# Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings

**Article** *in* Journal of Automated Reasoning · January 2005

DOI: 10.1007/s10817-005-9011-0 · Source: DBLP



**4 authors**, including:

Alan M. Frisch
The University of York
**121** PUBLICATIONS **1,881** CITATIONS

SEE PROFILE

# Solving Non-Boolean Satisfiability Problems with Stochastic Local Search: A Comparison of Encodings

ALAN M. FRISCH [†], TIMOTHY J. PEUGNIEZ,
ANTHONY J. DOGGETT and PETER W. NIGHTINGALE[§]
*Artificial Intelligence Group, Department of Computer Science, University of York, York YO10 5DD, UK. e-mail: frisch@cs.york.ac.uk, pn@dcs.st-and.ac.uk*

**Abstract.** Much excitement has been generated by the success of stochastic local search procedures at finding solutions to large, very hard satisfiability problems. Many of the problems on which these procedures have been effective are non-Boolean in that they are most naturally formulated in terms of variables with domain sizes greater than two. Approaches to solving non-Boolean satisfiability problems fall into two categories. In the direct approach, the problem is tackled by an algorithm for non-Boolean problems. In the transformation approach, the non-Boolean problem is reformulated as an equivalent Boolean problem and then a Boolean solver is used.

This paper compares four methods for solving non-Boolean problems: one direct and three transformational. The comparison first examines the search spaces confronted by the four methods then tests their ability to solve random formulas, the round-robin sports scheduling problem and the quasigroup completion problem. The experiments show that the relative performance of the methods depends on the domain size of the problem, and that the direct method scales better as domain size increases.

Along the route to performing these comparisons we make three other contributions. First, we generalise Walksat, a highly-successful stochastic local search procedure for Boolean satisfiability problems, to work on problems with domains of any finite size. Second, we introduce a new method for transforming non-Boolean problems to Boolean problems and improve on an existing transformation. Third, we identify sufficient conditions for omitting at-least-one and at-most-one clauses from a transformed formula. Fourth, for use in our experiments we propose a model for generating random formulas that vary in domain size but are similar in other respects.

**Keywords:** satisfiability, propositional logic, local search, encodings

## 1. Introduction

Much excitement has been generated by the success of stochastic local search (SLS) procedures at finding satisfying truth assignments to large formulas of propositional logic. These procedures stochasticly search

[†] Author for correspondence.
[§] Current address: School of Computer Science, University of St Andrews, Fife KY16 9SX, UK.

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 (SAT) 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 (NB-SAT) 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 Walksat (Selman et al., 1994), a highly-successful SLS procedure for Boolean satisfiability problems, to a new procedure, NB-Walksat (first reported by Peugniez, 1998 and by 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 random non-Boolean formulas, the round-robin tournament scheduling problem, and the quasi-group completion problem. 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.

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

We shall also see that the polarity of a non-Boolean formula—whether it is positive, negative or neither, as defined later—is another significant factor that affects its translation to a Boolean formula.

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 many cases are ineffective.

Besides introducing the generalisation of Boolean formulas to non-Boolean formulas and Walksat to NB-Walksat, we make several other new contributions, including the following three. (1) Of the three non-Boolean to Boolean transformations we use, one is new and one is an enhanced version of a well-known transformation. (2) We identify sufficient conditions for omitting at-least-one and at-most-one clauses from a transformed formula. (3) 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 propose such a method and use it in our experiments.

We conjecture that the transformation of non-Boolean SAT to Boolean SAT is an inherent component of using the encode-and-solve approach on any problem that is conceived of as having non-Boolean domains. More specifically, we put forward a hypothesis.

> *The SAT-Transform Hypothesis:* Let $P$ be a problem that we conceive of as having variables with finite domains of more than two elements. Let $T$ be a SAT-effective transform for $P$; that is, $P$ can be solved effectively by using a SAT solver on the result of applying transform $T$ to $P$. Then $T$ can be obtained by composing a transform from $P$ to NB-SAT with a transform from NB-SAT to SAT.

Note that this is an empirical hypothesis and that it could be refuted by identifying a SAT-effective transform that cannot be decomposed in the stated manner. Such a transform might, for example, exploit some complex interaction between the encoding of the non-Boolean variables and the encoding of some other aspect of the problem. But in all uses of encode-and-solve known to us the hypothesis does hold, and thus every use embeds a transform from NB-SAT to SAT. This underscores the importance of studying the transformation of NB-SAT to SAT, as well as the alternative of generalising SAT solvers to work directly on NB-SAT.

## 2. Non-Boolean Formulas

Syntactically, non-Boolean formulas are constructed from propositional variables, each of which is associated with a finite, non-empty domain. We refer to the domain of a variable $X$ as $dom(X)$. 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. Non-atomic non-Boolean formulas are constructed from atomic non-Boolean formulas with logical connectives in precisely the same manner as is used for Boolean formulas. As an example, if $X$ and $Y$ are variables both with domain $\{d_1, d_2, d_3\}$, then

$$X/d_1 \wedge (Y/d_2 \vee Y/d_3) \tag{1}$$

in a non-Boolean formula

Now consider the semantics of non-Boolean formulas. A non-Boolean assignment maps every variable to a member of its domain. A non-Boolean assignment, $A$, satisfies an atomic non-Boolean formula $X/d$ if and only if $A$ maps $X$ to $d$. The satisfaction of non-atomic non-Boolean formulas is determined from the satisfaction of atomic components in precisely the same manner as for Boolean formulas. So, an assignment that maps $X$ to $d_1$ and $Y$ to $d_3$ satisfies $X/1$, $Y/3$ and formula (1).

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 disjunction 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. Thus, formula (1) is in CNF and it is positive.

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

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 clausal form. We have generalised Walksat to a new procedure, NB-Walksat, that operates similarly on non-Boolean formulas. Indeed when handling a Boolean formula the

two procedures perform the same search.[2] NB-Walksat was implemented by replacing the code of the core search procedure of Walksat version 19. Obtaining this generality required a complete reworking of the data structures that maintain formulas and assignments. This section describes the operation of NB-Walksat[3] 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 $X/d_1 \vee \cdots \vee X/d_{i-1} \vee X/d_{i+1} \vee \cdots \vee X/d_n$, where $X$ is a variable with domain $\{d_1, \ldots, d_n\}$.

Like many other SLS procedures for satisfiability, 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, and hence the clause in which it appears. 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 variables 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 choosing which literal to flip. Though recent versions of Walksat provide a range of user-selectable heuristics for choosing the literal, the original heuristic is the one called "best" or "SKC." As it has been used in many reported experiments (e.g., Selman et al., 1994; Kautz et al., 1997; Walser, 1997) it is the "best" version of Walksat that forms the basis for NB-Walksat and is the focus of this paper.

NB-Walksat with the "best" 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. Let $L$ be the set of literals in the selected clause. 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 $L$ 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 $L$ contains no such literals, then a literal is chosen either (i) randomly with

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

[3] The description applies to NB-Walksat versions 4, 5 and 6. Version 6 is the most recent version at the time of writing this paper. Versions 1, 2 and 3 computed probability distributions in a subtly-different way—a difference that affects only some problem instances.

Input: $F$, a Non-Boolean formula in CNF; $P_{noise}$, the noise probability setting.
Output: $A$, an assignment that satisfies $F$.

(a) Transform $F$ into a positive CNF formula.
(b) From among all assignments to the variables of $F$ choose one at random with uniform distribution. Call it $A$.
(c) Repeat until $A$ satisfies $F$:
  (d) From among the clauses of $F$ that are not satisfied by $A$ select one at random with a uniform distribution. Let $L$ be the set of literals of this clause.
  (e) If there is a literal in $L$ such that flipping it would break no clauses then set $L$ to the set of all such literals in $L$.
  (f) Otherwise, with probability $1 - P_{noise}$ remove from $L$ all literals that, if flipped, would break more clauses than would flipping another literal of $L$.
  (g) From among the literals of $L$ select one at random with uniform distribution. Call it $X/d$.
  (h) Modify $A$ by flipping $X/d$.
(i) Output $A$.

*Figure 1.* NB-Walksat with the "best" heuristic.

uniform distribution from $L$ or (ii) randomly with uniform distribution from among the literals in $L$ 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. Figure 1 gives pseudo-code for the NB-Walksat procedure.

Experiments with Walksat (Selman et al., 1994). show that the incorporation of noisy choices dramatically improves its performance and that performance can vary greatly according to the value of $P_{noise}$. As one would expect, the same is true of NB-Walksat.

## 4. Transforming Non-Boolean Formulas

To transform NB-SAT to SAT we map each nb-formula to a b-formula such that the satisfying assignments of the two formulas correspond, though not necessarily one to one. 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 nb-atom in the formula with a b-formula that, in a sense, encodes the nb-atom it replaces. The resulting formula is known as the

| | Unary/Unary | Enhanced Binary | | Unary/Binary |
|---|---|---|---|---|
| | | Positive | Negative | |
| kernel variables | $VD$ | $V\lceil \lg D \rceil$ | $V\lceil \lg D \rceil$ | $VD$ |
| kernel size | $L$ | $\leq L\lceil \lg D \rceil$ | $\leq L\lceil \lg D \rceil$ | $L$ |
| kernel CNF size | $L$ | $\leq L\lceil \lg D \rceil^J$ | $\leq L\lceil \lg D \rceil$ | $L$ |
| at-least-one variables | $VD$ | 0 | 0 | $V(D + \lceil \lg D \rceil)$ |
| at-least-one size | $VD$ | 0 | 0 | $\leq VD(\lceil \lg D \rceil + 1)$ |
| at-least-one CNF size | $VD$ | 0 | 0 | $\leq VD(\lceil \lg D \rceil + 1)$ |
| at-most-one variables | $VD$ | 0 | 0 | $V(D + \lceil \lg D \rceil)$ |
| at-most-one size | $VD(D-1)$ | 0 | 0 | $\leq VD(\lceil \lg D \rceil + 1)$ |
| at-most-one CNF size | $VD(D-1)$ | 0 | 0 | $\leq 2VD\lceil \lg D \rceil$ |

*Figure 2.* Size of the Boolean formulas produced by each of the three transformations applied to a non-Boolean formula of size $L$ that has $V$ variables, each with domain size $D$. The "CNF size" rows give the size of the formula when put into CNF form. It is assumed that the non-Boolean formula is in CNF and its clauses each have $J$ literals. The size of the enhanced binary transformation is divided into two cases: a positive kernel and a negative kernal.

*kernel* of the transformation. The transforms employ two core 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 *enhanced 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 an enhanced binary encoding for the ALO and AMO formulas). The unary/binary transform is new, as is the enhanced version of the binary transform.

In the following presentation we discuss the size of the formula produced by each transform, where we take a formula's size to be the number of occurrences of atoms it contains. Throughout we assume that each transform is being applied to an nb-formula containing $V$ variables each of which has a domain of size $D$, and we present the sizes of the resulting formulas in terms of these two parameters. The results of the discussion are summarised in Figure 2.

### 4.1. 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 $\{d_1, \ldots, d_n\}$ 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 $X{:}d_1, \ldots, X{:}d_n$. This constraint is expressed as a conjunction of one formula (known as the ALO formula) asserting that at least one of the variables is true and another (known as the AMO formula) asserting that at most one of the variables is true.

To state that at *least* one of $X{:}d_1, \ldots, X{:}d_n$ must be satisfied we simply use the clause $X{:}d_1 \vee \cdots \vee X{:}d_n$. The entire ALO formula is a conjunction of such clauses, one clause for each nb-variable. Thus, the ALO formula consists of a conjunction of $V$ clauses each with $D$ literals, giving it a total size of $VD$.

To say that at *most* one of $X{:}d_1, \ldots, X{:}d_n$ 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 such clauses, $\frac{1}{2}D(D-1)$ clauses for each nb-variable. Thus, the AMO formula consists of a conjunction of $\frac{1}{2}VD(D-1)$ clauses, each containing 2 literals, giving it a total size of $VD(D-1)$.

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.

### 4.2. 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 $\lceil \lg D \rceil$ b-variables to encode the same nb-variable.[4] If $X$ is a variable with domain $\{d_1, \ldots, d_n\}$, 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 $\lceil \lg n \rceil$ Boolean variables. For example, if $n$ is 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

[4] The ceiling of a real value $x$, written $\lceil x \rceil$, is the smallest integer that is greater than or equal to $x$.

$X/d_3$ 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 would be to add an ALO formula,

$$(\neg X_2 \wedge \neg X_1) \vee (\neg X_2 \wedge X_1) \vee (X_2 \wedge X_1),$$

which ensures that the *extraneous* binary combination $X_2 \wedge X_1$ cannot be satisfied in any solution. Alternatively, one could make the logically-equivalent statement that the extraneous binary combination must be false: $\neg(X_2 \wedge X_1)$. The latter of these two has the advantage that it can be put into CNF without any blowup, and is the method adopted by Hoos (1998, page 180).

Frisch and Peugniez (2001) introduced a version of the binary transform in which no extraneous combinations are produced and therefore no ALO formula is required. We call this the *enhanced binary transform*, and the version with extraneous combinations the *basic binary transform*. We use the term *binary transform* to refer generically to any version of the transform.

In the example considered above 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 $\{d_1, \ldots, d_n\}$ and let $k$ be $2^{\lceil \lg n \rceil} - n$. Then $X/d_1, \ldots, X/d_k$ are each mapped to cover two binary combinations and $X/d_{k+1}, \ldots, X/d_n$ 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, and thus is identical to the basic binary transform. Finally, to confirm that the extended binary transform requires no AMO formula and no ALO formula, observe that every Boolean assignment must satisfy the extended binary transform of exactly one of $X/d_1, \ldots, X/d_n$.

Since the enhanced binary transform replaces each nb-atom with a conjunction of at least $\lfloor \lg D \rfloor$ b-atoms and at most $\lceil \lg D \rceil$ b-atoms, it produces a formula whose size is $\lfloor \lg D \rfloor$ to $\lceil \lg D \rceil$ times that of the original formula.[5]

[5] The floor of a real value $x$, written $\lfloor x \rfloor$, is the largest integer that is less than or equal to $x$.

Notice that the enhanced binary transformation of a CNF formula is not necessarily in CNF. However, the enhanced binary transformation of a negative CNF formula is *almost* in CNF; it is a conjunction of disjunctions of negated conjunctions of literals. For example, using variables $X$ and $Y$, both with domain $\{d_1, d_2, d_3\}$, the enhanced binary transform of the clause $\neg X/d_1 \vee \neg Y/d_2$ is $\neg(\neg X_2) \vee \neg(Y_2 \wedge \neg Y_1)$. By using De Morgan's law, the negations can be moved inside of the innermost conjunctions, resulting in a CNF formula of the same size. Thus, our example formula becomes the clause $X_2 \vee \neg Y_2 \vee Y_1$. At the other extreme, the enhanced 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 to a disjunction of $n$ conjunctions, each with $m$ literals, produces a CNF formula with $n^m$ conjuncts, each with $m$ literals. Thus, if an nb-clause has $J$ literals, its enhanced binary transformation consists of between $\lfloor \lg D \rfloor^J$ and $\lceil \lg D \rceil^J$ clauses, each with $J$ literals. Thus, if a positive nb-formula consists of $L/J$ clauses each with $J$ literals, then the size of its enhanced binary transform is between $L\lfloor \lg D \rfloor^J$ and $L\lceil \lg D \rceil^J$.

It is possible to avoid this exponential expansion by introducing new variables into the formula and, indeed, this is what is generated by the unary/binary transform to which we now turn our attention.

### 4.3. The Unary/Binary Transform

The unary/binary transform, originally introduced by Frisch and Peugniez (2001), produces the same kernel as the unary/unary transform. The ALO and AMO formulas it produces achieve their effect by introducing the enhanced binary encodings of nb-atoms and adding formulas linking the two encodings together. Following the practice of the constraint programming community, we call these linking formulas "channeling" formulas. Since the enhanced binary encoding requires no ALO or AMO formulas, the unary/binary encoding requires no ALO or AMO formulas beyond the channelling formulas.

The channelling formulas that act as AMO formulas state that the unary encoding of each nb-atom implies its enhanced binary encoding. So, for example, if the nb-variable $X$ has domain $\{d_1, d_2, d_3\}$ then the AMO formula for $X$ is

$$\begin{array}{lll} (X{:}d_1 & \rightarrow & \neg X_2) \qquad\qquad \wedge \\ (X{:}d_2 & \rightarrow & (X_2 \wedge \neg X_1)) \ \wedge \\ (X{:}d_3 & \rightarrow & (X_2 \wedge X_1)), \end{array}$$

which is logically equivalent to the CNF formula

$$\begin{array}{l} (\neg X{:}d_1 \vee \neg X_2) \ \wedge \\ (\neg X{:}d_2 \vee X_2) \quad \wedge \ (\neg X{:}d_2 \vee \neg X_1) \ \wedge \\ (\neg X{:}d_3 \vee X_2) \quad \wedge \ (\neg X{:}d_3 \vee X_1). \end{array}$$

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

It is easy to see that the CNF of the channelling formula for each variable consists of between $D\lfloor \lg D \rfloor$ and $D\lceil \lg D \rceil$ clauses of two literals. Since the entire AMO formula consists of a conjunction of channelling formulas for each of $V$ variables, its total size is between $2VD\lfloor \lg D \rfloor$ and $2VD\lceil \lg D \rceil$.

The channelling formulas that act as ALO formulas state that for each nb-atom, its unary encoding is implied by its enhanced binary encoding. So, for example, if the nb-variable $X$ has domain $\{d_1, d_2, d_3\}$ then the ALO formula for $X$ is

$$\begin{array}{l} (\neg X_2 \rightarrow X{:}d_1) \qquad\quad \wedge \\ ((X_2 \wedge \neg X_1) \rightarrow X{:}d_2) \ \wedge \\ ((X_2 \wedge X_1) \rightarrow X{:}d_3). \end{array}$$

which is logically equivalent to the CNF formula

$$\begin{array}{l} (X_2 \vee X/d_1) \qquad\quad \wedge \\ (\neg X_2 \vee X_1 \vee X{:}d_2) \quad \wedge \\ (\neg X_2 \vee \neg X_1 \vee X{:}d_3). \end{array}$$

The entire ALO formula is a conjunction of channelling formulas, one linking formula for each nb-variable. The ALO formula for each nb-variable is a conjunction of $D$ clauses each of size $\lfloor \lg D \rfloor + 1$ or $\lceil \lg D \rceil + 1$; thus the entire ALO formula is a conjunction of $DV$ clauses and has as a total size of between $DV(\lfloor \lg D \rfloor + 1)$ and $DV(\lceil \lg D \rceil + 1)$.

### 4.4. When Are ALO and AMO Formulas Needed?

It has been known for some time that certain unary SAT-encodings do not require ALO clauses and certain others do not require AMO clauses. For example, Jonsson and Ginsberg (1993) argue that AMO clauses are not needed in graph colouring. It has also been observed that when it is possible to omit either the AMO or ALO clauses, doing so improves the performance of local search algorithms. Prestwich (2004) hypothesises that this improvement is partly a result of increasing the solution density of the search space. The important open question is when can ALO and AMO clauses be omitted?

This section shows that the need for such clauses is *not* a property of the problem, but rather a property of the non-Boolean encoding of the problem. This section identifies, and proves correct, a sufficient syntactic condition for excluding ALO clauses and another for excluding AMO clauses. In fact, we shall state this on a variable-by-variable basis; some variables may require ALO and/or AMO clauses while others may not.

The following definition applies to all formulas, both Boolean and non-Boolean.

**Definition 1** (Positive and Negative Formulas)**.** *A formula occurs positively within itself. If $\alpha$ occurs positively (negatively) within $\gamma$ then $\alpha$ occurs positively (negatively) within $\gamma \wedge \beta$, $\beta \wedge \gamma$, $\beta \vee \gamma$, $\gamma \vee \beta$, $\beta \rightarrow \gamma$, $\gamma \leftrightarrow \beta$ and $\beta \leftrightarrow \gamma$. If $\alpha$ occurs positively (negatively) within $\gamma$ then $\alpha$ occurs negatively (positively) within $\neg\gamma$, $\gamma \rightarrow \beta$, $\gamma \leftrightarrow \beta$ and $\beta \leftrightarrow \gamma$. A formula is said to be negative (positive) with respect to an atom if that atom does not occur positively (negatively) in the formula. A formula is said to be positive (negative) if it is positive (negative) with respect to all atoms.*

Notice that this generalises our previous definition that a CNF formula is negative if it contains no positive literals and it is positive if it contains no negative literals.

We can partially order the Boolean assignments by the atoms that they satisfy. If $A$ is an atom then we write $\alpha \geq_A \beta$ to mean that $\alpha$ and $\beta$ are identical with the possible exception that $\alpha$ satisfies $A$ but $\beta$ does not.

**Lemma 1** (Monotonicity)**.** *Let $\phi$ be a formula, let $A$ be an atom and let $\alpha$ and $\beta$ be two Boolean assignments such that $\alpha \geq_A \beta$. If $\phi$ is formula that is positive with respect to $A$ and is satisfied by $\beta$, then it is satisfied by $\alpha$. If $\phi$ is negative with respect to $A$ and is satisfied by $\alpha$, then it is satisfied by $\beta$.*

*Proof.* Both statements can be proved simultaneously by a straightforward induction on the structure of $\phi$. □

We now turn our attention to the main theorem, which identifies conditions under which AMO and ALO formulas can be omitted without affecting satisfiability. The correctness of the theorem depends on the semantics, not the syntax, of the ALO and AMO formulas. In particular, all that matters is that an ALO (AMO) formula for $X{:}d_1, \ldots, X{:}d_n$ is satisfied by an assignment if and only if at least (most) one of $X{:}d_1, \ldots, X{:}d_n$ is satisfied by that assignment.

**Theorem 1** (Satisfiability without ALO or AMO)**.** *Let $K$ be an arbitrary Boolean formula, $L$ be a conjunction of ALO formulas and $M$ be a conjunction of AMO formulas. Let $AMO(X)$ and $ALO(X)$ be AMO and ALO formulas, respectively, for $X{:}d_1, \ldots, X{:}d_n$. (1) If $K$ is negative with respect to each of $X{:}d_1, \ldots, X{:}d_n$ and $K \wedge L \wedge M$ is satisfiable then so is $K \wedge L \wedge M \wedge AMO(X)$. (2) If $K$ is positive with respect to each of $X{:}d_1, \ldots, X{:}d_n$ and $K \wedge L \wedge M$ is satisfiable then so is $K \wedge L \wedge M \wedge ALO(X)$.*

*Proof.* Parts (1) and (2) are proved separateley.

(1) The proof proceeds by assuming the antecedent and proving the consequent. Let $\alpha$ be an assignment that satisfies $K \wedge L \wedge M$. We now prove that $K \wedge L \wedge M \wedge AMO(X)$ is satisfiable by induction on $m$, the number of atoms in $X{:}d_1, \ldots, X{:}d_n$ that are satisfied by $\alpha$.

*Base case, $m$ is 0 or 1*: In this case $\alpha$ itself satisfies $K \wedge L \wedge M \wedge AMO(X)$.

*Inductive case, $m \geq 2$*: The inductive hypothesis is that if $K$ is negative with respect to each of $X{:}d_1, \ldots, X{:}d_n$ and $K \wedge L \wedge M$ is satisfied by an assignment that satisfies $m-1$ of $X{:}d_1, \ldots, X{:}d_n$ then $K \wedge L \wedge M \wedge AMO(X)$ is satisfiable. Let $X{:}d_i$ be any one of the $m$ atoms of $X{:}d_1, \ldots, X{:}d_n$ that are satisfied by $\alpha$. Let $\alpha'$ be an assignment that is identical to $\alpha$ except that it falsifies $X{:}d_i$. We now show, in turn, that $\alpha'$ satisfies $K$, $L$ and $M$. Since $K$ is negative with respect to $X{:}d_i$ and $\alpha$ satisfies $K$, by the Monotonicity Lemma $\alpha'$ also satisfies $K$. Secondly, $L$ must be satisfied by $\alpha'$; the truth of an ALO formula for a variable other than $X$ is unaffected by the change of $X{:}d_i$ and we have constructed $\alpha'$ so that it satisfies $m-1 \geq 1$ of $X{:}d_1, \ldots, X{:}d_n$. Finally, $\alpha'$ satisfies $M$ since $\alpha$ satisfies $M$ and $\alpha'$ satisfies fewer atoms than $\alpha$. Since $\alpha'$ satisfies $K \wedge L \wedge M$ and $m-1$ of $X{:}d_1, \ldots, X{:}d_n$ then, by the inductive hypothesis, $K \wedge L \wedge M \wedge AMO(X)$ is satisfiable.

(2) We assume the antecedent and prove the consequent. Let $\alpha$ be an assignment that satisfies $K \wedge L \wedge M$. If $\alpha$ also satisfies $ALO(X)$ then the consequent trivially holds. Otherwise, $\alpha$ doesn't satisfy $ALO(X)$; rather it falsifies each of $X{:}d_1, \ldots, X{:}d_n$. Let $X{:}d_i$ be any one of $X{:}d_1, \ldots, X{:}d_n$ and let $\alpha'$ be an assignment that is identical to $\alpha$ except that it satisfies $X{:}d_i$. Clearly $\alpha'$ satisfies $ALO(X)$, so it remains to show that $\alpha'$ satisfies $K$, $L$ and $M$, which we do in turn. Since $K$ is positive with respect to $X{:}d_i$ and $\alpha$ satisfies $K$, by the Monotonicity Lemma $\alpha'$ also satisfies $K$. Secondly, $\alpha'$ satisfies $L$ since $\alpha$ satisfies $L$ and $\alpha'$ satisfies more atoms than $\alpha$. Finally, $M$ must be satisfied by $\alpha'$; the truth of an AMO formula for a variable other than $X$ is unaffected by the change of $X{:}d_i$ and we have constructed $\alpha'$ so that it satisfies exactly one of $X{:}d_1, \ldots, X{:}d_n$. □

**Corollary 1** (Unary Transform without ALO or AMO)**.** *The unary translation of a negative NB-formula is satisfiable if and only if it is satisfiable when the AMO clauses are omitted. The unary translation of a positive NB-formula is satisfiable if and only if it is satisfiable when the ALO clauses are omitted.*

*Proof.* We prove the first statement; the second is analagous. Let $\alpha$ be the unary translation of an arbitrary negative NB-formula from which $n$ AMO formulas have been omitted. The "only-if" part is obvious. The "if" part is proved by induction on $n$. For the base case, if $n = 0$ then the corollary is obvious. The inductive case follows from Theorem 1 by taking $K$ to be the kernal of $\alpha$, $L$ and $M$ to be the ALO formulas and AMO formulas of $\alpha$, and $AMO(X)$ to be one of the AMO formulas omitted from $\alpha$. If $\alpha$ is satisfiable then, by the theorem, it remains satisfiable if we conjoin it with $AMO(X)$. □

To see that the need for AMO and ALO clauses is solely a property of the encoding, not the problem being encoded, consider the problem of colouring a graph of two connected nodes with the colours red, blue and green. We can encode this in NB-SAT by using two variables, $X$ and $Y$, for the nodes and $\{red, blue, green\}$ as the domain of each. There are (at least) two ways to encode the constraint that both nodes cannot be red.

$$\neg X/red \vee \neg Y/red \tag{2}$$

$$X/blue \vee X/green \vee Y/blue \vee Y/green \tag{3}$$

According to Corollary 1, AMO clauses are not needed with (2) and ALO clauses are not needed with (3).

### 4.5. Mixed Transformations

In presenting the three transformations it was assumed that the same transformation is applied to all nb-variables. However, there is no reason why different transformations couldn't be applied to different variables. Nor is there any reason why the AMO formula for a variable couldn't use one encoding (say, binary) while its ALO formula uses another encoding (say, unary).

This flexibility extends to the issue of whether ALO and AMO formulas are required in a unary transformation. Observe that Theorem 1 applies to the translation of a single nb-variable, $X$. Thus the need for ALO and AMO formulas can be considered on a variable-by-variable basis.

## 5. Comparison of Search Spaces

This section considers the search spaces confronted by the four solution methods: the direct non-Boolean method and the three transformation methods based on the transforms of the previous section. For each method, the states in the search space are all assignments to the variables of the formula—Boolean or non-Boolean—and the state transitions are made by flipping a single literal.

Consider an nb-formula $F$ containing $V$ variables, each with a domain of size $D$. The search space consists of $D^V$ states with $V(D-1)$ transitions from each.

The unary/unary transformation of $F$ contains $DV$ b-variables, and hence has $2^{DV}$ states with $DV$ transitions from each. An nb-variable $X$ with domain $\{d_1, \ldots, d_n\}$ is represented in the unary/unary transformation by the b-atoms $X{:}d_1, \ldots, X{:}d_n$. A non-Boolean assignment that maps $X$ to $d_i$ corresponds to a Boolean assignment that maps $X{:}d_j$ to *TRUE* if $j = i$ and *FALSE* if $j \neq i$. In a Boolean assignment, such as this, where exactly one $X{:}d_j$ is mapped to *TRUE* we say that $X$ is *singularly* assigned. *Non-singular* assignments of $X$ are either *empty*—mapping every $X{:}d_j$ to *FALSE*—or *multiple*—mapping more than one $X{:}d_j$ to *TRUE*. We also use the term "singular" to describe an entire assignment in which every nb-variable is singularly assigned.

Though non-singular assignments occur in the unary/unary search space, they cannot be solutions if all AMO and ALO clauses are included in the encoding. As domain sizes grow the unary/unary search space becomes dominated by non-singular assignments. A variable with domain size $D$ has $D$ singular assignments compared with $2^D - D$ non-singular assignments. In a problem with $V$ nb-variables the ratio of non-singular to singular assignments is raised to the power of $V$.

Consider a transition from state $S$ to state $S'$ in in the search space of $F$. This transition changes the value assigned to some variable, $X$, from value $d$ to a different value $d'$. Both $S$ and $S'$ correspond to singular states in the search space of the unary/unary transform of $F$. However, the latter search space contains no transition between these two states. The shortest paths between these two states contain two moves: flipping $X{:}d$ from *FALSE* to *TRUE* and flipping $X{:}d'$ from *TRUE* to *FALSE* in either order. A local search procedure operating on the unary/unary transformation must inevitably move through some non-singular states because there are no transitions between two singular states.

Much work on SLS has noted that solution density (the ratio of number of solutions to number of states in the search space) is one factor influencing the effectiveness of SLS. An nb-encoding has the same number of solutions as its unary/unary transform if all ALO and AMO

clauses are included. As the unary/unary encoding generally has many more states, it generally has a lower solution density. Removing ALO or AMO clauses from a unary/unary encoding potentially increases the number of solutions without changing the number of states.

The binary transformation of $F$ contains $\lceil \lg D \rceil V$ variables, and hence has $2^{\lceil \lg D \rceil V}$ states; from each there are $\lceil \lg D \rceil V$ transitions. If $D$ is a power of 2 then the non-Boolean states and binary states are in a one-to-one correspondence and, hence, contain the same number of solutions, same number of states, and same solution density. However, the $\lceil \lg D \rceil V$ transitions from each binary state are a subset of the $(D-1)V$ transitions in the corresponding nb-state. If $D$ is not a power of two, then the binary search space has more states than the nb-search space. The basic binary encoding has the same number of solutions as the nb-encoding, but the extended binary encoding potentially has more solutions.

Notice that if all variables in $F$ have a domain size of 2 then the binary transform of $F$ is essentially the same as $F$ and the non-Boolean and binary search spaces are isomorphic. NB-Walksat operating on $F$ behaves identically to Walksat operating on the binary transform of $F$. This equivalence was exploited in testing the correctness of the NB-Walksat implementation.

The unary/binary transformation of $F$ contains both the variables produced by the unary/unary transform and those produced by the binary transform, a total of $DV + \lceil \lg D \rceil V$ variables. Its search space is a cross product of the other two search spaces. More precisely, if we let $S_U$ be the states of the unary/unary space and $S_B$ be the states of the binary space, then the states of the unary/binary space are the cross product of $S_U$ and $S_B$. If $u$ and $u'$ are elements of $S_U$ and $b$ and $b'$ are elements of $S_B$ then there is a transition from $\langle u, b \rangle$ to $\langle u', b' \rangle$ in the unary/binary space if and only if either (1) $b = b'$ and there is a transition from $u$ to $u'$ in the unary/unary space, or (2) $u = u'$ and there is a transition from $b$ to $b'$ in the binary space.

This discussion raises two questions: Are non-singular states helpful to the search, perhaps by providing useful paths to a solution or out of local minima, or is better performance achieved by restricting search to a smaller space containing only singular assignments? Does the reduction of transitions that results from using the binary representation help or hinder the search process?

## 6. Performance Evaluation

Using four problem domains, this section presents experiments that compare the performance of the four methods, which we shall refer to as NB (non-Boolean encoding), UU (unary/unary encoding), EB (enhanced binary encoding), and UB (unary/binary encoding). In all experiments, Walksat version 35 was used to solve the Boolean encodings. The non-Boolean encodings, even in cases where the domain size is 2, were solved with NB-Walksat; version 4 was used for the graph colouring problems in Section 6.1 and version 6 was used in all other experiments. Both Walksat and NB-Walksat 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. The random formulas (Section 6.2) and round-robin tournament problems (Section 6.3) were run on a 700Mhz Athlon with 256 megabytes of memory. The quasigroup problems (Section 6.4) were run on an Athlon XP 2400+ 2GHz with 512Mb 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, in another formula, yield performance that is several orders of magnitude below optimal. 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 common practice to restart the search at a new, randomly-selected assignment if the procedure has not found a solution after a prescribed number of flips. Since the runtime distribution using a restart strategy is a function of the runtime distribution without restarts, this study need be concerned only with the performance without restarts.

### 6.1. Graph Colouring

Frisch and Peugniez (2001) report experiments with 6 instances of the graph colouring problem. 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 a pair of nodes connected by an arc and $c$ is a colour. For each

instance the domain size of the variables is the number of colours, which for these six instances is 5, 5, 15, 17, 18 and 25.

Three SAT-encodings of each problem instance were produced by applying the unary/unary, enhanced binary and unary/binary transforms. Since the nb-encodings are negative, AMO clauses were omitted from the UU and UB encodings, as justified by Corollary 1. Also note that because the nb-encodings are negative, the enhanced binary transform maps each nb-clause to a single b-clause (as discussed at the end of Section 4.2).

Frisch and Peugniez's experiments reveal that NB and UU are effective on all six instances and have roughly comparable solution times (within a factor of 2 to 3). On the two problem instances with domain size five, the three transformation methods equaled or outperformed the direct method. However, on each of the other four instances (domain size 15 to 25), the direct method equaled or bettered each of the transformation methods. Of these same four instances, UB was ineffective on three and EB was ineffective on two.

Overall, their graph-colouring experiments show that with increasing domain size, NB scales much better than both UB and EB and slightly better than UU.

### 6.2. 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 stubborn 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 (with uniform distribution) chosing $V$ distinct variables and then, for each, randomly (with uniform distribution) choosing $L$ 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 see that performance on this task would exhibit typical phase-

transition behaviour (Mitchell et al., 1992): small values of $D$ would produce under-constrained instances, which would become critically-constrained and then over-constrained as $D$ increases. The problem instances generated by this method would not be similar in terms of their location relative the solubility 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 solubility 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.

Our first attempt to solve this was based on the idea of keeping the problem size constant. With each formula consisting of 1000 clauses, each with three literals, we experimentally determined the appropriate value of $N$ to put the problem class at the phase transition. As we later discovered, this approach is faulty, a consequence of the somewhat counterintuitive observation that the appropriate value of $N$ grows fairly rapidly with $D$. A problem instance with 1000 clauses of three literals is at the phase transition if it has 1031 variables with domain size 64. Such a problem instance contains only 3000 literal occurrences drawn out of the possible 65,984 atoms ($1031 \times 64$). That is, only about one in 21 of the possible atoms occur in the formula; so, on the average, each variable occurs with only about 3 of its 64 possible values. Since the problem instance has all positive literals, each variable has, in effect, a domain size of approximately 3. Even for a problem with domain size 8, about half of the possible atoms do not occur in a random problem instance.

The solution we have adopted is to keep $D^N$, 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 $D/2$. This follows the constant length model advocated by Mitchell et al. (1992). Finally $C$ is set to whatever value puts the problem class at the solubility phase transition. 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 $D^N$ at a constant value; $2^{60}$ was chosen since $2^{60} = 2^{N_2} = 4^{N_4} = 8^{N_8} = 16^{N_{16}} = 32^{N_{32}}$ if $N_2 = 60$, $N_4 = 30$, $N_8 = 20$, $N_{16} = 15$ and $N_{32} = 12$. Then for each domain size,

| Problem | method | $P_{noise}$ setting | formula size | flip rate (flips/sec) | median flips | median time (ms) |
|---|---|---|---|---|---|---|
| Domain size 2 | NB | .53 | 783 | 272,000 | 506 | 1.9 |
| 60 vars | UU | .44 | 903 | 684,000 | 3,450 | 5.0 |
| 261 clauses | UB | .39 | 1,023 | 792,000 | 9,560 | 12.1 |
| | EB | .53 | 783 | 443,000 | 502 | 1.1 |
| Domain size 4 | NB | .42 | 1,680 | 179,000 | 803 | 4.5 |
| 30 vars | UU | .17 | 2,040 | 414,000 | 8,770 | 21.2 |
| 280 clauses | UB | .16 | 2,160 | 608,000 | 44,600 | 73.4 |
| | EB | | 319,448 | | | |
| Domain size 8 | NB | .31 | 3,528 | 101,000 | 1,130 | 11.2 |
| 20 vars | UU | .05 | 4,648 | 251,000 | 20,400 | 81.0 |
| 294 clauses | UB | .05 | 4,488 | 513,000 | 519,000 | 1,010 |
| Domain size 16 | NB | .24 | 7,248 | 55,400 | 2,250 | 40.6 |
| 15 vars | UU | .018 | 10,848 | 145,000 | 86,0000 | 592 |
| 302 clauses | UB | | 9,168 | | | |
| Domain size 32 | NB | .18 | 14,736 | 27,900 | 4,770 | 171 |
| 12 vars | UU | .01 | 26,640 | 77,000 | 515,000 | 6,690 |
| 307 clauses | UB | | 18,576 | | | |

*Figure 3.* Results, to no more than three significant figures, for a suite of 101 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.

we used NB-Satz[6] (Stock, 2000) to experimentally locate the point of the phase transition The leftmost column of Figure 3 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 kept the first 101 satisfiable ones as identified by NB-Satz. At each domain size, 1001 attempts were made to solve the suite of 101 formulas with the NB, UU and UB methods. By Corollary 1 a positive CNF nb-formula can be transformed to a b-formula without the ALO formulas. Since our randomly-generated formulas are positive, the unary/unary and unary/binary encodings that are used here contain only kernels and AMO formulas.

[6] Satz (Li and Anbulagan, 1997) is an implementation of the DPLL algorithm (Davis et al., 1962) for deciding Boolean satisfiability. NB-Satz generalises Satz to handle non-Boolean formulas. Satz and other Boolean decision procedures are incapable of solving these random instances with large domain sizes.

The results of these experiments,[7] as shown in Figure 3, consistently follow a clear pattern. At all domain sizes, in terms of both time and number of flips, NB outperforms UU, which in turn outperforms UB. All the methods show a decline in performance as domain size grows. The decline is so sharp for the transformation methods 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 flip rates of all methods decline with increasing domain size.

Results on the EB method are reported only for domain size 2 since at larger domain sizes the enhanced binary transformation produces unreasonably large formulas. As discussed at the end of Section 4.2, if the enhanced 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 $\frac{3}{2}D$. With a domain size of 4 the transformation produces a b-formula with 319,488 atoms (which Walksat cannot solve even with hours of CPU time) and with a domain size of 8, the b-formula has over $5 \cdot 10^9$ atoms. Finally, recall (from Section 4.2) that applying the binary transform to an nb-formula of domain size 2 results in an essentially identical Boolean formula. Hence at domain size 2, the only difference between the NB and EB methods are that NB uses NB-Walksat and EB uses Walksat. The results in the table show that for these random formulas NB-Walksat's flip rate is 61% of Walksat's flip rate; this slow down is the overhead incurred by the generality of NB-Walksat. It should be noted that the amount of overhead could be quite different on problems with other characteristics.

### 6.3. Round Robin Tournament Scheduling

The round robin tournament scheduling problem appears as problem 26 in CSPLib (Gent and Walsh, ), where it is specified as

> The problem is to schedule a tournament of $n$ teams over $n-1$ weeks, with each week divided into $n/2$ periods, and each period divided into two slots. The first team in each slot plays at home,

[7] The experiments reported here improve upon similar ones reported by Frisch and Peugniez (2001). Here we correct a faulty $P_{noise}$ setting (that for UU at domain size 4), use larger test suites (101 instances each, instead of 25), and use more sample more runs (1001 instead of 101). The median flips to solution is generally higher in these experiments than in the previous ones. We believe that this is a consequence of using larger test suites. Since the $P_{noise}$ parameter is set to optimise performance over the entire suite, it more closely fits the optimal settings of the instances in a small suite than those in a large suite. To see this, just consider a one-instance suite and a two-instance suite.

whilst the second plays the first team away. A tournament must satisfy the following three constraints: (1) every team plays once a week; (2) every team plays at most twice in the same period over the tournament; (3) every team plays every other team.

An example schedule for 8 teams is:

| | week 1 | week 2 | week 3 | week 4 | week 5 | week 6 | week 7 |
|---|---|---|---|---|---|---|---|
| period 1 | 8 vs 1 | 8 vs 2 | 4 vs 7 | 3 vs 6 | 3 vs 7 | 1 vs 5 | 2 vs 4 |
| period 2 | 2 vs 3 | 1 vs 7 | 8 vs 3 | 5 vs 7 | 1 vs 4 | 8 vs 6 | 5 vs 6 |
| period 3 | 4 vs 5 | 3 vs 5 | 1 vs 6 | 8 vs 4 | 2 vs 6 | 2 vs 7 | 8 vs 7 |
| period 4 | 6 vs 7 | 4 vs 6 | 2 vs 5 | 1 vs 2 | 8 vs 5 | 3 vs 4 | 1 vs 3 |

Before proceeding it is worth observing a constraint that is implied by the given ones. Since a team plays exactly $n-1$ times and at most twice in each of the $n/2$ periods it follows that a given team plays twice in all but one period and in the remaining period it plays once.

We use two general approaches to encoding this problem in NB-SAT. In the *singleton approach* the domains of the variables are the teams, so each match is represented by a pair of variables. In the *pairwise approach* the domains of the variables are unordered pairs of teams, so each match is represented by a single variable.

To aid conceptualisation, we define three macros for producing sets of NB-formulas. Let *Vars* be a set of nb-variables and *Atoms* be a set of $m$ nb-atoms. Then ALLDIFF(*Vars*) is true if and only if every variable in *Vars* is assigned a different value; AM2(*Atoms*) is true if and only if at most two of the atoms in *Atoms* is true; and AL2(*Atoms*) is true if and only if at least two of the atoms in *Atoms* is true. The AL2 macro works by asserting that for every $m-1$ atoms chosen from *Atoms* at least one of the $m-1$ is true.

$$\text{ALLDIFF}(\mathit{Vars}) \stackrel{def}{=} \{\neg v_1/d \vee \neg v_2/d \mid \{v_1, v_2\} \subseteq \mathit{Vars}, d \in \mathit{dom}(v_1) \cap \mathit{dom}(v_2)\}$$ [8]

$$\text{AM2}(\mathit{Atoms}) \stackrel{def}{=} \{\neg a_1 \vee \neg a_2 \vee \neg a_3 \mid \{a_1, a_2, a_3\} \subseteq \mathit{Atoms}\}$$

$$\text{AL2}(\mathit{Atoms}) \stackrel{def}{=} \{a_1 \vee a_2 \vee \cdots \vee a_{m-1} \mid \{a_1, \ldots, a_{m-1}\} \subseteq \mathit{Atoms}\}$$

ALLDIFF(*Vars*) comprises $\sum_{\{v_1,v_2\} \subseteq \mathit{Vars}} |\mathit{dom}(v_1) \cap \mathit{dom}(v_2)|$ clauses, each containing two literals. AM2(*Atoms*) comprises $m(m-1)(m-2)/6$ clauses, each containing three literals, and AL2(*Atoms*) comprises $m$ clauses, each containing $m-1$ literals.

[8] The notation $\{v_1, v_2\} \subseteq \mathit{Vars}$ means that a two-element subset is selected from *Vars* and, without loss of generality, the two elements are arbitrarily named $v_1$ and $v_2$

6.3.1. *Singleton Approach*

For each period $p$ and each week $w$ we use a pair of variables, $\langle X_{p,w}, Y_{p,w}\rangle$ to represent the pair of teams that play against each other in period $p$ during week $w$. We consider the $n$ teams to be denoted by the integers 1 to $n$, hence the domain of every variable is $\{1,\ldots,n\}$. By symmetry and the fact that a team cannot play itself, we can limit our search to solutions in which $X_{p,w}$ takes a value that is strictly less than that of $Y_{p,w}$. Hence, the domain of each $X_{p,w}$ is $\{1,\ldots,n-1\}$ and the domain of each $Y_{p,w}$ is $\{2,\ldots,n\}$. Furthermore, our encoding of the problem includes a symmetry-breaking constraint, which asserts that the value of $X_{p,w}$ is strictly less than that of $Y_{p,w}$. As we shall see, the inclusion of the symmetry-breaking constraints allows one of the other constraints to be stated much more compactly with the overall effect of producing a more compact encoding of the entire problem.

Hence to represent the 4 team problem we use 12 variables as follows:

| | week 1 | week 2 | week 3 |
|---|---|---|---|
| period 1 | $X_{1,1}$ vs $Y_{1,1}$ | $X_{1,2}$ vs $Y_{1,2}$ | $X_{1,3}$ vs $Y_{1,3}$ |
| period 2 | $X_{2,1}$ vs $Y_{2,1}$ | $X_{2,2}$ vs $Y_{2,2}$ | $X_{2,3}$ vs $Y_{2,3}$ |

where each $X_{p,w}$ has domain $\{1,2,3\}$ and each $Y_{p,w}$ has domain $\{2,3,4\}$.

There are many ways to encode in NB-SAT the three constraints of the original problem statement and the symmetry-breaking constraint. Here we present one way of handling each of the problem constraints and two ways of handling the symmetry-breaking constraint.

*(1) Every team plays once a week.* Since there are $n$ slots each week and there are $n$ teams, it suffices to stipulate that each team plays at most once in each week. For each week $w$, we use the formula

$$(s1)\quad \textsc{alldiff}(\{X_{p,w} \mid p \in \mathit{periods}\} \cup \{Y_{p,w} \mid p \in \mathit{periods}\})$$

Each instance of ALLDIFF generates $\Omega(n^3)$ clauses of size two. Overall, $\Omega(n^4)$ clauses are generated.

*(2) Every team plays at most twice in the same period over the tournament.* For any team $t$, for every period $p$ we have the clauses

$$(s2)\quad \textsc{am}2(\{X_{p,w}/t \mid w \in \mathit{weeks}, t \neq n\} \cup \{Y_{p,w}/t \mid w \in \mathit{weeks}, t \neq 1\})$$

The restrictions $t \neq n$ and $t \neq 1$ prevent the set comprehensions from generating the atoms $X_{p,w}/n$ and $Y_{p,w}/1$, respectively, both of which are ill-formed because the specified value is not in the domain of the specified variable. Each instance of AM2 generates $\Omega(n^3)$ clauses of size three. Overall, $\Omega(n^5)$ clauses are generated.

*(3) Every team plays every other team.* This constraint can be obtained by stipulating that no two teams play each other twice. For any two distinct weeks,$w$ and $w'$, and any teams $t$ and $t'$ such that $t < t'$ and any periods $p$ and $p'$ we have the formula

$$(s3) \quad \neg X_{p,w}/t \vee \neg X_{p',w'}/t \vee \neg Y_{p,w}/t' \vee \neg Y_{p',w'}/t'$$

We only need to consider teams $t$ and $t'$ such that $t < t'$ because the symmetry-breaking constraint ensures that in every solution $X_{p,w}$ has a strictly smaller value than does $Y_{p,w}$. Furthermore, because of the symmetry-breaking constraint, we need enforce only $\langle X_{p,w}, Y_{p,w}\rangle \neq \langle X_{p',w'}, Y_{p',w'}\rangle$, and not $\langle X_{p,w}, Y_{p,w}\rangle \neq \langle Y_{p',w'}, X_{p',w'}\rangle$. Each of these considerations halves the number of clauses needed to impose this constraint. Attention can be restricted to distinct weeks since no team plays more than once in the same week. This encoding produces $\Omega(n^6)$ clauses, each with four literals. As we will see, the symmetry-breaking constraint is encoded far more compactly than the present constraint, so using the symmetry-breaking constraint enables a smaller encoding of the entire problem

*Symmetry breaking.* The symmetry-breaking constraint requires that for each period $p$ and each week $w$ the value of $X_{p,w}$ is strictly less than that of $Y_{p,w}$. Notice that this also enforces the constraint that no team ever plays itself. We consider two ways to encode this constraint.

The first encoding, called *lopsided*, states that if $X_{p,w}$ has value $t$ then $Y_{p,w}$ has a value strictly greater than $t$. The value of $X_{p,w}$ implies a restriction on the value of $Y_{p,w}$. For every team $t$ such that $2 \leq t \leq n-1$ and every period $p$ and every week $w$ we have the clause

$$(lopsided) \quad \neg X_{p,w}/t \ \vee \ Y_{p,w}/t{+}1 \ \vee \ Y_{p,w}/t{+}2 \ \vee \ \cdots \ \vee \ Y_{p,w}/n.$$

This encoding generates $\Omega(n^3)$ clauses ranging in size from 2 to $n-1$ literals.

The second encoding, called *negated*, states that $X_{p,w}$ and $Y_{p,w}$ cannot take on any pair of values that results in $Y_{p,w}$ being less than or equal to $X_{p,w}$. For every two teams $t$ and $t'$ such that $2 \leq t \leq t' \leq n-1$ and every period $p$ and week $w$ we have the clause

$$(negated) \quad \neg X_{p,w}/t' \ \vee \ \neg Y_{p,w}/t$$

This encoding produces $\Omega(n^4)$ clauses, each containing two literals.

Béjar and Manyá have experimented with two singleton encodings. In one paper (Béjar and Manyà, 1999b) they use the constraints s1, s2, s3 and lopsided. In another paper (Béjar and Manyà, 2000) they use

| Instance | method | $P_{noise}$ setting | clauses | formula size |
|---|---|---|---|---|
| 6 teams | NB | 0.310 | 2,790 | 9,300 |
| DS=5 | UU | 0.250 | 2,820 | 9,450 |
| 8 teams | NB | 0.130 | 18,088 | 61,992 |
| DS=7 | UU | 0.090 | 18,144 | 62,384 |
| 10 teams | NB | 0.051 | 72,900 | 254,520 |
| DS=9 | UU | 0.033 | 72,990 | 255,330 |

| Instance | method | flip rate (flips/s) | median flips | median time (s) | mean flips | S.D. flips |
|---|---|---|---|---|---|---|
| 6 teams | NB | 49,528 | 333 | 0.0067 | 476 | 461 |
| DS=5 | UU | 246,867 | 1,485 | 0.0060 | 2,019 | 1,853 |
| 8 teams | NB | 20,658 | 1,718 | 0.0832 | 2,408 | 2,221 |
| DS=7 | UU | 126,675 | 7,386 | 0.0583 | 10,778 | 9,423 |
| 10 teams | NB | 12,730 | 15,786 | 1.2401 | 23,990 | 24,577 |
| DS=9 | UU | 73,369 | 73,079 | 0.9961 | 100,838 | 99,678 |

*Figure 4.* Round Robin formulated with s1, s2, s3, negated and ALO. The domain size (DS) is one less than the number of teams. Sample size is 1000.

a UU encoding of the constraints s1, s2, s3 and negated. In this UU encoding they added ALO clauses but no AMO clauses.

We experimented with two nb-encodings of the problem. The first encoding uses s1, s2, s3 and negated. The unary/unary transformation was applied to produce a Boolean encoding. As the original nb-formula is negative, ALO clauses are included in the UU encoding, but AMO clauses are omitted. Figure 4 shows the results of 1000 runs each on the UU and NB encodings for 6, 8 and 10 teams.

The second encoding uses s1, s2, s3 and lopsided. The unary/unary transformation was applied to produce a Boolean encoding. As the original nb-formula is neither negative nor positive, both ALO and AMO clauses are included in the UU encoding. Figure 5 shows the results of 1000 runs each on the UU and NB encodings for 6, 8 and 10 teams.

The two tables exhibit similar patterns. NB requires fewer flips to solve the problem instances, and its advantage over UU grows as the domain size increases, markedly so for the lopsided encodings. Both figures show that UU has a significantly higher flip rate; consequently UU has a slightly faster solution time in the negated encodings. However, since UU scales so poorly with the lopsided encodings, NB has a time advantage here and its advantage grows as the domain size grows. No results are given for UB or EB as these are ineffective on these instances.

| Instance | method | $P_{noise}$ setting | clauses | formula size |
|---|---|---|---|---|
| 6 teams | NB | 0.270 | 2,700 | 9,165 |
| DS=5 | UU | 0.180 | 3,030 | 9,915 |
| 8 teams | NB | 0.100 | 17,668 | 61,432 |
| DS=7 | UU | 0.090 | 18,900 | 64,176 |
| 10 teams | NB | 0.052 | 71,640 | 252,945 |
| DS=9 | UU | 0.028 | 72,970 | 260,235 |

| Instance | method | flip rate (flips/s) | median flips | median time (s) | mean flips | S.D. flips |
|---|---|---|---|---|---|---|
| 6 teams | NB | 50,258 | 304 | 0.0060 | 467 | 475 |
| DS=5 | UU | 227,018 | 1,683 | 0.0074 | 2,372 | 2,249 |
| 8 teams | NB | 22,762 | 1,957 | 0.0860 | 2,714 | 2,604 |
| DS=7 | UU | 120,935 | 12,997 | 0.1075 | 19,540 | 19,380 |
| 10 teams | NB | 12,899 | 16,672 | 1.2925 | 22,260 | 20,857 |
| DS=9 | UU | 69,153 | 313,134 | 4.5281 | 445,906 | 449,436 |

*Figure 5.* Round Robin formulated with s1, s2, s3, lopsided, ALO and AMO. The domain size (DS) is one less than the number of teams. Sample size is 1000.

6.3.2. *Pairwise Approach*

The pairwise approach uses a single variable $XY_{p,w}$ for each period $p$ and week $w$. The domain of each variable is the set of all unordered pairs of distinct teams. We shall refer to this set of values as *matches* and note that it has $n(n-1)/2$ elements. We shall also write $matches(t)$ to denote those pairs in *matches* that contain team $t$ and note that for all $t$ this set contains $n-1$ elements. Hence to represent the 4 team problem we use 6 variables as follows:

| | week 1 | week 2 | week 3 |
|---|---|---|---|
| period 1 | $XY_{1,1}$ | $XY_{1,2}$ | $XY_{1,3}$ |
| period 2 | $XY_{2,1}$ | $XY_{2,2}$ | $XY_{2,3}$ |

where each $XY_{p,w}$ has the domain:

$$matches = \{\{1,2\},\{1,3\},\{1,4\},\{2,3\}\{2,4\},\{3,4\}\}$$

Also, to illustrate the notation, $matches(2) = \{\{1,2\},\{2,3\}\{2,4\}\}$.

We now present a single encoding of each of the three problem constraints.

*(1) Every team plays once a week.* Unlike the singleton approach, we impose this constraint by saying that each team plays at least once

each week. For every week $w$, for each team $t$ we have the clause:

$$(p1) \qquad \bigvee_{p \in periods,\ m \in matches(t)} XY_{p,w}/m$$

This encoding produces $\Omega(n^2)$ clauses, each of length $\Omega(n^2)$.

*(2) Every team plays at most twice in the same period over the tournament.* To impose this constraint we add an "imaginary" extra week and write $weeks^+$ to denote the expanded set of weeks. It follows that over the course of $n$ $weeks^+$ a team plays at most twice in the same period if and only if it plays at least twice in the same period. For each period $p$ for each team $t$ we have the following formula:

$$(p2) \quad \text{AL2}(\{XY_{p,w}/m \mid m \in matches(t), w \in weeks^+\})$$

Each instance of AL2 produces $\Omega(n^2)$ clauses each with $\Omega(n^2)$ literals. Overall, this produces $\Omega(n^4)$ clauses.

*(3) Every team plays every other team.* In the pairwise approach it is straightforward to stipulate that every pair of teams plays at least once. This can be done with one clause for each $m$ in $matches$:

$$(p3) \qquad \bigvee_{p \in periods,\ w \in weeks} XY_{p,w}/m$$

This yields $\Omega(n)$ clauses, each containing $\Omega(n^2)$ literals.

We experimented with the NB encoding of p1, p2 and p3, and the UU transform of this. As the NB encoding is neither positive nor negative, the UU encoding contains both ALO and AMO clauses. Figure 6 shows the results of 1000 runs each on the NB encodings for 6, 8, 10 and 12 teams, and the UU encoding for 6 teams. Beyond 6 teams, the UU encoding is ineffective and the UB and EB encodings are ineffective even for 6 teams. Indeed, the Boolean encodings rapidly become prohibitively large as the number of teams increase. However, the NB method is highly effective, and easily the best of all the solution methods considered in this section. Here we see that with large domain sizes, NB can be the only effective solution method.

### 6.4. The Quasi-Group Completion Problem

A quasi-group (also called a Latin Square[9]) of order $n$ is an $n \times n$ table of symbols There are $n$ symbols and each symbol occurs exactly once in

[9] A quasi-group is a group whose multiplication table is a latin square. However, in the AI community the table is commonly referred to as the quasi-group.

| Instance | method | $P_{noise}$ setting | clauses | formula size |
|---|---|---|---|---|
| 6 teams | NB | 0.120 | 159 | 3,465 |
| DS=15 | UU | 0.010 | 2,049 | 7,245 |
| 8 teams | NB | 0.020 | 348 | 15,120 |
| DS=28 | | | | |
| 10 teams | NB | 0.012 | 645 | 47,025 |
| DS=45 | | | | |
| 12 teams | NB | 0.008 | 1,074 | 118,404 |
| DS=66 | | | | |
| 14 teams | NB | 0.002 | 1,659 | 257,985 |
| DS=91 | | | | |

| Instance | method | flip rate (flips/s) | median flips | median time (s) | mean flips | S.D. flips |
|---|---|---|---|---|---|---|
| 6 teams | NB | 86,067 | 294 | 0.0034 | 404 | 364 |
| DS=15 | UU | 240,768 | 7,787 | 0.0323 | 11,523 | 11,198 |
| 8 teams | NB | 45,136 | 973 | 0.0216 | 1,373 | 1,260 |
| DS=28 | | | | | | |
| 10 teams | NB | 24,333 | 6,026 | 0.2476 | 8,619 | 7,846 |
| DS=45 | | | | | | |
| 12 teams | NB | 13,809 | 28,290 | 2.0487 | 41,010 | 40,375 |
| DS=66 | | | | | | |
| 14 teams | NB | 8,308 | 21,8574 | 26.3087 | 318,896 | 319,472 |
| DS=91 | | | | | | |

*Figure 6.* Round Robin formulated with p1, p2 and p3. The domain size (DS) is $n(n-1)/2$, where $n$ is the number of teams. The sample size is 1000.

each row and column of the table. Therefore each row and each column is a permutation of the symbols.

Generating a quasi-group of order $n$ is trivial and can be done in $O(n^2)$ time. However, completing a quasi-group which is partially filled is an NP-complete problem (Colbourn, 1983). This is known as the quasi-group completion problem (QCP). For a fixed $n$, as the number of unfilled slots (or holes, $h$) increases, QCP exhibits both a phase transition from unsatisfiable to satisfiable and an easy-hard-easy pattern of solvability (Gomes and Selman, 1997).

To evaluate local search procedures with QCP, the instances must be filtered with a systematic search procedure so that only satisfiable ones remain. Achlioptas et al. (2000) found generating QCP instances and filtering with a complete search procedure too compute-intensive. They suggest generating complete quasi-groups then emptying some of the slots to create $h$ holes. Problems generated this way are clearly guaranteed to be satisfiable, so they do not have traditional phase-transition

behaviour. However, they do retain the easy-hard-easy pattern with increasing $h$. This method of generating QCP instances is referred to as quasi-group with holes (QWH). QWH does include all satisfiable QCP instances, however Kautz et al. (2001) observe that it is biased away from the uniform distribution of satisfiable instances because QWH can generate the same QCP instance from a variable number of complete quasi-groups. Empirically, the most difficult QWH instances are found where $h = 1.6n^{1.55}$ (Achlioptas et al., 2000). This result follows from observation of the performance of Walksat on the unary/unary transform with ALO clauses.

6.4.1. *Generating QWH Instances*
For each order size $n \in \{4, 8, 12, 16, 20\}$, we generated a suite of 25 QWH instances. We used the lsencode v1.0 software (Gomes et al., 2001) to systematically generate a complete quasi-group instance and then shuffle it 10000 times. This gives a uniform distribution over all complete quasi-groups of order $n$. Using lsencode, $1.6n^{1.55}$ holes were then poked in the quasi-group by the random method.

6.4.2. *The Non-Boolean Encoding*
We used our own program, PLS-NB, to encode each of the generated QWH instances as an NB-formula. We initially encoded each instance with $n^2$ variables, one for each entry in the quasi-group table. For each row and column, the variables must be assigned a permutation of the values. For a row or column containing *variables*, it is sufficient to assert that each value appears at least once in *variables*, since this entails that each value appears exactly once. For each row or column containing *variables*, and each symbol $s$ we have the clause

$$\bigvee_{A \in variables} A/s \tag{4}$$

For each variable $A$ that corresponds to a filled entry in the table (i.e., not a hole), we add the unit clause $A/s$, where $s$ is the symbol that fills the entry. This gives us a CNF NB-formula that encodes the QWH instance.

This NB representation is then simplified by performing unit propagation. To propagate the unit clause $A/s$, all other clauses containing the variable $A$ are processed with one of two rules:

– *Unit subsumption* applies to a clause $C$ containing $A/s$. The unit clause logically subsumes $C$, so $C$ is removed from the formula.

– *Unit resolution* applies to a clause $C$ containing $A/t$ where $t \neq s$. The literal $A/t$ is removed from $C$ because $(A/t \vee \phi) \wedge (A/s)$ entails $\phi$.

Unit propagation is performed to closure, after which all unit clauses are deleted from the NB formula.

The variables remaining in the formula correspond to the holes in the QWH instance. The domains of these variables typically contain less than $n$ values, and can vary from variable to variable. For each suite Figure 7 shows the mean domain size (MDS) of the NB encodings for each suite of QWH instances.

It is possible to simplify QWH instances much more than the unit propagation that we have done. For example, before encoding an instance into NB-SAT, one could consider the instance as a clique of disequalities for each row and column and perform arc-consistency. Or even stronger, one could consider an instance as an alldifferent constraint for each row and column and perform generalised arc-consistency (Kautz et al., 2001; Shaw et al., 1998). Unfortunately, both of these simplifications reduce the domains of the variables so effectively, that all reasonably-sized problems have small domains. For example, consider simplifying the instances with arc-consistency (as explained above) and then with unit propagation (as we have done). Then our suite of 25 instances for order 4 has a mean domain size of 2.84; for order 16 it is 3.85; and for order 20 it is 4.09. As such it is impossible to investigate the effect of increasing domain size, because the problem instances become too hard for large $n$. Since we are interested in producing a useful testbed, not with solving the problem as fast as possible, we pre-process by unit propagation only.

6.4.3. *The Boolean Encodings*

In addition to the NB-encoding, three Boolean transforms of the non-Boolean formula are considered: UU, UB and EB. As the NB encodings are all positive, no ALO clauses are included in the UU or UB encodings.

Kautz et al. (2001) use two other Boolean encodings, which we (but not they) now describe as Boolean transforms of non-Boolean encodings. In both encodings they preprocess the instance by performing generalised arc-consistency as described above. In the encoding they call "2D" they do not use (4), but instead impose ALLDIFF(*variables*) on each row and column. Finally they apply the unary/unary transform, including ALO clauses, but not AMO clauses. In their "3D" encoding they impose both (4) and ALLDIFF(*variables*) on all rows and columns. Finally, they apply the unary/unary transform, including both ALO and AMO clauses.

6.4.4. *Results*

Once again, we treat the process of solving an entire suite as a single sample execution. For each solution method, for each test suite, the optimal $P_{noise}$ setting was experimentally determined. These are shown in Figure 7. However, using a cutoff of 400 seconds (for the suite), EB could not solve the suites with $n \geq 8$ and UB could not solve those with $n \geq 12$.

In particular, EB made nine attempts to solve the order 8 suite at each of the $P_{noise}$ settings 0.4, 0.45, 0.5, 0.55 and 0.6. None of the attempts completed the suite in 400 seconds; the most successful was $P_{noise} = 0.45$, which completed the first 12 instances on one of its runs. Thus, a $P_{noise}$ setting of .45 is recorded in Figure 7, with an asterisk to indicate that this is a special situation.

UB made nine attempts to solve the order 12 suite at each of the $P_{noise}$ settings 0.05, 0.1, 0.15, 0.2 and 0.25. The most successful was $P_{noise} = 0.1$, though not all of its runs completed within the cutoff of 400 seconds. Thus a $P_{noise}$ setting of .10 is recorded in Figure 7, with an asterisk to indicate that this is a special situation.

For all the other methods and suites, 101 runs were performed and all completed within the 400 second limit. The results are shown in Figure 7, where the the flips and time measurements are reported per instance (i.e., the measurements for the suite are divided by 25). The order 4 suite was solved so fast that a reliable measure of CPU time could not be obtained; thus no flip rates or times are given.

The results reported in Figure 7 show that the performance of EB and UB scale poorly as domain size increases. EB is competitive for order 4 instances (MDS=3.93), but not for order 8 (MDS=7.01). UB is effective, though substantially inferior to NB and UU for instances of order 4 and 8; it is ineffective for order 12 instances (MDS=9.46).

The results also show that both UU and NB remain effective for instances of order 20 (MDS=13.52), though NB is superior to UU, and its superiority grows as domain size increases.

## 7. Related Work

This section considers related work, first work on solving problems without encoding, then work on the direct approach and finally work on the transformational approach.

| Problem | method | $P_{noise}$ setting | mean variables | mean clauses | mean formula size |
|---|---|---|---|---|---|
| Order 4 | NB | .22 | 13.0 | 26.0 | 88.3 |
| 13 holes | UU | .16 | 51.1 | 101 | 238 |
| 3.93 MDS | UB | .41 | 77.1 | 127 | 290 |
| | EB | .50 | 26.0 | 293 | 1,074 |
| Order 8 | NB | .20 | 40.0 | 79.9 | 424 |
| 40 holes | UU | .029 | 279 | 936 | 2,136 |
| 7.01 MDS | UB | .16 | 399 | 890 | 2,024 |
| | EB | .45∗ | 119 | 51,720 | 345,085 |
| Order 12 | NB | 15 | 75.0 | 150 | 998 |
| 75 holes | UU | .035 | 709 | 3,215 | 7,129 |
| 9.46 MDS | UB | .10∗ | 992 | 2,504 | 5,707 |
| Order 16 | NB | .12 | 117 | 234 | 1826 |
| 117 holes | UU | .020 | 1,354 | 7,558 | 1,6473 |
| 11.57 MDS | | | | | |
| Order 20 | NB | .108 | 166 | 332 | 2,953 |
| 166 holes | UU | .015 | 2,244 | 14,711 | 31,711 |
| 13.52 MDS | | | | | |

| Problem | method | mean flip rate (flips/ms) | median flips | mean flips | median time (ms) | mean time (ms) |
|---|---|---|---|---|---|---|
| Order 4 | NB | | 27.2 | 27.5 | | |
| | UU | | 154 | 161 | | |
| | UB | | 568 | 590 | | |
| | EB | | 56.0 | 57.9 | | |
| Order 8 | NB | 1,228 | 702 | 721 | 0.62 | 0.64 |
| | UU | 1,375 | 4,982 | 5,383 | 3.63 | 3.91 |
| | UB | 1,472 | 57,279 | 60,377 | 38.8 | 41.0 |
| | EB | 7.26 | | | $> 16,000$ | $> 16,000$ |
| Order 12 | NB | 954 | 7,819 | 8,122 | 7.97 | 8.33 |
| | UU | 1,116 | 56,458 | 58,966 | 50.5 | 52.7 |
| | UB | 1,422 | | | $> 16,000$ | $> 16,000$ |
| Order 16 | NB | 780 | 59,865 | 61,673 | 56.0 | 58.0 |
| | UU | 1,002 | 354,759 | 374,878 | 351 | 369 |
| Order 20 | NB | 736 | 167,738 | 183,396 | 228 | 249 |
| | UU | 779 | 1,412,583 | 1,422,320 | 1,810 | 1,830 |

*Figure 7.* Performance on quasigroup with holes.

### 7.1. Solving Problems without Encoding

A substantial body of work takes a purely direct approach by applying local search directly to a problem, for example, to vehicle routing (Shaw, 1998)). This typically involves the overhead of building a domain-specific solver from scratch, but allows the development of problem-specific neighbourhoods and heuristics. Brafman and Hoos (1999) show that for planning problems this approach can result in a solver that is more efficient than encoding and solving with SAT. Recent work on the development of the COMET system (Van Hentenryck and Michel, 2003) shows that generic facilities, such as incremental data structures, can be provided to ease the development of domain-specific local search systems.

The advantage of encoding a problem into SAT or NB-SAT is that developing an effective encoding is likely to be less difficult than implementing a problem-specific local search procedure. This is demonstrated by the continuing increase in the number of problems that can be solved effectively by applying both systematic and SLS solvers to encodings of the problems.

### 7.2. Other Work on the Direct Approach

As far as we know, only one other group has worked on a direct approach to solving non-Boolean satisfiability problems with stochastic local search. Bejar and his colleagues (Béjar, 2000; Béjar and Manyà, 1999a; Béjar and Manyà, 2000; Béjar et al., 2001) have generalised GSAT, Walksat, and some of their variants to operate directly on regular formulas of finitely-valued logic. In regular formulas all variables have the same finite, totally-ordered domain, which for the sake of presentation is usually taken to be $\{1, \ldots, n\}$.[10] A regular literal is of the form $\uparrow i{:}X$ or $\downarrow i{:}X$ where $i$ is a domain element and $X$ is a variable. Assignments in regular SAT are the same as those in NB-SAT. An assignment satisfies $\uparrow i{:}X$ if the value assigned to $X$ is at least $i$; it satisfies $\downarrow i{:}X$ if the value assigned to $X$ is at most $i$. A regular CNF formula is a conjunction of disjunctions of regular literals. Negation is not used in regular CNF; nor is it needed as the negation of $\uparrow i{:}X$ (resp. $\downarrow i{:}X$) is logically equal to $\downarrow(i-1){:}X$ (resp. $\uparrow(i+1){:}X$).

The NB-Walksat and Regular-Walksat algorithms can be seen as having only two differences. The first is that instead of step (g) (see Figure 1), Regular Walksat uses

[10] Here we stick with the terminology of this paper. Their presentation uses the terminology of multi-valued logic.

(g′) From among the variables that appear in L select one at random.
Call it X.
Let D be the set of values such that flipping X to that value would satisfy the clause.
Randomly select a member of $D$ and call it $d$.

The second difference is that the published descriptions of Regular-Walksat do not specify distributions for the random selections made in steps (d) or (g′). Assuming that all selections are made with uniform distribution, in certain situations (g) and (g′) can choose among flips with different distributions. To see this, consider two variables, $X$ and $Y$, each with domains $\{1, 2, 3\}$. Then the nb-clause $X/2 \vee X/3 \vee Y/3$ is logically equivalent to the regular clause $\uparrow 2{:}X \vee \uparrow 3{:}Y$. If these clauses are false, then each could be "repaired" by one of three flips: $X$ to 2, $X$ to 3 or $Y$ to 3. In a noisy move, NB-Walksat would chose each flip with probability 1/3, whereas Regular-Walksat would choose between $X$ and $Y$ with probability 1/2 and, if $X$ were chosen it would chose between 2 and 3 with probability 1/2. Hence, Regular Walksat would choose among the three flips with probability 1/4, 1/4 and 1/2 respectively. This situation could arise only if a problem has a clause in which different variables participate in a different number of repairing flips. Of all the formulations considered in Section 6, this situation arises only in the lopsided and pairwise formulations of the round-robin problem.

If we ignore the issue of distributions, NB-Walksat and Regular-Walksat are functionally equivalent in the following sense. Let $C_{reg} = C_1, \ldots, C_n$ be a set of regular clauses and $C_{NB} = C'_1, \ldots, C'_n$ be a set of NB-clauses. If $C_i$ and $C'_i$ are logically equivalent for $1 \leq i \leq n$, then the set of runs available to Regular-Walksat running on $C_{reg}$ is the same as those available to NB-Walksat running on $C_{NB}$.

How do the NB and regular languages compare in terms of their ability to represent problems and solve them with NB-Walksat and Regular-Walksat? Considering the equivalence between the algorithms, this issue hinges on the relative expressiveness of NB-clauses and regular clauses.

Regular clauses can be transformed easily to NB-clauses by replacing each occurrence of $\uparrow i{:}X$ (resp. $\downarrow i{:}X$) with $\bigvee_{j>i} X/j$ (resp. $\bigvee_{j<i} X/j$). Going the other way, negative NB-clauses can be transformed easily to regular clauses by replacing each occurrence of $\neg X/i$ with $\uparrow(i+1){:}X \vee \downarrow(i-1){:}X$. However, non-negative NB-clauses are not, in general, equivalent to any regular clause. The unit clause $X/i$ is equivalent to $\uparrow i{:}X \wedge \downarrow i{:}X$, but it is not equivalent to any regular clause. Consequently, NB-Walsksat is capable of all the behaviours that Regular-Walksat is, but not vice-versa. To be clear, the issue is not that

Regular-Walksat lacks capabilities, rather it is that regular-CNF is not sufficiently expressive.

Of course, every non-negative NB-formula $\phi_{NB}$ in CNF is logically equivalent to some regular formula $\phi_{reg}$ (by the correspondences mentioned in the last paragraph) and $\phi_{reg}$ can be transformed to a CNF formula $\phi'_{reg}$ by distributing disjunctions over conjunctions. $\phi_{NB}$ and $\phi'_{reg}$ will be logically equivalent, though generally they will not correspond clause by clause. Indeed, $\phi_{reg}$ may be unreasonably large. For this reason, Regular-Walksat is typically ineffective on encodings that correspond to negative NB-formulas in CNF. The two problems used in the Regular-Walksat experiments of Béjar and Manyà (2000) use negative encodings: graph colouring using the encoding of Section 6.1 and round robin tournament scheduling using clauses s1, s2, s3 and lopsided of Section 6.3.1. The most effective round-robin encoding, the pairwise one, uses both negative and positive clauses and is therefore ill-suited for Regular-Walksat. Again, the shortcoming is not with the Regular Walksat algorithm, it is that this non-negative encoding cannot be expressed in regular CNF.

An obvious difference between NB-SAT and regular SAT is that regular SAT uses a totally-ordered domain. One consequence is that in representing inequalities, regular CNF can be more compact. For example, the lopsided symmetry-breaking constraint of Section 6.3.1 can be represented by the regular formula

$$\uparrow(t+1){:}X_{p,w} \vee \downarrow(t-1){:}X_{p,w} \vee \uparrow(t+1){:}Y_{p,w}$$

A second consequence is that Regular-Walksat is able to exploit this compactness in its internal data structures. This, in effect, is a benefit that accrues from the limited expressiveness of regular clauses.

### 7.3. Other Work on the Transformational Approach

Let us now turn our attention to related work on using the transformation approach to solving problems with SLS on Boolean SAT. Among the numerous problems attacked with this approach are the n-queens problem (Selman et al., 1992), graph colouring (Selman et al., 1992; Hoos, 1998), the quasi-group completion problem (or quasi-group with holes) (Kautz et al., 2001; Achlioptas et al., 2000; Béjar et al., 2001), the all-interval-series problem (Hoos, 1998; Béjar et al., 2001), the Hamiltonian circuit problem (Hoos, 1998; Hoos, 1999), random instances of the finite-domain constraint satisfaction problem (Hoos, 1998; Hoos, 1999; Gent, 2002), the round-robin tournament problem (Béjar and Manyà, 2000), and planning (Kautz and Selman, 1992; Kautz and Selman, 1996; Kautz et al., 1996; Ernst et al., 1997; Hoos,

1998; Brafman and Hoos, 1999). Despite this widespread use we know only three studies, other than our own, that have compared the performance of SLS on different SAT encodings of non-Boolean variables; we discuss these below. Other than these three comparative studies, all work cited above uses the unary/unary encoding. As far as we know, our work is the first to use or mention the unary/binary encoding.

We preface our discussion of the three studies by noting that no work we know of considers the SAT-encoding process as two mappings, one from the problem to NB-SAT and then a second from NB-SAT to SAT.[11] Researchers who take the single-stage viewpoint often overlook some encodings. For example, from our viewpoint, we might see that a study considers three ways of mapping a problem to NB-SAT, and produces four SAT encodings by applying the unary/unary transform to all three but the binary transform to only one. A similar oversight sometimes occurs in considering the inclusion/exclusion of ALO and AMO clauses. To aid presentation, we shall adopt the two-stage viewpoint in the following discussion.

Hoos (1998; 1999) compares the performance of Walksat using the basic binary and unary/unary encodings of random instances of the Hamiltonian cycle problem (HCP) and of the finite-domain constraint satisfaction problem (CSP). The CSP instances are mapped to NB-SAT using the well-known "direct encoding." The HCP instances are first mapped to CSP, and then treated the same as the CSP instances. In both test sets, the NB-SAT encoding is negative; nonetheless, all the unary/unary encodings include both ALO and AMO clauses. The experiments show that the unary/unary encodings can be solved with fewer flips than the basic binary encodings; about 7 times fewer in CSP and 1.5 times fewer in the HCP.

Ernst et al. (1997) systematically study eight Boolean encodings for the STRIPS-style planning problem. Over an unidentified suite of 23 instances of the planning problem, they compare the size of each encoding and the average time that Walksat takes to solve each encoding. The eight encodings they explore are systematically generated by selecting one of two frame axioms (called classical and explanatory) and one of four action representations (called regular, simple splitting, overloaded splitting and bitwise). The regular, simple splitting and overloaded splitting encodings are three ways of representing the choice of actions and all three employ a unary/unary encoding. The bitwise action representation is not truly a different action representation; it is a binary encoding of the regular action representation. Thus, in terms of encod-

[11] The nearest exception is that Hoos (1998; 1999) maps the Hamiltonian Cycle Problem to SAT by mapping it first to the constraint satisfaction problem and then mapping this to SAT.

ing non-Boolean variables, their study makes two relevant comparisons: between the classical/regular and the classical/bitwise encodings and between the explanatory/regular and explanatory/bitwise encodings. Ernst, Millstein and Weld are keenly aware that the unary/unary encoding does not always need ALO and AMO formulas and they pay particular attention to creating and identifying opportunities for omitting them. Though their binary encoding is not specified, their claim that it has no extraneous values means that it is the enhanced version or something similar. On the whole, it appears that each unary/unary encoding generally outperforms the corresponding binary encoding, but, as Ernst, Millstein and Weld note, the "timing data is hard to interpret."

Prestwich (2004) considers seven SAT-encodings of the graph colouring problem. From our point of view these use four ways of mapping colouring to NB-SAT. Three of these are mapped to SAT using solely the unary/unary transform. The fourth, which is based on conflict clauses, is the one of interest here as it is mapped to SAT in four ways: (UULM) unary/unary with ALO and AMO, (UUL) unary/unary with ALO, (BB) basic binary, and (EB) enhanced binary. On a suite of 30 colouring instances with between 4 and 49 colours, each encoded in seven ways, he tests the performance of Walksat with the best heuristic. He observes that on no instance does EB require more flips on average than BB. And on almost all instances UUL requires fewer flips on average than UULM. Since EB and UUL encodings admit more solutions than the BB and UULM encodings, respectively, these results are consistent with the hypothesis that higher solution density (solutions divided by number of possible assignments) tends to yield better performance for SLS. A surprising result of the experiments is that the EB and BB encodings performed much better than one would have expected from the previous work of us and others. The puzzling pattern presented by the literature on the whole is that EB and BB sometimes perform quite well and other times very poorly. Our conjecture is that these two encodings do not perform well on intrinsically hard instances. Prestwich's instances are mostly much easier than the ones we used for colouring and other problems. On the hardest instances of his, EB and BB perform poorly. On three of the four easiest graph colouring instances tested by Frisch and Peugniez (2001) EB performed competitively, but it was totally ineffective on the two hardest instances.

## 8. Conclusion

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 nb-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. Nonetheless, this issue has been overlooked. All problem instances that Hoos considers have a domain size of 10. The planning instances used by Ernst, Millstein and Weld are not identified and domain sizes are not reported. The colouring instances used by Prestwich do vary considerably in domain size, but his experiments are not designed to control this parameter. In contrast to all other 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. Our study is also the first to explicitly consider the role of a formulation's polarity (being positive or negative). By considering polarity and domain size we are able to make some observations not revealed by other studies.

- Of all the methods considered here, NB scales best with increasing domain size.

- Many researchers have remarked that solving a problem by mapping it to SAT (inevitably with the unary/unary encoding) and using an SLS SAT solver can compete with using a custom-made SLS solver directly on the problem. Though our results show the UU method to be quite robust, it can run into difficulties with very large domain sizes. This is particularly true for non-negative formulations, since excessively many AMO clauses must be included.

- The UB method is rarely effective, and it is never effective on problems with moderate or large domain sizes.

- On non-negative formulations, the EB method is ineffective since the encoding becomes prohibitively large with modest domain sizes.

- On negative formulations the EB method is sometimes effective but often ineffective. It always requires more flips than NB. It sometimes requires somewhat fewer flips than UU (as shown in Prestwich's results), but sometimes requires vastly more. We know of no good explanation for this puzzling pattern with respect to

UU, but we can offer a conjecture. The EB encoding generates a difficult search space for SLS (a point argued well by Hoos), so a problem that is inherently difficult becomes practically unsolvable by EB. On the other hand, the EB search space is much smaller than than that of UU, especially with large domain sizes. We thus hypothesise that EB will be more effective than UU on problem instances that have very large domain sizes but are very easy. Note the we, and almost everyone else, have focussed on hard problem instances, often deliberately selecting ones at the phase transition. The one exception is the set of colouring instances used by Prestwich, and he has observed the most success with BB and EB.

Many questions remain to be addressed by future work. A wide range of problems and encodings have yet to be explored. Stamm-Wilbrandt (1993) shows how more than two dozen problems—most of which are NP-complete—can be transformed to SAT. The present paper shows that there is great flexibility in combining the different transforms. For example, it is possible to use a binary AMO formula with a unary ALO formula and it is possible to use different transforms on each variable. From this we see that of all the possible ways of encoding non-Boolean problems in Boolean formulas, very few have ever been tried. Even within the small sphere of well-studied problems and well-studied encodings, there are untried combinations.

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 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 and polarity as the critical parameters.

In addition to developing new problem encodings, we claim that the applicability of SLS technology can also be extended by enriching the language on which the SLS can operate. This claim is supported by recent results on pseudo-Boolean constraints (Walser, 1997) and integer optimisation (Walser, 1999), non-clausal formulas (Sebastiani, 1994); and efficient handling of variable dependencies (Kautz et al., 1997). Our success with NB-Walksat adds further weight to the claim.

**Acknowledgements**

We thank Henry Kautz and Bram Cohen for providing their Walksat code so that we could develop NB-Walksat from it; Carla Gomes for allowing us to use their quasigroup problem generator; Peter Stock for providing his NB-Satz program and for using it to filter out the unsatisfiable instances from our suite of random formulas; and Toby Walsh, Ian Gent, Steve Minton, Bart Selman and the anonymous referees for useful suggestions.

## References

Achlioptas, D., C. P. Gomes, H. A. Kautz, and B. Selman: 2000, 'Generating Satisfiable Problem Instances'. In: *Proc. of the Seventeenth National Conf. on Artificial Intelligence*. pp. 256–261.

Béjar, R.: 2000, 'Systematic and Regular Search Algorithms for Regular-SAT'. Ph.D. thesis, Universitat Autonòma de Barcelona.

Béjar, R., A. Cabicol, C. Fernàndez, F. Manyá, and C. P. Gomes: 2001, 'Capturing Structure with Satisfiability'. In: T. Walsh (ed.): *Principles and Practice of Constraint Programming — CP 2001*. pp. 135–149.

Béjar, R. and F. Manyà: 1999a, 'A Comparison of Systematic and Local Search Algorithms for Regular CNF Formulas'. In: A. Hunter and S. Parsons (eds.): *Proc. Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'99, London, UK*, Vol. 1638 of *Lecture Notes in Artificial Intelligence*. pp. 22–31.

Béjar, R. and F. Manyà: 1999b, 'Solving Combinatorial Problems with Regular Local Search Algorithms'. In: H. Ganzinger and D. McAllester (eds.): *Proc. 6th Int. Conference on Logic for Programming and Automated Reasoning, LPAR, Tbilisi, Georgia*, Vol. 1705 of *Lecture Notes in Artificial Intelligence*. pp. 33–43.

Béjar, R. and F. Manyà: 2000, 'Solving the Round Robin Problem Using Propositional Logic'. In: *Proc. of the Seventeenth National Conf. on Artificial Intelligence*. pp. 261–266.

Brafman, R. I. and H. H. Hoos: 1999, 'To Encode or not to Encode — I: Linear Planning.'. In: *Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence*. pp. 988–993.

Colbourn, C. J.: 1983, 'Embedding Partial Steiner Triple Systems is NP-Complete'. *Journal of Combinatorial Theory* **35**.

Davis, M., G. Logemann, and D. Loveland: 1962, 'A Machine Program for Theorem Proving'. *Communications of the ACM* **5**(7).

Ernst, M. D., T. D. Millstein, and D. S. Weld: 1997, 'Automatic SAT-Compilation of Planning Problems'. In: *Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence*. pp. 1169–1176.

Frisch, A. M. and T. J. Peugniez: 1998, 'Solving Non-Boolean Satisfiability Problems with Local Search'. In: *Fifth Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*. St. Andrews, Scotland.

Frisch, A. M. and T. J. Peugniez: 2001, 'Solving Non-Boolean Satisfiability Problems with Stochastic Local Search'. In: *Proc. of the Seventeenth Int. Joint Conf. on Artificial Intelligence*. Seattle, Washington, pp. 282–288.

Gent, I. P.: 2002, 'Arc Consistency in SAT'. In: *Proc. of the Fifteenth European Conf. on Artificial Intelligence*. pp. 121–125.

Gent, I. P. and T. Walsh (eds.), *CSPLib: A Problem Library for Constraints*. http://www.csplib.org.

Gomes, C. P., H. A. Kautz, and Y. Ruan: 2001, 'lsencode: A Generator of Quasigroup with Holes and Quasigroup Completion Poblems,'.

Gomes, C. P. and B. Selman: 1997, 'Problem Structure in the Presence of Perturbations'. In: *Proc. of the Fourteenth National Conf. on Artificial Intelligence*. pp. 221–226.

Hoos, H. H.: 1998, 'Stochastic Local Search—Methods, Models, Applications'. Ph.D. thesis, Technical University of Darmstadt.

Hoos, H. H.: 1999, 'SAT-Encodings, Search Space Structure, and Local Search Performance'. In: *Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence*. pp. 296–302.

Jonsson, A. K. and M. L. Ginsberg: 1993, 'Experimenting with New Systematic and Nonsystematic Search Techniques'. In: *Working Notes of the 1993 AAAI Spring Symposium on AI and NP-Hard Problems*. Stanford University in Palo Alto, California.

Kautz, H. A., D. McAllester, and B. Selman: 1996, 'Encoding Plans in Propositional Logic'. In: L. C. Aiello, J. Doyle, and S. Shapiro (eds.): *Principles of Knowledge Representation and Reasoning: Proc. of the Fifth Int. Conf.* San Francisco, pp. 374–385.

Kautz, H. A., D. McAllester, and B. Selman: 1997, 'Exploiting Variable Dependency in Local Search (Abstract)'. In: *Poster Session Abstracts of IJCAI-97*. p. 57.

Kautz, H. A., Y. Ruan, D. Achioptas, C. Gomes, B. Selman, and M. Stickel: 2001, 'Balance and Filtering in Structured Satisfiable Problems'. In: *Proc. of the Seventeenth Int. Joint Conf. on Artificial Intelligence*. Seattle, Washington, pp. 351–358.

Kautz, H. A. and B. Selman: 1992, 'Planning as Satisfiability'. In: B. Neumann (ed.): *Proc. of the Tenth European Conf. on Artificial Intelligence*. Vienna, Austria, pp. 359–363.

Kautz, H. A. and B. Selman: 1996, 'Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search'. In: *Proc. of the Thirteenth National Conf. on Artificial Intelligence*. Portland, Oregon, USA, pp. 1202–1207.

Li, C. M. and Anbulagan: 1997, 'Heuristics based on unit propagation for satisfiability problems'. In: *Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence*. pp. 366–371.

Mitchell, D., B. Selman, and H. J. Levesque: 1992, 'Hard and Easy Distributions of SAT Problems'. In: *Proc. of the Tenth National Conf. on Artificial Intelligence*. San Jose, CA, pp. 459–465.

Peugniez, T. J.: 1998, *Solving Non-Boolean Satisfiability Problems with Local Search*. BSc dissertation, Department of Computer Science, University of York.

Prestwich, S.: 2004, 'Local Search on SAT-encoded Colouring Problems'. In: *Theory and Applications of Satisfiability Testing: 6th Int. Conf.* pp. 105–119.

Sebastiani, R.: 1994, 'Applying GSAT to Non-Clausal Formulas'. *Journal of Artificial Intelligence Research* **1**, 309–314.

Selman, B., H. A. Kautz, and B. Cohen: 1994, 'Noise Strategies for Improving Local Search'. In: *Proc. of the Twelfth National Conf. on Artificial Intelligence*. Menlo Park, CA, USA, pp. 337–343.

Selman, B., H. J. Levesque, and D. Mitchell: 1992, 'A new method for solving hard satisfiability problems'. In: *Proc. of the Tenth National Conf. on Artificial Intelligence*. pp. 440–446.

Shaw, P.: 1998, 'Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems'. In: *Principles and Practice of Constraint Programming — CP 1998*. pp. 417–431.

Shaw, P., K. Stergiou, and T. Walsh: 1998, 'Arc Consistency and Quasigroup Completion'. In: *Proceedings of ECAI98 Workshop on Non-binary Constraints*.

Stamm-Wilbrandt, H.: 1993, 'Programming in Propositional Logic, or Reductions: Back to the Roots (Satisfiability)'. Technical report, Insitut für Informatik III, Universität Bonn.

Stock, P. G.: 2000, *Solving Non-Boolean Satisfiability with the Davis-Putnam Method*. BSc dissertation, Dept. of Computer Science, Univ. of York.

Van Hentenryck, P. and L. Michel: 2003, 'Control Abstractions for Local Search'. In: *Principles and Practice of Constraint Programming — CP 2003*. pp. 65–80.

Walser, J. P.: 1997, 'Solving Linear Pseudo-Boolean Constraint Problems with Local Search'. In: *Proc. of the Fourteenth National Conf. on Artificial Intelligence*. pp. 269–274.

Walser, J. P.: 1999, *Integer Optimization by Local Search: A Domain-Independent Approach*. Heidelberg: Springer-Verlag.