classes. COMPOSER was used to configure an antenna-network scheduling algorithm (Gratch & Chien 1996). STAGE (Boyan & Moore 2000) is a learning algorithm applied to SAT local search. STAGE uses an on-line learning to adapt its heuristic by predicting objective function values based on features seen during the search; in contrast, CLASS is an off-line learning system. CLASS strongly resembles a genetic programming (GP) system (Koza 1992). In fact, we can view the CLASS as a heavily modified GP - specifically, a strongly-typed (Montana 1993), steady-state GP using only a novel composition operator. Note that unlike our composition operator, standard GP operators recombine and modify arbitrary subcomponents of the parents, and do not propagate the PAC property. Our experiments with implementation of various GP mutation and crossover operators in CLASS have not been successful yet.

6 Discussion and Future Work

We have described and evaluated an automated discovery system for finding SAT local search variable selection heuristics. It is interesting that repeated brute-force application of a single operator, composition, to randomly generated heuristics is sufficient to generate CH1 and CH2, which are competitive with Walksat and Novelty variants. CLASS-L, an extension that uses the Novelty schema as primitives, generated CLH1, which is competitive with the best, tuned version of the standard local search heuristics on the target problem class (100-variable random 3-SAT). All three learned heuristics were shown to scale and generalize well on larger random instances; generalization to other problem classes varied.

Despite of the good performance on the benchmarks, there is a concern that the automatically generated heuristics might be "getting lucky", or exploiting some bizarre, hidden structure of the benchmarks. Measurements of the depth, mobility, and coverage metrics show that CH1, CH2, and CLH1 exhibit the characteristics of successful local search algorithms as identified in (Schuurmans & Southey 2001). The best learned heuristics seems to descend to promising regions and explore near the bottom of the objective as rapidly (low depth), broadly (high mobility), and systematically (high coverage) as possible, compared to the standard algorithms. In other words, there is convincing evidence that the best learned heuristics perform well because they behave in the way that good local search algorithms are expected to behave (according to our current scientific understanding of

local search), and not because of some unexplainable exploitation of the benchmark instances. Furthermore, note that with a little effort, one can study Figures 4-5 and observe that the learned heuristics encode "reasonable" behavior.

Recent algorithms such as DLM (Wu & Wah 1999) and SDF (Schuurmans & Southey 2001) are significantly different from the GSAT and Walksat family considered in this paper, due to 1) clause-weighting and 2) objective function metrics that go beyond the simple gain metrics identified in Section 2. For example, SDF uses a scoring metric based on the number of variables that satisfy each clause. Such advances in the building blocks of SAT local search algorithms are orthogonal to the issue addressed by CLASS, which is the problem of combining these building blocks into effective decision procedures. Future work will extend CLASS with a clause-weighting mecshanism, as well new objective function metrics (such as the SDF scoring function).

In our current implementation, the learned heuristic functions are slower (execute fewer flips per second) than the standard heuristics. In principle, learned heuristics based on primitives which only select variables from a particular clause should not be much more expensive than Walksat, and learned heuristics which require maintaining globally optimal variables should not be much more expensive than GSAT. This is because in an efficient implementation, the complexity of a variable flip is dominated by the incremental computation of gain values, which scales linearly with the size of the problem (average # of clauses per variable), while repeatedly accessing these values repeatedly in a complex selection heuristic is a constant overhead, which should become asymptotically insignificant. In addition, although there is currently no mechanism in CLASS to bias the system for the discovery of faster, simpler heuristics, there is much that can be done to speed up the learned heuristics. For example, runtime, size, and complexity could be used as part of the scoring function for the discovery algorithm. Also, post-processing optimizations of the heuristics is possible. For example, in CH1, the single occurrence of the VarBestNetGainWFF primitive requires instantiation of the GSAT-equivalent data structures to identify the globally optimal variable with respect to net gain, causing significant slowdown; without this symbol, CH1 would only need Walksat-equivalent data structures. However, since that primtive is only called 2.5% of the time (it is called only after 3 layers of randomization), an optimizing postprocessor could try to eliminate this kind of bottleneck from the heuristic while maintaining search efficiency.

Evaluation of 1200 new heuristics generated by CLASS with respect to depth, mobility, and coverage metrics (Schuurmans & Southey 2001) supported the conjecture that these metrics are widely applicable for analyzing SAT local search heuristics. In particular, the coverage rate seems to be the most highly correlated with performance on random 3-SAT instances. Furthermore, this result indicates that CLASS could possibly use the metrics (particularly coverage) as a partial proxy for actual algorithm runs in order to evaluate candidate heuristics much faster is currently possible.

Local search algorithms are very sensitive to control parameters (McAllester, Selman, & Kautz 1997; Hoos & Stutzle 2000). CLASS currently does not explicitly perform parameter tuning, although nested probabilistic compositions yields the equivalent in some cases (CH1 obviously exploits this trick). While we have implemented parameter tuning, it is currently turned off because in our current implementation, local optimization via control parameter tuning takes just as much computational resources as evaluating an entirely new heuristic; so far, the tradeoff has favored broader exploration of the space of structures. However, using metrics such as those in (Schuurmans & Southey 2001; McAllester, Selman, & Kautz 1997) as a proxy for complete runs might enable fast parameter tuning.

While the surprisingly good performance of heuristics learned by CLASS is interesting, the primary motivation of this work is to explore an automated approach to the process of designing composite search heuristics. Humans excel at finding and classifying the relevant features/components that can be used to solve problems, such the primitives identified in Section 2. Note that all of these primitives were proposed in the literature by 1993, shortly after the introduction of GSAT - variants that used variable age, negative gain, random walk can be found in (Gent & Walsh 1993a; Selman & Kautz 1993). However, the task of combining these features into effective composite heuristics appears to be a combinatorial problem that is difficult for humans. As evidence for this, note that Novelty was not introduced until 1997 (McAllester, Selman, & Kautz 1997). We have shown that this task can be effectively formulated and solved as a meta-level search problem. SAT local search algorithms have been intensely studied by many researchers for 10 years. The demonstration that a simple mechanical procedure which composes previously proposed primitives can compete with some of the best human-designed heuristics suggests that the problem of designing composite search heuristics, in general, might benefit from an automated discovery system (especially for problem domains which have not been as intensely studied as SAT).

Acknowledgments

Thanks to Rich Korf and Jason Fama for helpful discussions. Finnegan Southey provided valuable clarifications regarding the implementation of the coverage metric measurements.

References

Boyan, J., and Moore, A. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1(2).

Gent, I., and Walsh, T. 1993a. An empirical analysis of search in gsat. *Journal of Artificial Intelligence Research* 1:47–59.

Gent, I., and Walsh, T. 1993b. Towards an understainding of hill-climbing procedures for sat. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*, 28–33.

Gratch, J., and Chien, S. 1996. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research* 4:365–396.

Hoos, H., and Stutzle, T. 2000. Local search algorithms for sat: An empirical evaluation. *Journal of Automated Reasoning* 24:421–481.

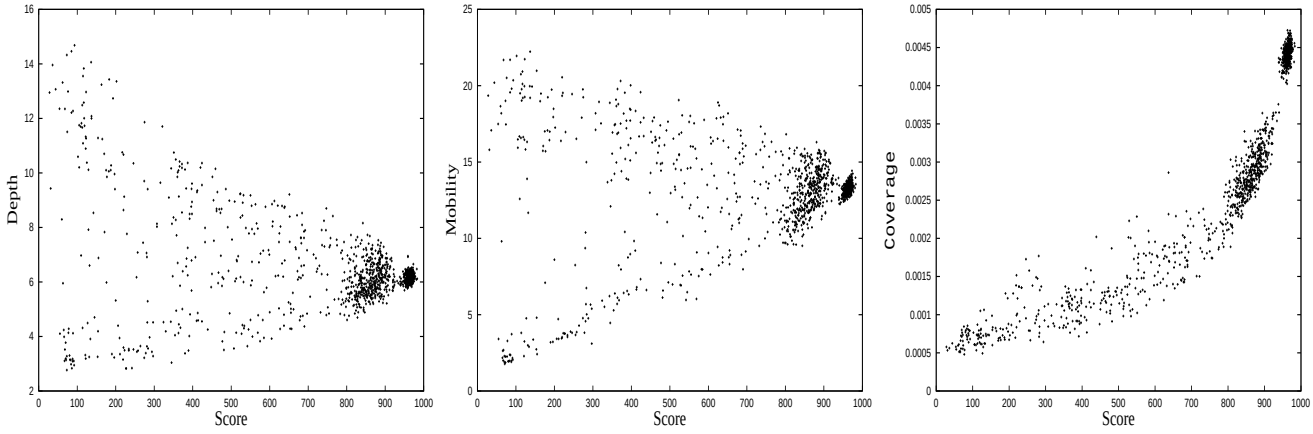

Figure 6: Average depth, mobility, and coverage rates vs. Score (# of successful runs) for 1200 automatically generated heuristics

Hoos, H. 1998. *Stochastic local search - methods, models, applications*. Ph.D. Dissertation, TU Darmstadt.

Koza, J. 1992. *Genetic Programming: On the Programming of Computers By the Means of Natural Selection*. MIT Press.

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *Proc. Intl. Joint Conf. Artificial Intelligence (IJCAI)*, 366–371.

McAllester, D.; Selman, B.; and Kautz, H. 1997. Evidence for invariants in local search. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*, 459–465.

Minton, S. 1996. Automatically configuring constraint satisfaction problems: a case study. *Constraints* 1(1).

Mitchell, D.; Selman, B.; and Levesque, H. 1992. Hard and easy distributions of sat problems. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*, 459–65.

Montana, D. 1993. Strongly typed genetic programming. Technical report, Bolt, Beranek and Neuman (BBN).

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, 530–535.

Schuurmans, D., and Southey, F. 2001. Local search characteristics of incomplete sat procedures. *Artificial Intelligence* 132:121–150.

Selman, B., and Kautz, H. 1993. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proc. Intl. Joint Conf. Artificial Intelligence (IJCAI)*.

Selman, B.; Kautz, H.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A new method for solving hard satisfiability problems. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*, 440–446.

Wu, Z., and Wah, B. 1999. Trap escaping strategies in discrete lagrangian methods for solving hard satisfiability and maximum satisfiability problems. In *Proceedings of National Conf. on Artificial Intelligence (AAAI)*, 673–678.

Appendix: Additional CLASS Language Primitives

List of some CLASS language primitives that appear in Fig. 2,4,5. The actual syntax of the CLASS s-expressions differs slightly, but we present a simplified version for brevity and clarity (see Section 3.1 and 3.3 for definitions of some other primitives).

`(rnd num)` - true if random number $r < num$, else returns false.

`BC0` - randomly selected broken clauses. Note that BC0 refers to the same broken clause throughout an s-expression. Also, note that the symbol BC1 refers to a different randomly selected clause.

`(NegativeGain v)` and `(NetGain v)` returns the negative gain and net gain of variable v.

`VarRandomBC0` - a random variable from broken clause *BC0*

`VarRandomWFF` - random variable in the formula

`VarBestNetGainBC0` - variable with the best net gain in *BC0*.

`VarBestNegativeGainBC1` - var. with best negative gain in *BC1*.

`VarBestNetGainWFF` - the var. with best net gain in the formula

`(OlderVar` v_1 v_2`)` - picks the var with the max age from v_1 and v_2.

`(IfNotMinAge varset` v_1 v_2`)` - if v_1 does not have minimal age among variables in a set of vars then return v_1, else v_2.

`(Select2RankWFF gaintype1 gaintype2 )` - select best variable from formula according to *gaintype1*, breaking ties using *gaintype2*.