

Solving Non-Boolean Satisfiability Problems with Stochastic Local Search

Alan M. Frisch and Timothy J. Peugniez

Artificial Intelligence Group
Department of Computer Science
University of York
York YO10 5DD
United Kingdom

Abstract

Much excitement has been generated by the recent success of stochastic local search procedures at finding satisfying assignments to large formulas. Many of the problems on which these methods have been effective are non-Boolean in that they are most naturally formulated in terms of variables with domain sizes greater than two. To tackle such a problem with a Boolean procedure the problem is first reformulated as an equivalent Boolean problem. This paper introduces and studies the alternative of extending a Boolean stochastic local search procedure to operate directly on non-Boolean problems. It then compares the non-Boolean representation to three Boolean representations and presents experimental evidence that the non-Boolean method is often superior for problems with large domain sizes.

1 Introduction

Much excitement has been generated by the recent success of stochastic local search (SLS) procedures at finding satisfying truth assignments to large formulas of propositional logic. These procedures stochastically search a space of all assignments for one that satisfies the given formula. Many of the problems on which these methods have been effective are non-Boolean in that they are most naturally formulated in terms of variables with domain sizes greater than two. To tackle a non-Boolean problem with a Boolean procedure, the problem is first reformulated as an equivalent Boolean problem in which multiple Boolean variables are used in place of each non-Boolean variable.

This encode-and-solve approach often results in comparable, if not superior, performance to solving the problem directly. Because Boolean satisfiability is conceptually simple, algorithms for it are often easier to design, implement and evaluate. And because SLS algorithms for Boolean satisfiability have been studied intensively for more than a decade, highly-optimised implementations are publicly available.

This paper proposes and studies a new approach to solving non-Boolean satisfaction problems: that of generalising a Boolean SLS procedure to operate directly on a non-Boolean formula by searching through a space of assignments to non-Boolean variables. In particular, we have generalised Walk-

sat [Selman *et al.*, 1994], a highly-successful SLS procedure for Boolean satisfiability problems, to a new procedure, NB-Walksat [Peugniez, 1998; Frisch and Peugniez, 1998], that works on formulas whose variables have domains of any finite size.¹

In this way we are able to apply highly-refined SLS technology directly to non-Boolean problems without having to encode non-Boolean variables as Boolean variables.

The main question addressed by this paper is how the performance of the direct approach compares to that of the transformational (or encode and solve) approach. In particular we compare one direct method, NB-Walksat, and three transformational methods by empirically testing their ability to solve large graph colouring problems and large random non-Boolean formulas. Our three transformation methods consist of applying Walksat to the results of three transforms.

Boolean variables are merely a special case of non-Boolean variables, and, intuitively, the difference between the non-Boolean and Boolean variables grows as the domain size of the non-Boolean variable increases. Consequently, one would expect that in a comparison of encodings for non-Boolean problems that domain size would be the most important parameter to consider and that one would find that any difference in performance between the encodings would increase when domain size is increased. Ours is the first study to consider this.

Our experimental results show NB-Walksat to be highly effective, demonstrating that the effectiveness of the Walksat strategies can be transferred from the Boolean case to the non-Boolean case. On problems with large domain sizes our direct method is often superior to the transformation methods, which, in some cases, are ineffective.

In the course of studying this new method of solving non-Boolean problems with SLS, we make several other new contributions. Most naturally, we have had to generalise the notion of a Boolean formula to that of a non-Boolean formula. Of the three transformations we use, one is new and one is an enhanced version of a transformation used by others. In order to test the effect of domain size on problem solving performance we want a method for generating random formulas that vary in domain size but are similar in other respects. We

¹NB-Walksat and a suite of supporting programs are available at <http://www.cs.york.ac.uk/~frisch/NB>.

propose such a method and use it in our experiments.

2 Non-Boolean Formulas

Non-Boolean formulas are constructed from propositional variables, each of which is associated with a finite, non-empty domain. A non-Boolean assignment maps every variable to a member of its domain. Atomic non-Boolean formulas (or nb-atoms) are of the form X/d , where X is a variable and d is a member of its domain. This formula is assigned *TRUE* by an assignment if the assignment maps X to d ; otherwise it is assigned *FALSE*. Molecular non-Boolean formulas are constructed from atomic non-Boolean formulas and truth-functional connectives with the usual syntax and Boolean semantics. Observe that though non-Boolean variables can take on values from a domain of arbitrary size, the semantics is still two-valued in that every formula is assigned either *TRUE* or *FALSE*. As usual, we say that an assignment satisfies a non-Boolean formula if it maps it to *TRUE*.

Walksat, and many other Boolean SLS procedures, operate on Boolean formulas in conjunctive normal form (CNF), and NB-Walksat, our generalisation of Walksat, operates on non-Boolean formulas in CNF. A formula, Boolean or non-Boolean, is in CNF if it is a conjunction of disjunctions of literals. A literal is either an atomic formula (called a positive literal) or its negation (called a negative literal). We say that a CNF formula is *positive* if all its literals are positive and *negative* if all its literals are negative.

Non-Boolean formulas generalise Boolean formulas since a Boolean formula can be transformed to a non-Boolean formula by replacing every atom P with P'/TRUE , where P' is a variable whose domain is $\{\text{TRUE}, \text{FALSE}\}$.

We sometimes use terms such as “nb-atom” or “nb-formula” to emphasise that these syntactic objects are part of the non-Boolean language. Similar use is made of terms such as “b-atom” and “b-formula”.

3 NB-Walksat

Walksat is a highly successful SLS procedure for finding satisfying assignments to Boolean formulas in CNF. We have generalised Walksat to a new system, NB-Walksat, that operates similarly on non-Boolean formulas. Indeed when handling a Boolean formula the two procedures perform the same search.² This paper is concerned with version 4 of NB-Walksat, which we derived from Walksat version 19. This section describes the operation of NB-Walksat and, since on Boolean formulas NB-Walksat and Walksat perform the same search, this section implicitly describes the operation of Walksat.

The simplest way to understand the operation of NB-Walksat is to consider it as working on positive CNF nb-formulas. This can be achieved by considering NB-Walksat’s first step to be the replacement of every negative literal $\neg X/d_i$ with $X/d_1 \vee \dots \vee X/d_{i-1} \vee X/d_{i+1} \vee \dots \vee X/d_n$, where X is a variable with domain $\{d_1, \dots, d_n\}$.

²We used this property to help test that NB-Walksat was correctly implemented.

NB-Walksat operates by choosing a random assignment and then, until a satisfying assignment is found, repeatedly selecting a literal from an unsatisfied clause and modifying the assignment so as to satisfy that literal. Since the selected literal, X/d , occurs in an unsatisfied clause, the present assignment must map X to a value other than d . The present assignment is modified so that it maps X to d , and its mapping of all other variable is unmodified. We say that the literal X/d has been *flipped*.

What distinguishes NB-Walksat and Walksat from other procedures is the heuristic employed for selecting which literal to flip. Though Walksat provides a range of such heuristics, the greedy heuristic is generally the most effective [Selman *et al.*, 1994] and many reported experiments have used it. Consequently, it is the greedy version of Walksat that forms the basis for NB-Walksat and only the greedy heuristic is considered in this paper.

NB-Walksat with the greedy heuristic chooses a literal to flip by first randomly selecting a clause with uniform distribution from among all the clauses that are not satisfied by the current assignment. We say that flipping a literal *breaks* a clause if the clause is satisfied by the assignment before the flip but not after the flip. If the selected clause contains a literal such that flipping it would break no clauses, then the literal to flip is chosen randomly with uniform distribution from among all such literals. If the selected clause contains no such literals, then a literal is chosen either (i) randomly with uniform distribution from the set of all literals in the clause or (ii) randomly with uniform distribution from among the literals in that clause that if flipped would break the fewest clauses. The decision to do (i) or (ii) is made randomly; with a user-specified probability, P_{noise} , the “noisy” choice (i) is taken.

4 Transforming Non-Boolean Formulas

To transform nb-satisfaction problems to b-satisfaction problems we map each nb-formula to a b-formula such that the satisfying assignments of the two formulas correspond. This paper presents three such transforms, called the *unary/unary*, *unary/binary* and *binary* transforms. Each operates on an arbitrary formula, though our experiments only apply the transforms to CNF formulas. Each transform operates by replacing each atom in the nb-formula with a b-formula that, in a sense, encodes the nb-atom it replaces. The resulting formula is known as the *kernel* of the transformation. The transforms employ two ways of producing a kernel; we call these two encodings “unary” and “binary”.

If the unary encoding of the kernel is used, the transform also needs to conjoin two additional formulas to the kernel, known as the *at-least-one* formula (or ALO formula) and the *at-most-one* formula (or AMO formula). As with the kernel, two encodings can be used for the ALO and AMO formulas: unary and binary. The three transforms we use in this paper are *binary* (which uses a binary encoding for the kernel and no ALO or AMO formula), *unary/unary* (which uses unary encodings for the kernel and for the ALO and AMO formulas) and *unary/binary* (which uses a unary encoding for the kernel and a binary encoding for the ALO and AMO formulas).

The longer version of this paper analyses the sizes of the formulas produced by all the encodings.

The Unary/Unary Transform The unary/unary transform produces a kernel by transforming each nb-atom X/d to a distinct propositional variable, which we shall call $X:d$. The idea is that a Boolean assignment maps $X:d$ to *TRUE* if and only if the corresponding non-Boolean assignment maps X to d . Thus, the role of an nb-variable with domain $\{d_1, \dots, d_n\}$ is played by n b-variables.

Furthermore, one must generally add additional formulas to the Boolean encoding to represent the constraint that a satisfying assignment must satisfy exactly one of $X:d_1, \dots, X:d_n$. This constraint is expressed as a conjunction of an ALO formula (asserting that at least one of the variables is true) and an AMO formula (asserting that at most one of the variables is true). To state that at *least* one of $X:d_1, \dots, X:d_n$ must be satisfied we simply use the clause $X:d_1 \vee \dots \vee X:d_n$. The entire ALO formula is a conjunction of such clauses, one clause for each nb-variable. To say that at *most* one of $X:d_1, \dots, X:d_n$ must be satisfied we add $\neg X:d_i \vee \neg X:d_j$, for all i and j such that $1 \leq i < j \leq n$. The entire AMO formula is a conjunction of all such clauses produced by all nb-variables.

Notice that these ALO and AMO formulas are in CNF. And since the transform produces a kernel whose form is identical to that of the original formula, the entire b-formula produced by the unary/unary transform is in CNF if and only if the original nb-formula is.

The Binary Transform The unary/unary transform uses D b-variables to encode a single nb-variable of domain size D and, hence, uses a base 1 encoding. By using a base 2 encoding, the binary transformation requires only $\lceil \log_2 D \rceil$ b-variables to encode the same nb-variable.³ If X is a variable of domain size D then the binary transform maps an nb-literal of the form X/d_i by taking the binary representation of $i - 1$ and encoding this in $\lceil \log_2 D \rceil$ Boolean variables. For example, if X has domain $\{d_1, d_2, d_3, d_4\}$ then

X/d_1 is mapped to $\neg X_2 \wedge \neg X_1$,
 X/d_2 is mapped to $\neg X_2 \wedge X_1$,
 X/d_3 is mapped to $X_2 \wedge \neg X_1$ and
 X/d_4 is mapped to $X_2 \wedge X_1$.

To see what happens when the domain size is not a power of two, reconsider X to have the domain $\{d_1, d_2, d_3\}$. If we map X/d_1 , X/d_2 and X/d_3 as above then there is a problem in that the Boolean assignment that satisfies $X_2 \wedge X_1$ does not correspond to an assignment of a domain value to X . One solution to this problem, which has been employed by Hoos [1998], is to add an ALO formula to ensure that the *extraneous* binary combination $X_2 \wedge X_1$ cannot be satisfied in any solution.

Here we introduce and use a new and better solution to this problem in which no extraneous combinations are produced

³We write $\lceil x \rceil$ to denote the smallest integer that is greater than or equal to x .

and therefore no ALO formula is required. In this example the extraneous combination is eliminated if

X/d_1 is mapped to $\neg X_2$,
 X/d_2 is mapped to $X_2 \wedge \neg X_1$ and
 X/d_3 is mapped to $X_2 \wedge X_1$.

Here, the transform of X/d_1 covers two binary combinations: $(\neg X_2 \wedge X_1)$ and $(\neg X_2 \wedge \neg X_1)$.

To see what happens in general let X be a variable with domain $\{d_1, \dots, d_n\}$ and let k be $2^{\lceil \log_2 n \rceil} - n$. Then $X/d_1, \dots, X/d_k$ are each mapped to cover two binary combinations and $X/d_{k+1}, \dots, X/d_n$ are each mapped to cover a single binary combination.

Notice that this transform generates no extraneous binary combinations. Also notice that, as a special case, if n is a power of two then each X/d_i ($1 \leq i \leq n$) is mapped to cover a single binary combination. Finally, to confirm that this binary transform requires no AMO formula and no ALO formula, observe that every Boolean assignment must satisfy the binary transform of exactly one of $X/d_1, \dots, X/d_n$.

Notice that the binary transformation of a CNF formula is not necessarily in CNF. However, the binary transformation of a negative CNF formula is *almost* in CNF; it is a conjunction of disjunctions of negated conjunctions of literals. By using DeMorgan's law, the negations can be moved inside of the innermost conjunctions, resulting in a CNF formula of the same size. At the other extreme, the binary transformation of a positive CNF formula is a conjunction of disjunctions of conjunctions of literals. One way of transforming this to CNF is to distribute the disjunctions over the conjunctions. Unfortunately applying this distribution process to the binary transform of a positive nb-clause produces a CNF formula that can be exponentially larger than the original clause.

The Unary/Binary Transform The unary/binary transform produces the same kernel as the unary/unary transform. The ALO and AMO formulas it produces achieve their effect by introducing the binary encodings of nb-atoms and linking each to the unary encoding of the same nb-atom. Since the binary encoding requires no ALO or AMO formulas, the unary/binary encoding requires no ALO or AMO formulas beyond these linking formulas.

The links provided by the AMO formula state that for each nb-atom, X/d_i , its unary encoding, $X:d_i$, implies its binary encoding. For example, if the nb-variable X has domain $\{d_1, d_2, d_3, d_4\}$ then the AMO linking formula for X is

$$(X:d_1 \rightarrow (\neg X_2 \wedge \neg X_1)) \wedge (X:d_2 \rightarrow (\neg X_2 \wedge X_1)) \wedge \\ (X:d_3 \rightarrow (X_2 \wedge \neg X_1)) \wedge (X:d_4 \rightarrow (X_2 \wedge X_1)).$$

The entire AMO formula is a conjunction of such linking formulas, one for each nb-variable.

The links provided by the ALO formula state that for each nb-atom, X/d_i , its unary encoding $X:d_i$ is implied by its binary encoding. For example, if the nb-variable X has domain $\{d_1, d_2, d_3, d_4\}$ then the ALO-linking formula for X is

$$((\neg X_2 \wedge \neg X_1) \rightarrow X:d_1) \wedge ((\neg X_2 \wedge X_1) \rightarrow X:d_2) \wedge \\ ((X_2 \wedge \neg X_1) \rightarrow X:d_3) \wedge ((X_2 \wedge X_1) \rightarrow X:d_4).$$

The entire ALO formula is a conjunction of linking formulas, one linking formula for each nb-variable. Observe that both

the AMO and ALO linking formulas can be put easily into CNF with at most linear expansion.

5 Experimental Results

This section presents experiments on graph colouring problems and random nb-formulas that compare the performance of the four methods, which we shall refer to as NB (non-Boolean encoding), BB (binary encoding) UU (unary/unary encoding) and UB (unary/binary encoding). In all experiments, Walksat version 35 was used to solve the Boolean encodings and NB-Walksat version 4 was used to solve the non-Boolean encodings, even in cases where the domain size is 2. Both programs provide the user the option of either compiling the program with fixed size data structures or allocating the data structures when the formula is input at runtime; the latter option was used in all experiments. All experiments were run on a 600 MHz Pentium III with 126 megabytes of memory.

Considerable care must be taken in setting the P_{noise} parameter for the experiments. Much work in this area has been reported without giving the value used for P_{noise} , and thus is irreproducible. Setting the parameter to any fixed value over all formulas is not acceptable; we have observed that a parameter setting that is optimal for one formula can be sub-optimal by several orders of magnitude for another formula. The best option is to report performance at the optimal setting for P_{noise} , which—in the absence of any known method to determine this *a priori*—we have determined experimentally. This is also the route followed by Hoos [1998] in his extremely careful work.

In using SLS procedures it is often considered advantageous to restart the search at a new, randomly-selected assignment if the procedure has not found a solution after a prescribed number of flips. This issue is not relevant for the current study. Since the optimal restart point and the amount gained by restarting are functions of the procedure's performance without restarts, this study need only be concerned with the performance without restarts.

Graph Colouring Our experiments used 6 instances of the graph colouring problem that are publicly available at the Rutgers Univ. DIMACS FTP site.⁴

Each problem instance was encoded as a CNF nb-formula. For each node in the graph the formula uses a distinct variable whose domain is the set of colours. The formula itself is a conjunction of all clauses of the form $\neg X/c \vee \neg Y/c$, such that X and Y are nodes connected by an arc and c is a colour.

Three Boolean CNF representations of each problem instance were produced by applying the three transforms of Section 4 to the non-Boolean representation. The longer version of this paper proves that a negative CNF nb-formula can be transformed to a b-formula without the AMO formulas. So the two unary encodings used here contain only kernels and ALO clauses. Also because the nb-clauses representing instances of the graph colouring problem are negative, the binary transform maps each nb-clause to a single b-clause (as discussed in Section 4).

⁴<ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/>

Figure 1 shows the results obtained for solving each problem instance 101 times with each of the four methods. The formula size column shows the size (in terms of atom occurrences) of the formula input to Walksat or NB-Walksat. All experiments were run with P_{noise} set to its optimal value, which is recored in the P_{noise} column. The flip rate column gives the mean flip rate over all 101 runs. The median flips and median time column gives the sample median of the number of flips and amount of cpu time taken to solve the problem.

Examination of these results reveals that over the range of problems NB and UU have roughly comparable (within a factor of 2 to 3) solution times.⁵ With increasing domain size (number of colours) the NB fliprate drops sharply but the advantage that NB has in terms of number of flips increases. These two opposing factors cancel each other, resulting in the roughly comparable solution times.

On the two problems with small domain size (5 colours) BB and UB produce solution times that are competitive with the other methods. However, on three problems with larger domain size—le450.15a with 15 colours and DSJC125.5.col with 17 and 18 colours—BB and UB do not compete well with the other methods. On le450.25a, a very easy problem with a domain size of 25, BB has competitive performance but UB is totally ineffective. We conjecture that UB cannot solve this very easy problem because the method is highly sensitive to domain size.

Random Non-Boolean CNF Formulas Since we can control certain parameters in the generation of random CNF nb-formulas, they provide a good testbed. In particular, since this paper is a study of solving problems with domain sizes greater than two, we would like to know how problem solving performance varies with domain size. To measure this we need to select problem instances that have different domain sizes but are otherwise similar. Formulating a notion of “otherwise similar” has been one of the most challenging problems faced by this research.

We have developed a program that generates random, positive CNF nb-formulas using five parameters: N , D , C , V and L . Each generated formula consists of exactly C clauses. Using a fixed set of N variables each with a domain size of D , each clause is generated by randomly choosing V distinct variables and then, for each variable, randomly choosing L distinct values from its domain. Each of the chosen variables is coupled with each of its L chosen values to form $L \cdot V$ positive literals, which are disjoined together to form a clause.

The simplest conjecture on how to study varying domain size is to fix the values of N , C , V , and L and then to systematically vary D . One can easily see that performance on this task would just exhibit typical phase-transition behaviour [Mitchell *et al.*, 1992]: small values of D would produce underconstrained problems, which would become critically-constrained and then over-constrained as D increases. The

⁵We caution the reader against taking solution time too seriously as we have found that the ratio of solution times between computers varies greatly from formula to formula.

Problem instance	method	P_{noise} setting	formula size	flip rate (flips/sec)	median flips	median time (secs)
le450.5a	NB	.560	57,140	34,000	242,000	7.12
5 colours	BB	.465	137,136	51,000	309,000	6.06
450 nodes	UU	.600	59,390	119,000	350,000	2.95
5714 arcs	UB	.453	64,790	232,000	1,700,000	7.34
le450.5d	NB	.703	97,570	17,700	98,700	5.58
5 colours	BB	.605	234,168	26,100	68,400	2.62
450 nodes	UU	.719	99,820	65,500	155,000	2.37
9757 arcs	UB	.597	105,220	132,000	729,000	5.52
le450.15a	NB	.088	245,040	15,400	196,000	12.7
15 colours	BB		963,824			
450 nodes	UU	.131	251,790	53,900	1,268,000	23.5
8168 arcs	UB		278,340			
DSJC125.5.col	NB	.177	132,294	9,490	495,000	52.2
17 colours	BB		544,740			
125 nodes	UU	.11	134,419	49,000	3,350,000	68.4
3891 arcs	UB		143,169			
DSJC125.5.col	NB	.202	140,076	8,920	6,850	0.768
18 colours	BB	.050	591,432	15,200	1,160,000	76.1
125 nodes	UU	.140	142,326	46,200	34,700	0.751
3891 arcs	UB	.016	151,826	273,000	1,590,000	5.82
le450.25a	NB	.120	413,000	2,840	638	.224
25 colours	BB	.088	1,949,360	6,460	2,900	0.449
450 nodes	UU	.163	424,250	27,400	11,700	0.427
8260 arcs	UB		477,350			

Figure 1: Results, given to three significant figures, for 101 sample runs on each graph colouring problem. On those rows missing entries, 11 attempts, each of 10,000,000 flips, were made at each of four P_{noise} settings (.083, .167, .25 and .33); no solutions were found.

problem instances generated by this method would not be similar in terms of their location relative the phase transition.

Our solution to this shortcoming is to vary D and to adjust the other four parameters so as to put the problem class at the phase transition—that is, at the point where half the instances in the class are satisfiable. But for any given value of D many combinations of values for N , C , V and L put the problem class at the phase transition. We determine the values of these parameters by keeping D^V , the size of the search space, at a fixed value for all problem instances and then requiring all clauses to have the same “constrainedness”, as measured by $(L/D)^V$. Mimicking Boolean 3CNF, we set V to three and aim to keep $(L/D)^V$ at $1/8$, which is achieved by setting L to $.5D$. This follows the constant length model advocated by Mitchell, Selman and Levesque [1992]. Finally C is set to whatever value puts the problem class at the phase transition, which we determined experimentally. Thus, while varying D , the parameters that we are holding constant are search space size, the number of variables constrained by each clause and the amount of constraint placed on each, and the proportion of instances with that parameter setting that are solvable.

Our experiments were conducted on domain sizes of 2, 4, 8, 16 and 32. Following the model above we kept D^V at a constant value, 2^{60} . Then for each domain size, we experimentally located the point of the phase transition. The leftmost column of Figure 2 shows the value of C (number of clauses) at which these experiments located the phase transitions.

With the parameters of the random formulas determined, at each domain size we generated a series of random formulas according to the above method and, using a non-Boolean version of the Davis-Putnam procedure, kept the first 25 satisfiable ones.⁶ At each domain size 101 attempts were made to solve the suite of 25 formulas with the NB, UU and UB meth-

ods. The longer version of this paper proves that a positive CNF nb-formula can be transformed to a b-formula without the ALO formulas. Since our randomly-generated formulas are positive, the two unary encodings that are used here contain only kernels and AMO formulas.

The BB method was not attempted since the binary transformation of these random nb-formulas is unreasonably long. As discussed at the end of Section 4, if the binary transform is used to produce a CNF b-formula from a positive CNF nb-formula, the size of the resulting b-formula is exponential in the clause length of the original nb-formula—which in this case is $\frac{3}{2}D$.

The results of these experiments, shown in Figure 2, consistently follow a clear pattern. In terms of both time and number of flips, all methods show a decline in performance as domain size grows. However, the decline of UU and UB is so sharp that UB is ineffective on domain size 16 and UU is two orders of magnitude slower than NB on domain size 32. On these random formulas, as on the graph colouring problems, the fliprates of all methods decrease as the domain size increases. However, on the random problems, unlike the colouring problems, NB’s fliprate suffers the least from increasing domain size. Indeed, by the time domain size reaches 8, NB has a higher fliprate than UU, and this advantage grows as domain size increases. The sharp decline in the UU fliprate results from the rapid increase in the size of the UU formulas, which occurs because the size of their AMO component grows quadratically with domain size.

6 Discussion and Conclusions

The literature reports a number of cases where the transformation approach has been used to attack a non-Boolean problem with Boolean SLS. Such problems include the n-queens problem [Selman *et al.*, 1992], graph colouring [Selman *et al.*, 1992; Hoos, 1998, Chp. 4], the all-interval-series problem [Hoos, 1998, Chp. 4], the Hamiltonian circuit problem

⁶A Boolean Davis-Putnam procedure is incapable of solving Boolean encodings of the instances with larger domain sizes!

Problem	method	P_{noise} setting	formula size	flip rate (flips/sec)	median flips	median time (ms)
Domain size 2 60 vars 261 clauses	NB	.586	783	199,000	471	2.36
	UU	.550	903	324,000	3,280	10.1
	UB	.440	1023	422,000	8,040	19.1
Domain size 4 30 vars 280 clauses	NB	.431	1,680	131,000	699	5.36
	UU	.354	2,040	153,000	12,600	82.9
	UB	.156	2,040	262,000	35,800	137
Domain size 8 20 vars 294 clauses	NB	.332	3,528	80,800	1,050	13.0
	UU	.044	4,648	75,900	17,800	235
	UB	.0524	4,160	193,000	431,000	2230
Domain size 16 15 vars 302 clauses	NB	.225	7,248	49,500	3,136	63.4
	UU	.018	10,848	35,600	98,900	2,780
	UB		8,448			
Domain size 32 12 vars 307 clauses	NB	.191	14,736	24,700	5,430	220
	UU	.010	26,640	15,400	599,000	38,900
	UB		17,040			

[Hoos, 1999; 1998, Chp. 7], random instances of the (finite, discrete) constraint satisfaction problem [Hoos, 1999; 1998, Chp. 7], and STRIPS-style planning [Kautz and Selman, 1992; Kautz *et al.*, 1996; Kautz and Selman, 1996; Ernst *et al.*, 1997; Hoos, 1998, Chp. 4]. Yet, in spite of this widespread use of the transformation approach, only two previous studies have compared the performance of SLS on different encodings of non-Boolean variables: Ernst, Millstein and Weld [1997] compare unary/unary and binary encodings of STRIPS-style planning problems and Hoos [1999; 1998, Chapter 7] compares the unary/unary and binary encodings of random constraint satisfaction problems and of Hamiltonian circuit problems. Other than these two comparative studies, we know of no work that uses Boolean encodings other than unary/unary.

Although Hoos' experiments use a clever methodology designed to provide insight into the structure of SLS search spaces, his study examines only problem instances whose non-Boolean formulation involves variables with domain size 10. And though the work of Ernst, Millstein and Weld is also enlightening in many respects, one cannot determine the domain sizes of the non-Boolean variables in their encodings since they do not report the identity of their suite of planning instances. In contrast to all previous studies, ours considers the effect of varying domain size and hence is able to confirm the expectation that domain size is generally a critical parameter. In particular, our study suggests that Hoos' conclusion—which also appears to be the dominant view in the community—that UU is a viable alternative to NB is generally true only for problems with smaller domain sizes. On problems that require AMO formulas and have large domain sizes our study shows that NB tends to be the best choice and is sometimes the only feasible choice.

In almost all work, when non-Boolean problems are encoded as Boolean formulas the encoding is done in a single stage, directly mapping the problem to a Boolean formula. In some cases this is done manually and in others it is done automatically. In contrast, we advocate a two-stage approach in which the problem is first mapped (manually or automatically) to a non-Boolean formula which is then systematically (and, in our case, automatically) transformed to a Boolean formula. We conjecture that all Boolean encodings that have so far been produced by the single stage approach could be

produced by the two stage approach.

It has recently come to our attention that work by Béjar and Manyà [1999] in the area of multi-valued logic has independently led to the development of another generalisation of Walksat, called Regular-WSAT, that appears to be almost identical to NB-Walksat. Regular-WSAT operates on a special class of signed clauses called regular signed clauses. A signed clause is a syntactic variant of an NB-clause except that every variable must be associated with the same domain. To define regular signed clauses, the domain values first must be totally ordered. (In certain other settings, the order is only required to be a lattice.) A signed clause is said to be regular if, in effect, it is positive and for each literal X/d it contains it also contains either every literal X/d' such that $d' < d$ or every literal X/d' such that $d < d'$.

Though Regular-WSAT is presented as a procedure for regular signed clauses, it appears that it would work on any signed clause and thus is akin to NB-Walksat. Nor does there appear to be anything inherent in the presentation of Regular-WSAT that prevents it from handling variables with different domain sizes. However, when operating on a clause containing variables with differing domain sizes it appears that Regular-WSAT and NB-Walksat use different distributions in choosing which literal to flip. Regular-WSAT first chooses a variable from among those in the clause and then chooses one of the values that appear with that variable, whereas NB-Walksat simply chooses from among all the literals in the clause.

Though regular signed clauses are a special class of NB-clauses, they can be used effectively to encode a range of problems including graph colouring problems and round-robin scheduling problems [Béjar and Manyà, 1999]. The extra restrictions placed on regular signed clauses can be exploited to translate them to Boolean clauses more compactly than can NB-clauses [Béjar *et al.*, 2001]. However, we are not aware of any attempt to exploit these translations to solve problems encoded as regular signed clauses.

It would be worthwhile to extend the comparisons made in this paper to include additional solution methods employing SLS. As just suggested, the comparisons could include Boolean translations of regular signed clauses. It would be especially worthwhile to consider SLS methods, such as min-conflicts [Minton *et al.*, 1992], that operate directly on the

Figure 2: Results, to three significant figures, for a suite of 25 satisfiable non-Boolean CNF formulas chosen at random. The values recorded in the flips and time column are the flips and time required to solve a single formula, not the entire suite. On those rows missing entries, the median number of flips to solution of 101 runs at noise levels .01, .02 and .03 all exceeded 5,000,000.

original problem or on a constraint satisfaction encoding of the problem.⁷

Of the many questions remaining to be addressed by future work the biggest challenge facing the study of problem encodings—including all encoding issues, not just the handling of non-Boolean variables—is the quest for generality. What can we say about encoding issues that can guide us in producing effective encodings of new problems? This challenge must be decomposed if progress is to be made. This paper's biggest contribution towards this end is separating out the issue of non-Boolean variables and identifying domain size as a critical parameter.

Finally we claim that, in addition to developing new problem encodings, the applicability of SLS technology can be extended by enriching the language on which the SLS operates. This claim is supported by recent results on pseudo-Boolean constraints [Walser, 1997] and non-CNF formulas [Sebastiani, 1994]. Our success with NB-Walksat adds further weight to the claim.

Acknowledgements

We are grateful to Henry Kautz and Bram Cohen for providing their Walksat code, from which the NB-Walksat implementation was developed. We also thank Peter Stock, who used his non-Boolean Davis-Putnam program to filter out the unsatisfiable instances from our suite of randomly-generated nb-formulas. Thanks to Toby Walsh for many illuminating discussions and to Steve Minton, Bart Selman and the anonymous referees for useful suggestions.

References

- [Béjar and Manyà, 1999] Ramon Béjar and Felip Manyà. Solving combinatorial problems with regular local search algorithms. In Harald Ganzinger and David McAllester, editors, *Proc. 6th Int. Conference on Logic for Programming and Automated Reasoning, LPAR, Tbilisi, Georgia*, volume 1705 of *Lecture Notes in Artificial Intelligence*, pages 33–43. Springer-Verlag, 1999.
- [Béjar et al., 2001] Ramon Béjar, Reiner Hähnle, and Felip Manyà. A modular reduction of regular logic to classical logic. In *Proc. 31st International Symposium on Multiple-Valued Logics, Warsaw, Poland*. IEEE CS Press, Los Alamitos, May 2001.
- [Brafman and Hoos, 1999] Ronen I. Brafman and Holger H. Hoos. To encode or not to encode — I: Linear planning. In *Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence*, pages 988–993. Morgan Kaufman, 1999.
- [Ernst et al., 1997] Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-compilation of planning problems. In *Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence*, pages 1169–1176, 1997.
- [Frisch and Peugniez, 1998] Alan M. Frisch and Timothy J. Peugniez. Solving non-Boolean satisfiability problems with local search. In *Fifth Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, St. Andrews, Scotland, April 1998.
- [Hoos, 1998] Holger H. Hoos. *Stochastic Local Search—Methods, Models, Applications*. PhD thesis, Technical University of Darmstadt, 1998.
- [Hoos, 1999] Holger H. Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence*, pages 296–302. Morgan Kaufman, 1999.
- [Kautz and Selman, 1992] Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *Proc. of the Tenth European Conf. on Artificial Intelligence*, pages 359–363, Vienna, Austria, August 1992. John Wiley & Sons.
- [Kautz and Selman, 1996] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of the Thirteenth National Conf. on Artificial Intelligence*, pages 1202–1207, Portland, Oregon, USA, 1996. AAAI Press / The MIT Press.
- [Kautz et al., 1996] Henry A. Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Fifth Int. Conf.*, pages 374–385, San Francisco, 1996. Morgan Kaufmann.
- [Minton et al., 1992] Steven Minton, Mark D. Johnson, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.
- [Mitchell et al., 1992] David Mitchell, Bart Selman, and Hector J. Levesque. Hard and easy distributions of SAT problems. In *Proc. of the Tenth National Conf. on Artificial Intelligence*, pages 459–465, San Jose, CA, 1992. MIT Press.
- [Peugniez, 1998] Timothy J. Peugniez. Solving non-Boolean satisfiability problems with local search. BSc thesis, Department of Computer Science, University of York, March 1998.
- [Sebastiani, 1994] Roberto Sebastiani. Applying GSAT to non-clausal formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.
- [Selman et al., 1992] Bart Selman, Hector J. Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the Tenth National Conf. on Artificial Intelligence*, pages 440–446. AAAI Press, 1992.
- [Selman et al., 1994] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of the Twelfth National Conf. on Artificial Intelligence*, pages 337–343, Menlo Park, CA, USA, 1994. AAAI Press.
- [Walser, 1997] Joachim P. Walser. Solving linear pseudo-Boolean constraint problems with local search. In *Proc. of the Fourteenth National Conf. on Artificial Intelligence*, pages 269–274. AAAI Press, 1997.

⁷Brafman and Hoos [1999] have performed a comparison of this type for linear planning problems.