A Compact Reformulation of Propositional Satisfiability as Binary Constraint Satisfaction ⋆

Matti Järvisalo and Ilkka Niemelä

Laboratory for Theoretical Computer Science,
P.O. Box 5400, FI-02015 Helsinki University of Technology, Finland
matti.jarvisalo@hut.fi,ilkka.niemela@hut.fi

Abstract. A binary CSP encoding for the propositional satisfiability problem (SAT) is introduced. To our knowledge, our encoding is the first one that is linear in the number of variables, domain size and constraint size w.r.t. the size of the SAT instance. Nevertheless, our encoding has the same attractive properties as the well-known hidden variable encoding when comparing the performance of (i) the Davis-Putnam procedure with the Forward Checking and Maintaining Arc Consistency algorithms on the encoding, and (ii) the local search methods WalkSAT and GSAT with the Min Conflicts algorithm. Moreover, considering 2-SAT, it is shown that a generalisation of Papadimitriou's Random Walk algorithm has a quadratic expected running time on our encoding.

1 Introduction

Applications of constraint satisfaction (CSP) often involve a mixture of Boolean constraints and other finite domain constraints. In order to develop an efficient solver for such problems one can encode all the constraints as Boolean constraints in conjunctive normal form (CNF) and exploit efficient Boolean satisfiability (SAT) checking techniques that have emerged in recent years, see e.g., chaff [15] and BerkMin [10]. For interesting reformulations of CSPs as SAT, see e.g. [9, 22, 4].

A problem in this approach is that the translation to CNF often leads to a substantial increase in the instance size especially for finite domain constraints with large domains. Another approach is to encode such instances as binary CSPs so that efficient CSP search techniques using e.g. filtering methods can be exploited. A number of interesting reformulations of Boolean constraints and other non-binary constraints as binary ones have been introduced and studied [2, 3, 22, 21, 20, 1]. It has been shown that, e.g., the hidden variable encoding of Boolean constraints in CNF to binary CSP has attractive computational properties [21]. For example, forward checking applied to the resulting binary CSP is as efficient as the Davis-Putnam procedure [8] with unit propagation for the original SAT

⋆ The financial support from Academy of Finland under the project *Applications of Rule-Based Constraint Programming* (#53695) is gratefully acknowledged.

instance. A remaining problem is that encodings with nice computational properties (such as the hidden variable encoding) lead to an exponential increase in the size of the CSP w.r.t. the length of the CNF clauses in the SAT instance.

In many promising applications of SAT/CSP techniques, such as planning [12], hardware verification [6], testing [13], and bounded model checking [7], long clauses naturally emerge. It is well-known that an arbitrary SAT instance in CNF can be translated into a SAT instance in 3-CNF in which all clauses are of length 3. However, this increases the number of Boolean variables and CNF clauses and can adversely affect the solution time for the instance.

In this paper we develop a novel reformulation of CNF SAT instances as binary CSPs with the aim of avoiding the exponential blow-up or introduction of new Boolean variables when handling long clauses while preserving the attractive computational properties of earlier promising encodings (such as the hidden variable encoding).

The rest of the paper is organised as follows. First, as preliminaries we introduce concepts related to propositional satisfiability and constraint satisfaction (Section 2), and review the known CSP encodings for SAT (Section 3). We then proceed by presenting our novel encoding (Section 4), and providing a theoretical comparison of our encoding and the previous ones (Section 5).

2 Preliminaries

In this section we review basic concepts related to *propositional satisfiability* and *constraint satisfaction*.

2.1 Propositional Satisfiability

A *literal* is a propositional variable v (a *positive literal*) or its negation, $\neg v$ (a *negative literal*). A *clause* of length k is a disjunction $\bigvee_{i=1}^{k} l_i$ of k literals l_i. The *place* of a literal l_i in a clause $\bigvee_{i=1}^{k} l_i$ is the index i. A clause of length one is a *unit clause*. A propositional formula in *conjunctive normal form* (a CNF formula) is a set $\mathcal{F} = \{C_1, \ldots, C_m\}$, where each C_i is a clause.

A *truth assignment* over a set of propositional variables $\mathcal{V}$ is a function $\pi : \mathcal{V} \to \{0, 1\}$. A positive (negative) literal v ($\neg v$) is *satisfiable* under a truth assignment π if $\pi(v) = 1$ ($\pi(v) = 0$). A clause $C = \bigvee_{i=1}^{k} l_i$ is *satisfiable* under π if some l_i is satisfiable. A CNF formula $\mathcal{F}$ is satisfiable under π, and π is called a *satisfiable truth assignment* for $\mathcal{F}$, if every clause in $\mathcal{F}$ is satisfiable under π.

The SAT *problem* is the following. *Given a CNF formula $\mathcal{F}$, is $\mathcal{F}$ satisfiable?* In the k-SAT *problem*, each clause in $\mathcal{F}$ is of length k.

Most state-of-the-art systematic SAT solvers today are based on the *Davis-Putnam procedure* (DP) [8]. DP is a search method that includes a pruning rule in addition to branching; DP does *unit propagation*, i.e., given a unit clause l all clauses having l are removed and $\neg l$ is removed from each clause containing it.[1]

[1] We do not consider additional rules such as subsumption or affirmative-negative, as they do not affect the efficiency of DP in theoretical terms.

In addition to DP, we consider in this paper the local search methods WalkSAT [18], GSAT [19], and RW [16] for SAT. Let $\Delta(v)$ be the change in the number of satisfied clauses achieved by flipping the truth value of the variable v. All of the methods start with a randomly chosen truth assignment, and do the following loops, respectively.

WalkSAT. Choose an unsatisfied clause at random, and (i) with probability p, where $0 < p < 1$, choose a variable within the clause at random, and flip its truth value, or (ii) with probability $1 - p$ choose at random one of the variables within the clause for which Δ is the greatest and flip its truth value.
GSAT. Choose at random one of the variables for which Δ is the greatest and flip its truth value.
RW. Choose an unsatisfied clause and a variable within the clause at random, and flip its truth value.

2.2 Constraint Satisfaction

A *constraint satisfaction problem* (CSP) consists of a finite *set of variables* $\mathcal{X} = \{x_1, \ldots x_n\}$, each x_i taking on values from the domain $\mathrm{Dom}(x_i)$, and a finite collection of *constraints* (relations) $\mathcal{R} = \{R_{X_1}, \ldots R_{X_m}\}$. Each R_{X_i} is defined over a tuple of variables $X_i = (x_{i_1}, \ldots, x_{i_k})$ by a subset of the Cartesian product $\mathrm{Dom}(x_{i_1}) \times \cdots \times \mathrm{Dom}(x_{i_k})$. The number of variables k in the tuple is the *arity* of the constraint. In a *binary* CSP, each constraint is of arity two (a *binary constraint*). A *non-binary* CSP contains constraints of arity greater than two. A CSP can be represented by a *constraint (hyper) graph*, the nodes of which correspond to variables and the edges to constraints.

An assignment ϕ over a subset of variables $\mathcal{X}' \subseteq \mathcal{X}$, where $\phi(x_i) \in \mathrm{Dom}(x_i)$ for each $x_i \in \mathcal{X}'$, is *consistent* with a constraint $R_{(x_{i_1}, \ldots x_{i_k})}$ if

(i) $\phi(x_{i_j})$ is defined for all x_{i_j}, and
(ii) $(\phi(x_{i_k}), \ldots, \phi(x_{i_1})) \in R_{(x_{i_1}, \ldots x_{i_k})}$, i.e., the assignment ϕ *satisfies* the constraint.

A *solution* to a CSP is an assignment over all the variables that is consistent with each of the constraints. A binary relation $R_{(x,y)}$ is *arc consistent* if

(i) for each $a \in \mathrm{Dom}(x)$, there is an element $b \in \mathrm{Dom}(y)$ such that $(a, b) \in R$, and
(ii) for each $b \in \mathrm{Dom}(y)$, there is an element $a \in \mathrm{Dom}(x)$ such that $(a, b) \in R$.

A binary CSP is *FC consistent* if all its constraints which have exactly one unassigned variable are arc consistent. A binary CSP is *arc consistent* if all its constraints are arc consistent. An arc consistent binary CSP is thus also FC consistent.

A local consistency property can be *enforced* if there is a function LC from CSPs to CSPs such that if $\mathcal{C}$ is a CSP, then $\mathrm{LC}(\mathcal{C})$ is a new CSP with the *same set of solutions*. Applying such a function is called *enforcing* the particular local consistency. Furthermore, it is required that enforcing a local consistency property must result in a CSP satisfying the property, and that if $\mathcal{C}$ satisfies the property, then $\mathrm{LC}(\mathcal{C}) = \mathcal{C}$. For the consistency properties considered in this paper, enforcement involves transformations that may only reduce the domains of some of the variables.

For systematic search methods for CSPs, we consider in this paper *maintaining arc consistency* (MAC) [17] and *forward checking* (FC) [11]. In addition to branching, MAC (or *full lookahead*) enforces arc consistency, while FC enforces FC consistency.

For local search methods for CSPs we consider Min Conflicts (MC) [14] and a generalisation of RW for CSPs. These methods start with a random assignment over all the variables, and then do the following loops, respectively.

MC. Pick a variable in a violated constraint at random, and select at random one of the values for the variable that most reduces the number of violated constraints.
Generalised RW. Pick a violated constraint and a variable within the constraint at random, and randomly select some other value for the variable.

3 Previous CSP Encodings for SAT

For the sake of comparison, we review in this section the previously known *dual, hidden variable* (see e.g. [22]), and *literal* (introduced in [2, 3]) binary CSP encodings and a *non-binary* CSP encoding for SAT. For the following, let $\mathcal{F} = \{C_1, \ldots, C_m\}$ be a CNF formula over the set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$.

Dual encoding. A variable c_i is associated with each clause $C_i \in \mathcal{F}$. The domain of c_i consists of the truth assignments over the variables in C_i that satisfy C_i. For every pair of clauses C_i, C_j that share a variable v the constraint

$$R_{(c_i,c_j)} = \mathrm{Dom}(c_i) \times \mathrm{Dom}(c_j) \setminus \{(\pi_i, \pi_j) \mid \pi_i(v) \neq \pi_j(v)\}$$

is introduced.

Hidden variable encoding. A variable x_i with domain $\{0, 1\}$ is associated with each $v_i \in \mathcal{V}$. As in the dual encoding, a variable c_i is associated with each clause $C_i \in \mathcal{F}$. The domain of c_i consists of those truth assignments over the variables appearing in C_i that satisfy C_i. For each clause $C_i = \bigvee_{j=1}^{k_i} l_j^i \in \mathcal{F}$,

(i) given that l_j^i is a positive literal v, the constraint

$$R_{(c_i,x)} = \mathrm{Dom}(c_i) \times \{0, 1\} \setminus \{(\pi, y) \mid \pi(v) \neq y\}$$

is introduced, where x is the variable associated with v, and

(ii) given that l_j^i is a negative literal $\neg v$, the constraint

$$R_{(c_i,x)} = \mathrm{Dom}(c_i) \times \{0, 1\} \setminus \{(\pi, y) \mid \pi(v) = y\}$$

is introduced, where x is the variable associated with v.

Literal encoding. A variable c_i is associated with each $C_i \in \mathcal{F}$. The domain of c_i consists of those literals that satisfy the clause C_i. For example, if $C_i = v_1 \vee \neg v_2$, the domain of c_i is $\{v_1, \neg v_2\}$. The constraint

$$R_{(c_i,c_j)} = \mathrm{Dom}(c_i) \times \mathrm{Dom}(c_j) \setminus \{(v, \neg v), (\neg v, v) \mid v \in \mathcal{V}\}$$

is introduced for every pair of clauses C_i, C_j that contain complementary literals.

Non-binary encoding. A variable x_i with domain $\{0,1\}$ is associated with each variable $v_i \in \mathcal{V}$. Let $\mathrm{Vars}(C)$ denote the set of variables that appear in a clause C. For each subset of variables $V = \{v_{j_1}, \ldots, v_{j_{k_j}}\} \subseteq \mathcal{V}$ such that

$$I_V = \{i \mid \{v_{j_1}, \ldots, v_{j_{k_j}}\} = \mathrm{Vars}(C_i)\} \neq \emptyset,$$

the constraint

$$R_{(x_{j_1},\ldots,x_{j_{k_j}})} = \bigcap_{i \in I_V} \{\pi \mid \pi \text{ is a truth assignment over } V \text{ and satisfies } C_i\}$$

is introduced, where each x_{j_i} is the variable associated with v_{j_i}.

4 A New Binary CSP Encoding for SAT

Defined as follows, we call our new binary CSP encoding for SAT the *place encoding.*

Place encoding. A variable x_i is associated with each $v_i \in \mathcal{V}$. If v_i appears in a unit clause l, then the domain of x_i has only the value that satisfies l, i.e.,

(i) $\mathrm{Dom}(x_i) = \{1\}$ if l is a positive literal v_i, and
(ii) $\mathrm{Dom}(x_i) = \{0\}$ if l is a negative literal $\neg v_i$.

The domain of each x_i is $\{0,1\}$. A variable c_i with domain $\{1, \ldots, k_i\}$ is associated with each clause $C_i \in \mathcal{F}$, where $k_i \geq 2$ is the length of C_i. For each clause $C_i = \bigvee_{j=1}^{k_i} l_j^i \in \mathcal{F}$ with $k_i \geq 2$,

(i) given that l_j^i is a positive literal v, the constraint

$$R_{(c_i,x)} = \{1, \ldots, k_i\} \times \{0,1\} \setminus \{(j,0)\}$$

is introduced, where x is the variable associated with v, and
(ii) given that l_j^i is a negative literal $\neg v$, the constraint

$$R_{(c_i,x)} = \{1, \ldots, k_i\} \times \{0,1\} \setminus \{(j,1)\}$$

is introduced, where x is the variable associated with v.

Example 1. The place encoding of the CNF formula

$$\{v_1 \vee v_2, \neg v_1 \vee v_3, v_2 \vee \neg v_3 \vee \neg v_4, \neg v_1 \vee \neg v_2 \vee \neg v_3, v_4\}$$

has the set of variables $\{x_1, x_2, x_3, x_4, c_1, c_2, c_3, c_4\}$ with domains $\{0, 1\}$ for x_1, x_2, x_3, $\{1\}$ for x_4, $\{1, 2\}$ for c_1, c_2, and $\{1, 2, 3\}$ for c_3, c_4, and has the following constraints.

$$\begin{aligned}
R_{(c_1,x_1)} &= \{(1,1),(2,0),(2,1)\}\\
R_{(c_1,x_2)} &= \{(1,0),(1,1),(2,1)\}\\
R_{(c_2,x_1)} &= \{(1,0),(2,0),(2,1)\}\\
R_{(c_2,x_3)} &= \{(1,0),(1,1),(2,1)\}\\
R_{(c_3,x_2)} &= \{(1,1),(2,0),(2,1),(3,0),(3,1)\}\\
R_{(c_3,x_3)} &= \{(1,0),(1,1),(2,0),(3,0),(3,1)\}\\
R_{(c_3,x_4)} &= \{(1,0),(1,1),(2,0),(2,1),(3,0)\}\\
R_{(c_4,x_1)} &= \{(1,0),(2,0),(2,1),(3,0),(3,1)\}\\
R_{(c_4,x_2)} &= \{(1,0),(1,1),(2,0),(3,0),(3,1)\}\\
R_{(c_4,x_3)} &= \{(1,0),(1,1),(2,0),(2,1),(3,0)\}
\end{aligned}$$

The name *place* encoding comes from the fact that in the domain of a variable associated with a clause in the encoding, each value specifies the place of a literal in the clause that should be satisfied.

The constraint graph of the CSP above is shown in Figure 1. Notice that the constraint graph is always bipartite, and of exactly the same structure as with the hidden variable encoding.

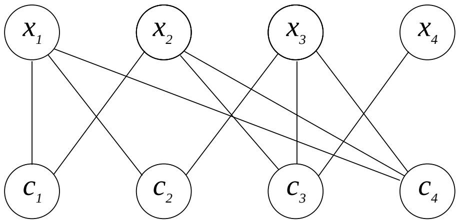

Fig. 1. Constraint graph of the place encoding in Example 1.

However, there is a fundamental difference between these two encodings. In the hidden variable encoding, each variable associated with a clause has a domain of exponential w.r.t. k, the length of the clause. The values in the domain correspond to the $2^k - 1$ satisfying truth assignments over the variables that appear in the clause. On the other hand, in the place encoding each variable associated with a clause has a domain of linear size w.r.t. k, each value specifying the place of a literal that at least must be satisfied in order to satisfy the clause. For instance, consider the clause $v_2 \vee \neg v_3 \vee \neg v_4$. Using the shorthand

$$\begin{array}{ll} \pi_0 = \{v_2 \to 0, v_3 \to 0, v_4 \to 0\} & \pi_4 = \{v_2 \to 1, v_3 \to 0, v_4 \to 0\} \\ \pi_1 = \{v_2 \to 0, v_3 \to 0, v_4 \to 1\} & \pi_5 = \{v_2 \to 1, v_3 \to 0, v_4 \to 1\} \\ \pi_2 = \{v_2 \to 0, v_3 \to 1, v_4 \to 0\} & \pi_6 = \{v_2 \to 1, v_3 \to 1, v_4 \to 0\} \\ \pi_3 = \{v_2 \to 0, v_3 \to 1, v_4 \to 1\} & \pi_7 = \{v_2 \to 1, v_3 \to 1, v_4 \to 1\}, \end{array}$$

in the hidden variable encoding the domain of the associated clause c_3 is

$$\{\pi_0, \pi_1, \pi_2, \pi_4, \pi_5, \pi_6, \pi_7\}.$$

This difference might have implications in practice. Notice that when branching on a variable associated with a clause in the hidden variable encoding (if enabled), enforcing arc consistency reduces the domain of each variable that appears in the clause into a singleton set in each branch (of which there are an exponential number). On the other hand, when enforcing arc consistency after branching on a variable associated with a clause in the place encoding, the domain of a single variable in the clause is reduced into a singleton set in each branch (of which there are a linear number).

Last, notice that the size of a constraint dealing with a variable associated with a clause of length k is $2k-1$ for the place encoding and 2^k-1 for the hidden variable encoding. For instance, again considering the clause $v_2 \vee \neg v_3 \vee \neg v_4$, in the hidden variable encoding we have the constraints

$$\begin{aligned} R_{(c_3,x_2)} &= \{(\pi_0,0),(\pi_1,0),(\pi_2,0),(\pi_4,1),(\pi_5,1),(\pi_6,1),(\pi_7,1)\} \\ R_{(c_3,x_3)} &= \{(\pi_0,0),(\pi_1,0),(\pi_2,1),(\pi_4,0),(\pi_5,0),(\pi_6,1),(\pi_7,1)\} \\ R_{(c_3,x_4)} &= \{(\pi_0,0),(\pi_1,1),(\pi_2,0),(\pi_4,0),(\pi_5,1),(\pi_6,0),(\pi_7,1)\}, \end{aligned}$$

where we have used the shorthand above.

Theorem 1. *Let $\mathcal{F}$ be a CNF formula and $\mathcal{C}$ the place encoding of $\mathcal{F}$. Then $\mathcal{F}$ is satisfiable if and only if $\mathcal{C}$ has a solution.*

Proof. Let π be a satisfying truth assignment for $\mathcal{F}$. Assign values to x_is according to π, i.e., assign $x_i \leftarrow 0$ if $\pi(v_i) = 0$, and $x_i \leftarrow 1$ if $\pi(v_i) = 1$. Now consider an arbitrary clause $C = \bigvee_{i=1}^{k} l_i$ with $k \geq 2$ in $\mathcal{F}$ (the case of unit clauses is trivial by the translation). Let c be the variable in $\mathcal{C}$ associated with C. As π satisfies $\mathcal{F}$, π must satisfy C, and furthermore π must satisfy some l_i. Consider any such l_i. Let l_i be a positive (negative) literal v ($\neg v$). Let x be the variable in $\mathcal{C}$ associated with v. By the translation, we have $(i,1) \in R_{(c,x)}$ ($(i,0) \in R_{(c,x)}$). Assign $c \leftarrow i$. By the translation, $\{(i,0),(i,1)\} \subset R_{(c,x')}$ for any other variable x' associated with a variable v' that appears in C. Thus we do not break any of the constraints in $\mathcal{C}$.

For the other direction, take any solution to $\mathcal{C}$. Consider any variable c in $\mathcal{C}$ associated with a clause C in $\mathcal{F}$, and all constraints $R_{(c,x_i)}$ dealing with c. In each such $R_{(c,x_i)}$ there appears only one of $(j,0),(j,1)$ for a distinct j. Thus the value of j assigned to c in the solution fixes a certain value for the variable x_i associated with the variable v_i appearing in the clause C. By the translation, we satisfy C by assigning the value assigned to x_i in the solution to the variable v_i. □

Remark 1. Note that the place encoding of any CNF formula is arc consistent (and thus also FC consistent).

5 Theoretical Comparison of the Encodings

5.1 Encoding k-SAT

For the following, consider a CNF formula over a set of n variables and with m clauses, in which each clause is of length k (a k-SAT instance). A comparison of the size of the resulting CSP w.r.t. the number of variables (#variables), the size of the variable domains, the number of constraints (#constraints), and the size of the constraints, using the encodings presented in the previous section is shown in Table 1. Notice that out of all the encodings, only the place encoding is linear in all of these parameters (number of variables, domain size, and constraint size). For convenience, for the domain size we write, e.g., $\mathcal{O}(1)$ (n) / $\mathcal{O}(k)$ (m) to notate that the encoding has n variables with domains of size $\mathcal{O}(1)$ and m variables with domains of size $\mathcal{O}(k)$.

Table 1. Comparison of the sizes of the different CSP encodings for k-SAT.

Encoding	#variables	domain size	#constraints	constraint size
place	$n+m$	$\mathcal{O}(1)$ (n) / $\mathcal{O}(k)$ (m)	$\mathcal{O}(kn)$	$\mathcal{O}(k)$
dual	m	$\mathcal{O}(2^k)$	$\mathcal{O}(m^2)$	$\mathcal{O}(2^k)$
hidden variable	$n+m$	$\mathcal{O}(1)$ (n) / $\mathcal{O}(2^k)$ (m)	$\mathcal{O}(kn)$	$\mathcal{O}(2^k)$
literal	m	$\mathcal{O}(k)$	$\mathcal{O}(m^2)$	$\mathcal{O}(k^2)$
non-binary	n	$\mathcal{O}(1)$	$\mathcal{O}(\binom{n}{k})$	$\mathcal{O}(2^k)$

5.2 Systematic Search

In this section we subject the place encoding to the same analysis that is done to the dual, hidden variable, literal, and non-binary encodings in [22, 21]. We show that (i) enforcing FC consistency on the place encoding results in an arc consistent CSP (Theorem 2), that (ii) unit propagation on a CNF formula $\mathcal{F}$ does the same work as enforcing FC consistency on the place encoding of $\mathcal{F}$ (Theorem 3), and thus (iii) DP on a CNF formula $\mathcal{F}$ explores the same number of branches as FC / MAC on the place encoding of $\mathcal{F}$ (Corollary 1). In the sense of these theorems, the place encoding is equivalent to the hidden variable encoding; the equivalent results for the hidden variable encoding are given in [22]. However, as shown in Table 1, variables associated with clauses have domains of exponential size in the hidden variable encoding, while these domains are of linear size in the place encoding.

We begin by showing that enforcing FC consistency on the place encoding results in an arc consistent CSP.

Theorem 2. *Let $\mathcal{F}$ be a CNF formula and $\mathcal{C}$ the place encoding of $\mathcal{F}$. Enforcing FC consistency on $\mathcal{C}$ results in an arc consistent CSP.*

Proof. Let FC($\mathcal{C}$) be the resulting CSPs when enforcing FC consistency on $\mathcal{C}$. Assume that FC($\mathcal{C}$) is not arc consistent, i.e., there is a constraint $R_{(c,x)}$ in FC($\mathcal{C}$), where $|\mathrm{Dom}(c)| \geq 2$ and $\mathrm{Dom}(x) = \{0, 1\}$, such that

(i) for some $b \in \{0, 1\}$, there is no $a \in \mathrm{Dom}(c)$ such that $(a, b) \in R_{(c,x)}$, or
(ii) for some $a \in \mathrm{Dom}(c)$, there is no $b \in \{0, 1\}$ such that $(a, b) \in R_{(c,x)}$,

where c is a variable associated with a clause C and x is a variable associated with a propositional variable v in $\mathcal{F}$. Now consider these two cases.

(i) Since by the place encoding only one of $(a, 0), (a, 1)$ belongs to $R_{(c,x)}$ only if a is the place of v in C, we have $\mathrm{Dom}(c) = \{a\}$, and thus a contradiction.
(ii) Since there is some $b \in \{0, 1\}$ such that $(a, b) \in R_{(c,x)}$ for each $a \in \mathrm{Dom}(c)$ by the place encoding, we should have that $\mathrm{Dom}(x) = \emptyset$, which is a contradiction.

□

Next we argue that unit propagation on a CNF formula $\mathcal{F}$ does the same work as enforcing FC consistency on the place encoding of $\mathcal{F}$. In fact, we show that unit propagation produces a positive literal v (a negative literal $\neg v$) if and only if enforcing FC consistency on the place encoding removes the value 0 (1) from the domain of the variable associated with v. Due to this we say that unit propagation and enforcing FC consistency are *propagation equivalent.*

Theorem 3. *Let $\mathcal{F}$ be a CNF formula and $\mathcal{C}$ the place encoding of $\mathcal{F}$. Unit propagation on $\mathcal{F}$ is propagation equivalent to enforcing FC consistency on $\mathcal{C}$.*

Proof. To unit propagate, we need to assume to have a unit clause, i.e., a positive literal v (negative literal $\neg v$) in $\mathcal{F}$. By the place encoding, we have $\mathrm{Dom}(x) = \{1\}$ ($\mathrm{Dom}(x) = \{0\}$) in $\mathcal{C}$, where x is associated with v. When unit propagating on v ($\neg v$) the literal $\neg v$ (v) is removed from each clause C in which it appears. Let c be the variable associated with an arbitrary such clause C. By the encoding we have that $(i, 0) \in R_{(c,x)}$ and $(i, 1) \notin R_{(c,x)}$ ($(i, 1) \in R_{(c,x)}$ and $(i, 0) \notin R_{(c,x)}$), where i is the place of the literal v ($\neg v$) in C. By having $\mathrm{Dom}(x) = \{1\}$ ($\mathrm{Dom}(x) = \{0\}$), enforcing FC consistency on $\mathcal{C}$ thus removes i from $\mathrm{Dom}(c)$.

Now assume that unit propagating on v ($\neg v$) produces a new literal l. This can be only if we have a clause $C = l \vee v$ ($l \vee \neg v$) in $\mathcal{F}$. Let c be the variable associated with C. By the above, we must have $\mathrm{Dom}(c) = \{i, j\}$, where i, j are the places of v, l in C, respectively. Thus enforcing FC consistency removes i

from the domain of c, and furthermore, having $\mathrm{Dom}(c) = \{j\}$, the value 0 (in case l is positive, or 1 if l is negative) from the domain of x_l associated with l.

This reasoning is straightforward to reverse. □

Now assume branching heuristics that instantiate variables associated with propositional variables before variables associated with clauses. This enables branching identically in a CNF formula and its place encoding. Applying Theorems 2 and 3, it is easy to see that a proof tree for MAC / FC can be mapped into a equivalent size search tree explored by DP, and vice versa. This is simply because branching only on the variables in the place encoding associated with propositional variables, by Theorem 3 we generate the equivalent subproblems by branching and then enforcing arc consistency / enforcing FC consistency / unit propagating.

Corollary 1. *Let $\mathcal{F}$ be a CNF formula and $\mathcal{C}$ the place encoding of $\mathcal{F}$. Given equivalent branching heuristics, we have that MAC and FC, respectively, applied to $\mathcal{C}$ explores the same number of branches as DP on $\mathcal{F}$.*

Combining Corollary 1 with previously known results [21], we summarise the behaviour of MAC and FC on the different CSP encodings with respect to the behaviour of DP on the original CNF formula in Table 2. For notation, considering X vs Y, $X = Y$ denotes that X and Y are equivalent in the above mentioned sense, while $X > Y$ and $X \neq Y$ denote that X is superior to and incomparable with Y, respectively.

Table 2. Summary of the behaviour of MAC and FC on the different CSP encodings with respect to the behaviour of DP on the original CNF formula. For the non-binary encoding, nFC0 and nFC1 refer to certain types of generalisations of forward checking for non-binary CSPs, see [5].

	place	**dual**	**hidden variable**	**literal**	**non-binary**
DP vs MAC	$=$	$\neq$	$=$	$>$	
DP vs FC	$=$	$>$	$=$	$>$	$=$ (nFC0), $<$ (nFC1)

5.3 Local Search

Next we turn to local search methods. As in [21], we say that a local search method B can *simulate* method A if it holds that from any state X, method B can move to state Y in the search space if method A can. We now show that (i) GSAT on the original CNF formula $\mathcal{F}$ cannot simulate MC on the place encoding $\mathcal{C}$ nor vice versa (Theorem 4), and that (ii) MC on $\mathcal{C}$ can simulate WalkSAT on $\mathcal{F}$ but not vice versa (Theorem 5). Again, in the sense of these theorems, the place encoding is equivalent to the hidden variable encoding; the equivalent theorems for the hidden variable encoding are proven in [21]. Here we continue to consider only flipping variables in the place encoding associated with propositional variables.

The justifications given in [21] for the hidden variable encoding apply here also. For Theorem 4, notice that (i) for a CNF formula with two disjoint sets of clauses, one satisfiable and the other unsatisfiable, GSAT can flip a variable occurring in some clause in the satisfiable set while MC cannot, and (ii) conversely, MC may flip a variable that would decrease the number of satisfiable clauses in the original CNF formula while GSAT cannot.

Theorem 4. *MC on the place encoding can neither simulate GSAT on the original CNF formula nor vice versa.*

For Theorem 5, notice that (i) when WalkSAT flips a propositional variable in an unsatisfied clause, MC can also flip the associated variable in the place encoding, while (ii) again, given a CNF formula with two disjoint sets of clauses and an assignment which satisfies only one of these sets, WalkSAT cannot flip a propositional variable in the satisfied set of clauses, while MC can flip the associated variable in the case that one of the variables associated with a clause in the satisfiable set has an assignment that contradicts the given truth assignment.

Theorem 5. *MC on the place encoding can simulate WalkSAT on the original CNF formula, but not vice versa.*

Combining Theorems 4 and 5 with previously known results [21], we can summarise the behaviour of MC on a selection of different CSP encodings with respect to the behaviour of GSAT and WalkSAT on the original CNF formula as in Table 2.

Table 3. Summary of the behaviour of MC on a selection of different CSP encodings with respect to the behaviour of GSAT and WalkSAT on the original CNF formula.

	place	**dual**	**hidden variable**	**literal**	**non-binary**
GSAT vs MC	$\neq$		$\neq$		$\neq$
WalkSAT vs MC	$<$		$<$		$=$

Finally we turn to the computationally tractable case of 2-SAT, and prove a theorem on the upper bound on the number of flips the (generalised) RW algorithm is expected to make on the place encoding of a 2-SAT instance with n variables and m clauses. This analysis is possible as in the case of 2-SAT all variables in the place encoding have domains of size two (whereas in the hidden as well as the dual encoding this kind of analysis is no possible as the variables associated with clauses have domains of size three).

Theorem 6. *Generalised RW is expected to take at most* $(n + m)^2$ *flips to find a satisfying assignment when applied to the place encoding of a satisfiable 2-SAT instance with* n *variables and* m *clauses.*

Proof. Similarly as explained in [16] and [21], the behaviour of RW in the place encoding can be reduced to a one-dimensional random walk with a reflecting and an absorbing barrier. In state i, where $0 \leq i \leq n+m$, we are i flips away from a solution. Letting $N(i)$ be the expected number of steps (flips) to end up in state 0 (to find a solution) when starting from state i, we have that $N(0) = 0$,

$$N(i) \leq \frac{1}{2}(N(i-1) + N(i+1)) + 1 \text{ for } 0 < i < n+m$$

(flipping a variable in an constraint inconsistent with the current assignment brings us closer to a solution at least with probability 50 %), and

$$N(n+m) \leq N(n+m-1) + 1.$$

Considering the solution $N(i) = 2i(n+m) - i^2$ of the recurrent relation in the worst case $N(i) = \frac{1}{2}(N(i-1)+N(i+1))+1$ for $0 < i < n+m$ and $N(n+m) = N(n+m-1)+1$, in the worst case $i = m+n$ we get $N(n+m) = (n+m)^2$. □

6 Conclusions

The paper develops methodology for handling constraint satisfaction problems consisting of a mixture of Boolean constraints and other finite domain constraints. The idea is to map non-binary constraints to binary ones in order to be able to exploit, e.g., advanced filtering techniques developed for binary constraints. A number of promising reformulations of non-binary constraints as binary ones have been developed. However, they are unoptimal when applied to Boolean constraints given in CNF especially when the clauses are long. In this paper we develop a new compact encoding of Boolean CNF constraints to binary constraints which is, to our knowledge, the first one that is linear in the number of variables, domain size, and constraint size w.r.t. to the size of the CNF Boolean constraints. Moreover, the encoding enjoys attractive computational properties. For example, unit propagation, often used in special purpose Boolean SAT solvers, is propagation equivalent to enforcing arc consistency / forward checking on the novel encoding of the SAT problem. This makes it possible to use the FC or the MAC search procedure to obtain as efficient search techniques as the DP procedure on the original SAT procedure by using suitable search heuristics employed for SAT. The novel encoding has nice properties also when using local search methods. For example, MC on the novel encoding can simulate WalkSAT on the original SAT instance and random walk search has a quadratic expected run time on the novel encoding of a 2-SAT instance. Properties are similar to another interesting encoding of non-binary constraints to binary ones, namely the hidden variable encoding. However, the advantage of the novel encoding is that it is linear in the clause length of the SAT instance whereas the hidden variable encoding is exponential. An interesting topic of further research is to study experimentally how the proposed place encoding compares to other CSP encodings of SAT.

References

1. Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. *Artificial Intelligence*, 140:1–37, 2002.
2. Hachemi Bennaceur. The satisfiability problem regarded as a constraint satisfaction problem. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI 1996)*, pages 155–159. John Wiley and Sons, Chichester, 1996.
3. Hachemi Bennaceur. A comparison between SAT and CSP techniques. *Constraints*, 9:123–138, 2004.
4. Christian Bessière, Emmanuel Hebrard, and Toby Walsh. Local consistencies in SAT. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*. Springer, 2004.
5. Christian Bessière, Pedro Meseguer, Eugene C. Freuder, and Javier Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141(1–2):205–224, 2002.
6. Armin Biere and Wolfgang Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of 20th IEEE/ACM International Conference on Computer Aided Design (ICCAD'02), San Jose CA, USA, November 2002.* IEEE Press, 2002.
7. Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, July 2001.
8. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
9. Ian P. Gent. Arc consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002)*, pages 121–125. IOS Press, 2002.
10. Evgueni Goldberg and Yakov Novikov. Berkmin: A fast and robust SAT solver. In *Proceedings of Automation and Test in Europe (DATE 2002)*, pages 142–149, 2002.
11. Robert M. Haralick and Gordon L. Elliot. Increasing tree-search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
12. Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359 – 363, 1992.
13. Tracy Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, January 1992.
14. Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'90)*, 1990.
15. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535. ACM, 2001.
16. Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 163–169. IEEE Computer Society Press, 1991.
17. Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94, Rosario, Orcas Island, Washington, USA*, pages 10–20, 1994.

18. Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *Proceedings of the Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, 1993.
19. Bart Selman, Hector J. Levesque, and David Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446. AAAI Press, 1992.
20. Kostas Stergiou and Toby Walsh. Encodings of non-binary constraint satisfaction problems. In *Proceedings of AAAI/IAAI 1999*, pages 163–168. AAAI Press / MIT press, 1999.
21. Toby Walsh. Reformulating propositional satisfiability as constraint satisfaction. In *Proceedings of Symposium on Abstraction, Reformulation and Approximation (SARA-2000)*, volume 1864 of *Lecture Notes in Computer Science*, pages 233–246. Springer, 2000.
22. Toby Walsh. SAT v CSP. In *Proceedings of Sixth International Conference on Principles and Practice of Constraint Programming (CP-2000)*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.