SAT *v* CSP

Toby Walsh[⋆]

University of York, York, England. `tw@cs.york.ac.uk`

Abstract. We perform a comprehensive study of mappings between constraint satisfaction problems (CSPs) and propositional satisfiability (SAT). We analyse four different mappings of SAT problems into CSPs, and two of CSPs into SAT problems. For each mapping, we compare the impact of achieving arc-consistency on the CSP with unit propagation on the SAT problem. We then extend these results to CSP algorithms that maintain (some level of) arc-consistency during search like FC and MAC, and to the Davis-Putnam procedure (which performs unit propagation at each search node). Because of differences in the branching structure of their search, a result showing the dominance of achieving arc-consistency on the CSP over unit propagation on the SAT problem does not necessarily translate to the dominance of MAC over the Davis-Putnam procedure. These results provide insight into the relationship between propositional satisfiability and constraint satisfaction.

1 Introduction

Despite their proximity, there has been little direct comparison between propositional satisfiability and constraint satisfaction. One of the most immediate differences is that propositional satisfiability (SAT) problems have binary domains and non-binary constraints whilst constraint satisfaction problems (CSPs) typically have binary constraints and non-binary domains. However, a more intimate understanding of the similarities and differences between the two problem domains would aid research in both areas. For example, are local search procedures developed for SAT like WalkSAT and Novelty useful for CSPs?

One method for exploring the relationship between SAT problems and CSPs is to study mappings between them. As each problem class is NP-complete, we can reduce one into the other in polynomial time. However, the details of such reductions and the properties preserved can, as we show here, be very informative. Bennaceur previously looked at encoding SAT problems as CSPs [Ben96], whilst Génisson and Jégou looked at encoding CSPs as SAT problems [GJ96]. However, both these studies were limited to a single mapping. It is therefore instructive to study the range of mappings possible between SAT problems and CSPs. A more complete picture of the relationship between propositional satisfiability and constraint satisfaction then starts to emerge.

[⋆] The author is supported by an EPSRC advanced research fellowship. The author is a member of the APES research group (http://www.cs.strath.ac.uk/~apes) and wishes to thank the other members for their comments and feedback.

2 Formal background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a domain of values, and a set of constraints (relations) on the allowed values for specified subsets of the variables. A binary CSP has only binary constraints. A binary CSP is arc-consistent (AC) iff it has non-empty domains and every binary constraint is arc-consistent. A binary constraint is arc-consistent iff any assignment to one of the variables in the constraint can be extended to a consistent assignment for the other variable. A (non-binary) CSP is generalized arc-consistent (GAC) iff for any variable in a constraint and value that it is assigned, there exist compatible values for all the other variables in the constraint [MM88]. Algorithms for solving CSPs typically maintain some level of consistency at every node in their search tree. For example, the MAC algorithm for binary CSPs maintains arc-consistency at each node in the search tree [Gas79]. The FC algorithm (forward checking) for binary CSPs maintains arc-consistency only on those constraints involving the most recently instantiated variable and those that are uninstantiated. Finally, for non-binary CSPs, the nFC0 algorithm maintains generalized arc-consistency on those constraints involving one uninstantiated variable, whilst the nFC1 algorithm maintains generalized arc-consistency on those constraints and constraint projections involving one uninstantiated variable [BMFL99].

Given a propositional formula, the satisfiability (SAT) problem is to determine if there is an assignment of truth values to the variables that makes the whole formula true [GJ79]. One of the best procedures to solve the SAT problem is the so-called Davis-Putnam (DP) procedure (though it is actually due to Davis, Logemann and Loveland [DLL62]). The DP procedure consists of three main rules: the empty rule (which fails and backtracks when an empty clause is generated), the unit propagation rule (which deterministically assigns any unit literal), and the branching or split rule (which non-deterministically assigns a truth value to a variable). As is often the case in implementations of DP, we will ignore the pure literal and tautology rules as neither are needed for completeness or soundness, nor usually for efficiency.

3 Encoding SAT into CSPs

There are several different ways that a SAT problem can be encoded as a binary or non-binary CSP.

Dual encoding: We associate a dual variable, D_i with each clause c_i. The domain of D_i consists of those tuples of truth values which satisfy the clause c_i. For example, associated with the clause $x_1 \vee x_3$ is a dual variable D_1 with domain $\{\langle T, F\rangle, \langle F, T\rangle, \langle T, T\rangle\}$. These are the assignments for x_1 and x_3 which satisfy the clause $x_1 \vee x_3$. Binary constraints are posted between dual variables which are associated with clauses that share propositional variables in common. For example, between the dual variable D_1 associated with the clause $x_1 \vee x_3$ and the dual variable D_2 associated with the clause $x_2 \vee \neg x_3$ is a binary constraint that the second element of the tuple assigned to D_1 must be the complement of the second element of the tuple assigned to D_2.

Hidden variable encoding: We again associate a dual variable, D_i with each clause c_i, the domain of which consists of those tuples of truth values which satisfy the clause. However, we also have (propositional) variables x_i with domains $\{T, F\}$. A binary constraint is posted between a propositional variable and a dual variable if its associated clause mentions the propositional variable. For example, between the dual variable D_2 associated with the clause $x_2 \vee \neg x_3$ and the variable x_3 is a binary constraint. This constrains the second element of the tuple assigned to D_2 to be the complement of the value assigned to x_3. There are no direct constraints between dual variables.

Literal encoding: We associate a variable, D_i with each clause c_i. The domain of D_i consists of those literals which satisfy the clause c_i. For example, associated with the clause $x_1 \vee x_3$ is a dual variable D_1 with domain $\{x_1, x_3\}$, and associated with the clause $x_2 \vee \neg x_3$ is a dual variable D_2 with domain $\{x_2, \neg x_3\}$. Binary constraints are posted between D_i and D_j iff the associated clause c_i contains a literal whose complement is contained in the associated clause c_j. For example, there is a constraint between D_1 and D_2 as the clause c_1 contains the literal x_3 whilst the clause c_2 contains the complement $\neg x_3$. This constraint rules out incompatible (partial) assignments. For instance, between D_1 and D_2 is the constraint that allows $D_1 = x_1$ and $D_2 = x_2$, or $D_1 = x_1$ and $D_2 = \neg x_3$, or $D_1 = x_3$ and $D_2 = x_2$. However, the assignment $D_1 = x_3$ and $D_2 = \neg x_3$ is ruled out as a nogood. Note that the literal encoding uses variables with smaller domains than the dual or hidden variable encodings. The dual variables have domains of size $O(2^k)$ where k is the clause length, whilst the variables in the literal encoding have domains of size just $O(k)$. This could have a significant impact on runtimes.

Non-binary encoding: The CSP has variables x_i with domains $\{T, F\}$. A non-binary constraint is posted between those variables that occurring together in a clause. This constraint has as nogoods those partial assignments that fail to satisfy the clause. For example, associated with the clause $x_1 \vee x_2 \vee \neg x_3$ is a non-binary constraint on x_1, x_2 and x_3 that has a single nogood $\langle F, F, T\rangle$.

4 Theoretical comparison

We now compare the performance of the Davis-Putnam (DP) procedure against some popular CSP algorithms like FC and MAC on these different encodings. Our analysis divides into two parts. We first compare the effect of unit propagation on the SAT problem and enforcing arc-consistency on the encoding. We then use this result to compare DP (which performs unit propagation at each node in its search tree) with MAC (which enforces arc-consistency at each node) and FC (which enforces a limited form of arc-consistency). When comparing two algorithms that are applied to (possibly) different representations of a problem, we say that algorithm A dominates algorithm B iff algorithm A visits no more branches than algorithm B assuming “equivalent” branching heuristics (we will discuss what we mean by “equivalent” in the proofs of such results as the exact details depend on the two representations). We say that algorithm A strictly dominates algorithm B iff it dominates and there exists one problem on which algorithm A visits strictly fewer branches.

4.1 Dual encoding

We first prove that enforcing arc-consistency on the dual encoding does more work than unit propagation on the original SAT problem. That is, if unit propagation identifies unsatisfiability then enforcing arc-consistency on the dual encoding also does (but there are problems which enforcing arc-consistency will show are insoluble that unit propagation will not). In addition, if unit propagation commits to particular truth assignments then enforcing arc-consistency on the dual encoding eliminates all contradictory values. To be more precise, if unit propagation assigns true (false) to some variable x_i, enforcing arc-consistency on the dual encoding removes those values from the domains of dual variables which correspond to assigning false (true) to x_i. When we come to assign these dual variables, the value chosen will correspond to assigning true (false) to x_i.

Theorem 1.

1. *If unit propagation commits to particular truth assignments, then enforcing arc-consistency on the dual encoding eliminates all contradictory values.*
2. *If unit propagation generates the empty clause then enforcing arc-consistency on the dual encoding causes a domain wipeout (but the reverse does not necessarily hold).*

Proof. 1. Suppose unit propagation makes a sequence of assignments: l_1, l_2, $\ldots l_j$. The proof uses induction on j. Consider the final assignment l_j. Unit propagation makes this assignment because l_j occurs in a clause $l'_1 \vee \ldots l'_k$ in which all the other literals l'_i $(\neq l_j)$ have been assigned to false. Consider the dual variable associated with this clause. By the induction hypothesis, enforcing arc-consistency eliminates all values which assign any of the l'_i $(\neq l_j)$ to true. Hence, the only value left in the domain of this dual variable assigns l'_i $(\neq l_j)$ to false, and l_j to true. Enforcing arc-consistency on the dual encoding thus eliminates all contradictory values.

2. Unit propagation generates an empty clause if there is a clause, $l_1 \vee \ldots l_k$ in which unit propagation assigns each literal l_i to false. Consider the dual variable associated with this clause. By the first result, enforcing arc-consistency on the dual encoding eliminates all contradictory values. Hence this dual variable has a domain wipeout. To show that the reverse may not hold, consider all possible 2-SAT clauses in 2 variables: $x_1 \vee x_2$, $\neg x_1 \vee x_2$, $x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_2$. This problem does not contain any unit clauses so unit propagation does nothing. However, enforcing arc-consistency on the dual encoding causes a domain wipeout.

There are difficulties in extending this result to algorithms that maintain (some form of) arc-consistency at each node of their search tree (like FC and MAC) and those that perform unit propagation at each node (like DP). One complication is that branching in DP can instantiate variables in any order, but branching on the dual encoding must follow the order of variables in the clauses. In addition, branching on the dual encoding effectively instantiates all the variables in a clause at once. In DP, by comparison, we can instantiate a strict subset of the variables that occur in a clause. Consider, for example, the two clauses $x_1 \vee \ldots x_k$ and $y_1 \vee \ldots y_k$. DP can instantiate the x_i and y_j in any order. By comparison, branching on the dual encoding either instantiates all the x_i before the y_j or vice versa. Similar observations hold for the literal encodings. In the following results, therefore, we start from a branching heuristic for the dual encoding and construct an "equivalent" branching heuristic for DP. It is not always possible to perform the reverse (i.e. start from a DP heuristic and construct an equivalent heuristic for the dual encoding).

Theorem 2. *Given equivalent branching heuristics, DP strictly dominates FC applied to the dual encoding.*

Proof. We show how to take the search tree explored by FC and map it onto a proof tree for DP with no more branches. The proof proceeds by induction on the number of branching points in the tree. Consider the root. Assume FC branches on the variable D_i associated with the SAT clause $l_1 \vee l_2 \vee \ldots \vee l_k$. There are $2^k - 1$ children. We can build a corresponding proof subtree for DP with at most $2^k - 1$ branches. In this subtree, we branch left at the root assigning l_1, and right assigning $\neg l_1$. On both children, we branch left again assigning l_2 and right assigning $\neg l_2$ unless l_2 is assigned by unit propagation (in which case, we move on to l_3). And so on through the l_i until either we reach l_k or unit propagation constructs an empty clause. Note that we do not need to split on l_k as unit propagation on the clause $l_1 \vee l_2 \vee \ldots \vee l_k$ forces this instantiation automatically. In the induction step, we perform the same transformation except some of the instantiations in the DP proof tree may have been performed higher up and so can be ignored. FC on the dual encoding removes some values from the domains of future variables, but unit propagation in DP also effectively makes the same assignments. The result is a DP proof tree which has no more branches than the tree explored by FC. To show strictness, consider a 2-SAT problem with all possible clauses in two variables: e.g. $x_1 \vee x_2$, $\neg x_1 \vee x_2$, $x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_2$. DP explores 2 branches showing that this problem is unsatisfiable, irrespective of the branching heuristic. FC, on the other hand, explores 3 branches, again irrespective of the branching heuristic.

Theorem 2 shows that DP, in a slightly restricted sense, dominates FC applied to the dual encoding. What happens if we maintain a higher level of consistency in the dual encoding than that maintained by FC? Theorem 1 shows that enforcing arc-consistency on the dual encoding does more work than unit propagation. This would suggest that MAC (which enforces arc-consistency at each node) might outperform DP (which performs unit propagation at each node). DP's branching can, however, be more effective than MAC's. As a consequence, there are problems on which DP outperforms MAC, and problems on which MAC outperforms DP, in both cases irrespective of the branching heuristics used.

Theorem 3. *MAC applied to the dual encoding is incomparable to DP.*

Proof. Consider a k-SAT problem with all 2^k possible clauses: $x_1 \vee x_2 \vee \ldots \vee x_k$, $\neg x_1 \vee x_2 \vee \ldots \vee x_k$, $x_1 \vee \neg x_2 \vee \ldots \vee x_k$, $\neg x_1 \vee \neg x_2 \vee \ldots \vee x_k$, $\ldots \neg x_1 \vee \neg x_2 \vee \ldots \vee \neg\neg x_k$. DP explores 2^{k-1} branches showing that this problem is unsatisfiable irrespective of the branching heuristic. If $k = 2$, MAC proves that the problem is unsatisfiable without search. Hence, MAC can outperform DP. If $k > 2$, MAC branches on the first variable (whose domain is of size $2^k - 1$) and backtracks immediately. Hence MAC takes $2^k - 1$ branches, and is outperformed by DP.

4.2 Hidden variable encoding

We first prove that enforcing arc-consistency on the hidden variable encoding does the same work as unit propagation on the original SAT problem. In particular, unit propagation identifies unsatisfiability if and only if enforcing arc-consistency also does, whilst unit propagation commits to particular truth assignments if and only if enforcing arc-consistency on the hidden variable encoding eliminates all contradictory values.

Theorem 4.

1. *Unit propagation commits to a particular truth assignment if and only if enforcing arc-consistency on the hidden variable encoding eliminates all contradictory values.*
2. *Unit propagation generates the empty clause if and only if enforcing arc-consistency on the hidden variable encoding causes a domain wipeout.*

Proof. 1. Suppose unit propagation makes a sequence of assignments: $l_1, l_2, \ldots l_j$. The proof uses induction on j. Consider the final assignment l_j. Unit propagation makes this assignment because l_j occurs in a clause $l'_1 \vee \ldots l'_k$ in which all the other literals l'_i $(\neq l_j)$ have been assigned to false. Consider the hidden variable encoding. By the induction hypothesis, enforcing arc-consistency reduces the domain of each l'_i $(\neq l_j)$ to false. Enforcing arc-consistency therefore removes any value from the domain of the dual variable associated with the clause $l'_1 \vee \ldots l'_k$ which assigns l'_i $(\neq l_j)$ to true. Hence, the only value left in the domain of this dual variable assigns l'_i $(\neq l_j)$ to false, and l_j to true. Enforcing arc-consistency on the constraint between this dual variable and l_j reduces the domain of l_j to true. Hence, enforcing arc-consistency on the hidden variable encoding eliminates all contradictory values. The proof reverses in a straightforward manner.

2. Unit propagation generates an empty clause if there is a clause, $l_1 \vee \ldots l_k$ in which unit propagation assigns each literal l_i to false. Consider the dual variable associated with this clause. By the first result, enforcing arc-consistency on the hidden variable encoding reduces the propositional variable associated with each

literal l_i to the appropriate singleton domain. Hence enforcing arc-consistency between these propositional variables and the dual variable in the hidden variable encoding causes a domain wipeout for the dual variable. The proof again reverses in a straightforward manner.

This result can be extended to algorithms that maintain (some level of) arc-consistency during search, provided we restrict ourselves to branching heuristics that instantiate propositional variables before the associated dual variables. It is then unproblematic to branch in an identical fashion in the hidden variable encoding and in the SAT problem.

Theorem 5. *Given equivalent branching heuristics, MAC applied to the hidden variable encoding explores the same number of branches as DP.*

Proof. We show how to take the search tree explored by DP and map it onto a proof tree for MAC with the same number of branches (and vice versa). The proof proceeds by induction on the number of propositional variables. In the step case, consider the first variable branched upon by DP or MAC. The proof divides into two cases. Either the first branch leads to a solution. Or we backtrack and try both truth values. In either case, as unit propagation and enforcing arc-consistency reduce both problems equivalently, we have "equivalent" subproblems. As these subproblems have one fewer variable, we can appeal to the induction hypothesis.

What happens if we maintain a lower level of consistency in the hidden variable encoding that that maintained by MAC? For example, what about the FC algorithm which enforces only a limited form of arc-consistency at each node? Due to the topology of the constraint graph of a hidden variable encoding, with equivalent branching heuristic, FC can be made to explore the same number of branches as MAC.

Theorem 6. *Given equivalent branching heuristics, FC applied to the hidden variable encoding explores the same number of branches as MAC.*

Proof. In FC, we need a branching heuristic which chooses first any propositional variable with a singleton domain. This makes the same commitments as unit propagation, without introducing any branching points. By Theorem 4, unit propagation is equivalent to enforcing arc-consistency on the hidden variable encoding. Hence, FC explores a tree with the same number of branches as MAC.

4.3 Literal encoding

As with the hidden variable encoding, enforcing arc-consistency on the literal encoding does the same work as unit propagation on the original SAT problem. In particular, unit propagation identifies unsatisfiability if and only if enforcing arc-consistency on the literal encoding also does, whilst unit propagation commits to a particular (partial) truth assignment if and only if enforcing arc-consistency on the literal encoding eliminates all contradictory values.

Theorem 7.

1. *Unit propagation commits to particular truth assignments if and only if enforcing arc-consistency on the literal encoding eliminates all contradictory values.*
2. *Unit propagation generates the empty clause if and only if enforcing arc-consistency on the literal encoding causes a domain wipeout.*

Proof. 1. Suppose unit propagation makes a sequence of assignments: $l_1, l_2, \ldots l_j$. The proof uses induction on j. Consider the final assignment l_j. Unit propagation makes this assignment because l_j occurs in a clause $l'_1 \vee \ldots l'_k$ in which all the other literals l'_i $(\neq l_j)$ have been assigned to false. Consider the literal encoding. By the induction hypothesis, enforcing arc-consistency removes l'_i $(\neq l_j)$ from the domain of the variables D_i associated with the clause $l'_1 \vee \ldots l'_k$. D_i therefore has the singleton domain $\{l_j\}$. Enforcing arc-consistency with any constraint between this dual variable and another that contains $\neg l_j$ removes $\neg l_j$ from the domain. Hence, enforcing arc-consistency on the literal encoding eliminates all contradictory values. The proof reverses in a straightforward manner.

2. Unit propagation generates an empty clause if there is a clause, $l_1 \vee \ldots l_k$ in which unit propagation assigns each literal l_i to false. Consider the variable D_i associated with this clause. By the first result, enforcing arc-consistency on the literal encoding eliminates each literal from its domain. This causes a domain wipeout. The proof again reverses in a straightforward manner.

When we consider algorithms that maintain arc-consistency at each node, we discover that DP can branch more effectively than MAC on the literal encoding (as we discovered with the dual encoding). Since unit propagation in the SAT problem is equivalent to enforcing arc-consistency on the literal encoding, DP dominates MAC applied to the literal encoding.

Theorem 8. *Given equivalent branching heuristic, DP strictly dominates MAC applied to the literal encoding.*

Proof. We show how to take the search tree explored by MAC and map it onto a proof tree for DP with no more branches. The proof proceeds by induction on the number of branching points in the tree. Consider the root. Assume MAC branches on the variable D_i associated with the SAT clause $l_1 \vee l_2 \vee \ldots \vee l_k$. There are k children, the ith child corresponding to the value l_i assigned to D_i. We can build a corresponding proof subtree for DP with k branches. In this subtree, we branch left at the root assigning l_1, and right assigning $\neg l_1$. On the right child, we branch left again assigning l_2 and right assigning $\neg l_2$. And so on through the l_i until we reach l_k. However, we do not need to split on l_k as unit propagation on the clause $l_1 \vee l_2 \vee \ldots \vee l_k$ forces this instantiation automatically. Schematically, this transformation is as follows:

$$node(l_1, l_2, \ldots, l_k) \;\Rightarrow\; node(l_1, node(l_2, \ldots node(l_{k-1}, l_k) \ldots)).$$

In the induction step, we perform the same transformation except: (a) some of the instantiations in the DP proof tree may have been performed higher up and so can be ignored, and (b) the complement of some of the instantiations may have been performed higher up and so we can close this branch by unit propagation. The result is a DP proof tree which has no more branches than the tree explored by MAC. To prove strictness, consider a k-SAT problem with all 2^k possible clauses where $k > 2$. DP explores 2^{k-1} branches showing that this problem is unsatisfiable irrespective of the branching heuristic. However, MAC takes $k!$ branches whatever variable and value ordering we use.

4.4 Non-binary encoding

If the SAT problem contains clauses with more than two literals, the non-binary encoding contains non-binary constraints. Hence, we compare unit propagation on the SAT problem with enforcing generalized arc-consistency on the non-binary encoding. Not surprisingly, generalized arc-consistency on the non-binary encoding dominates unit propagation.

Theorem 9.

1. *If unit propagation commits to particular truth assignments then enforcing generalized arc-consistency on the non-binary encoding eliminates all contradictory truth values.*
2. *If unit propagation generates the empty clause then enforcing generalized arc-consistency on the non-binary encoding causes a domain wipeout (but the reverse does not necessarily hold).*

Proof. 1. Suppose unit propagation makes a sequence of assignments: $l_1, l_2, \ldots l_j$. The proof uses induction on j. Consider the final assignment l_j. Unit propagation makes this assignment because l_j occurs in a clause $l'_1 \vee \ldots l'_k$ in which all the other literals l'_i $(\neq l_j)$ have been assigned to false. Consider the non-binary encoding. By the induction hypothesis, enforcing generalized arc-consistency removes those values which assign l'_i $(\neq l_j)$ to false. Enforcing generalized arc-consistency on the non-binary constraint involving l'_i eliminates the truth value that assigns false to l_j. Hence, enforcing generalized arc-consistency on the non-binary encoding eliminates all contradictory values.

2. Unit propagation generates an empty clause if there is a clause, $l_1 \vee \ldots l_k$ in which unit propagation assigns each literal l_i to false. By the first result, enforcing generalized arc-consistency on the non-binary encoding eliminates each truth value which assigns l_i to true. Consider the non-binary associated with this clause. Enforcing generalized arc-consistency on this constraint causes a domain wipeout. To show that the proof does not reverse even if we are in a polynomial subclass of SAT, consider a 2-SAT problem with all possible clauses in two variables: e.g. $x_1 \vee x_2$, $\neg x_1 \vee x_2$, $x_1 \vee \neg x_2$, $\neg x_1 \vee \neg x_2$. Enforcing (generalized) arc-consistency shows that this problem is insoluble, whilst unit propagation does nothing.

With equivalent branching heuristics, DP explores the same size search tree as nFC0, the weakest non-binary version of the forward checking algorithm. DP is, however, dominated by nFC1 (the next stronger non-binary version of forward checking) and thus an algorithm that maintains generalized arc-consistency at each node.

Theorem 10. *Given equivalent branching heuristics, DP explores the same number of branches as nFC0 applied to the non-binary encoding.*

Proof. We show how to take the proof tree explored by DP and map it onto a search tree for nFC0 with the same number of branches. The proof proceeds by induction on the number of propositional variables. In the step case, consider the first variable branched upon by DP. The proof divides into two cases. Either this is a branching point (and we try both possible truth values). Or this is not a branching point (and unit propagation makes this assignment). In the first case, we can branch in the same way in nFC0. In the second case, forward checking in nFC0 will have reduced the domain of this variable to a singleton, and we can also branch in the same way in nFC0. We now have a subproblem with one fewer variable, and appeal to the induction hypothesis. The proof reverses in a straightforward manner.

Theorem 11. *Given equivalent branching heuristics, nFC1 applied to the non-binary encoding strictly dominates DP.*

Proof. Trivially nFC1 dominates nFC0. To show strictness, consider a 3-SAT problem with all possible clauses in 3 variables: $x_1 \vee x_2 \vee x_3$, $\neg x_1 \vee x_2 \vee x_3$, $x_1 \vee \neg x_2 \vee x_3$, $\neg x_1 \vee \neg x_2 \vee x_3$, $x_1 \vee x_2 \vee \neg x_3$, $\neg x_1 \vee x_2 \vee \neg x_3$, $x_1 \vee \neg x_2 \vee \neg x_3$, $\neg x_1 \vee \neg x_2 \vee \neg x_3$. DP takes 4 branches to prove this problem is unsatisfiable whatever branching heuristic is used. nFC1 by comparison takes just 2 branches. Suppose we branch on x_1. The binary projection of the non-binary constraints on x_1, x_2 and x_3 onto x_1 and x_2 is the empty (unsatisfiable) constraint. Hence, forward checking causes a domain wipeout.

5 Encoding CSPs into SAT

We now consider mappings in the reverse direction. There are two common ways to encode a (binary) CSP as a SAT problem.

Direct encoding: We associate a propositional variable, x_{ij} with each value j that can be assigned to the CSP variable X_i. We have clauses that ensures each CSP variable is given a value: for each i, $x_{i1} \vee \ldots x_{im}$. We optionally have clauses that ensure each variable takes no more than one values: for each i, j, k with $j \neq k$, $\neg x_{ij} \vee \neg x_{ik}$. Finally, we have (binary) clauses that rule out any (binary) nogoods. For example, if $X_1 = 2$ and $X_3 = 1$ is not allowed then we have the clause $\neg x_{12} \vee \neg x_{31}$.

Log encoding: We have $n\lceil \log_2(m) \rceil$ propositional variables. The propositional variable x_{ij} is set iff the CSP variable X_i is assigned a value in which the j-th bit is set. We have a clause for each (binary) nogood. For example, if $X_1 = 2$ and

$X_3 = 1$ is not allowed, and each CSP variable has the domain $\{0, 1, 2, 3\}$ then we have the clause $x_{10} \vee \neg x_{11} \vee x_{20} \vee \neg x_{21}$ (which is logically equivalent to $(\neg x_{10} \wedge x_{11}) \rightarrow \neg(\neg x_{20} \wedge x_{21})$). Note that we do not need clauses to ensure that each CSP variable is given a value, nor to ensure that each CSP variable is given only one value (any complete assignment for the propositional variables corresponds to an assignment of a single value to each CSP variable). If $\lceil \log_2(m) \rceil > \log_2(m)$ then we also have clauses that rule out (spurious) values at the top of each domain. For example, if variable X_i has only 3 values, then we have a clause $\neg x_{30} \vee \neg x_{31}$ which prohibits us assigning a fourth value to X_3.

5.1 Direct encoding

We first prove that enforcing arc-consistency on the original problem does more work than unit propagation on the direct encoding.

Theorem 12.

1. *If unit propagation commits to particular truth assignments on the direct encoding, then enforcing arc-consistency on the original problem eliminates all contradictory values.*
2. *If unit propagation generates the empty clause in the direct encoding then enforcing arc-consistency on the original problem causes a domain wipeout (but the reverse does not necessarily hold).*

Proof. 1. Suppose unit propagation makes a sequence of assignments: $l_1, l_2, \ldots l_j$. The proof uses induction on j. Consider the final assignment l. Unit propagation makes this assignment because l occurs in a clause in which all the other literals have been assigned to false. The proof divides into three cases. If the clause is of the form $x_{i1} \vee \ldots x_{im}$ then, by the induction hypothesis, enforcing arc-consistency eliminates from the domain of X_i all but the value assigned by l. Hence all contradictory values have been eliminated for X_i. If the clause is of the form $\neg x_{ij} \vee \neg x_{pq}$ and (without loss of generality) $l = \neg x_{ij}$ then, by the induction hypothesis, enforcing arc-consistency eliminates the value q from the domain of X_p. Hence, enforcing arc-consistency on the constraint associated with the clause $\neg x_{ij} \vee \neg x_{pq}$ eliminates j from the domain of X_i. Hence all contradictory values have been eliminated for X_i. Finally, if the clause is of the form $\neg x_{ij} \vee \neg x_{ik}$ where $j \neq k$ and (without loss of generality) $l = \neg x_{ij}$ then X_i has been assigned the value k (and so cannot be assigned the contradictory value j).

2. Unit propagation generates an empty clause if there is a clause, $l_1 \vee \ldots l_k$ in the direct encoding in which unit propagation assigns each literal l_i to false. The proof divides into three cases. If the clause is of the form $x_{i1} \vee \ldots x_{im}$ then, by the first result, enforcing arc-consistency on the direct encoding eliminates all contradictory values. Hence X_i has a domain wipeout. The other two cases are similar. To show that the reverse may not hold, consider a CSP in two variables and two values in which there is a binary constraint ruling out every possible assignment. The direct encoding of this problem does not contain any unit clauses so unit propagation does nothing. However, enforcing arc-consistency causes a domain wipeout.

With equivalent branching heuristics, DP applied to the direct encoding explores the same size search tree as the forward checking algorithm FC applied to the original problem. DP is, however, dominated by MAC. Given equivalent branching heuristics, DP applied to the direct encoding also explores the same size search tree as the nFC0 algorithm applied to a non-binary problem. DP is again dominated by nFC1.

Theorem 13. *Given equivalent branching heuristics, DP applied to the direct encoding explores the same number of branches as FC applied to the original problem.*

Proof. We show how to take the proof tree explored by DP and map it onto a search tree for FC with the same number of branches. The proof proceeds by induction on the number of propositional variables. In the step case, consider the first variable branched upon by DP. The proof divides into two cases. Either this is a branching point (and we try both possible truth values). Or this is not a branching point (and unit propagation makes this assignment). In the first case, we can branch in the same way in FC. In the second case, forward checking in FC will have reduced the domain of this variable to a singleton, and we can also branch in the same way in FC. We now have a subproblem with one fewer variable, and appeal to the induction hypothesis. The proof reverses in a straightforward manner.

Theorem 14. *Given equivalent branching heuristics, MAC applied to the original problem strictly dominates DP applied to the direct encoding.*

Proof. MAC trivially dominates DP applied to the direct encoding since MAC dominates FC which itself dominates DP applied to the direct encoding. To show strictness, consider again the CSP in two variables and two values in which each possible assignment is ruled out. MAC solves this without search whilst DP takes two branches on the direct encoding.

5.2 Log encoding

We first prove that unit propagation on the log encoding is less effective than unit propagation on the direct encoding. As enforcing arc-consistency on the original problem is more effective than unit propagation on the direct encoding, it follows by transitivity that enforcing arc-consistency on the original problem is more effective than unit propagation on the log encoding.

Theorem 15.

1. *If unit propagation commits to particular truth assignments on the log encoding, then unit propagation commits to the same truth assignments on the direct encoding.*
2. *If unit propagation generates the empty clause in the log encoding then unit propagation generates the empty clause in the direct encoding then (but the reverse does not necessarily hold).*

Proof. 1. Suppose unit propagation makes a sequence of assignments in the log encoding: l_1, l_2, $\ldots l_j$. The proof uses induction on j. Consider the final assignment l. Unit propagation makes this assignment because l occurs in a clause in which all the other literals have been assigned to false. By construction, this will assign $\lceil \log_2(m) \rceil$ (i.e. all) bits associated with one CSP variable and $\lceil \log_2(m) \rceil - 1$ (i.e. all but one) bits associated with another. That is, one variable will have a value assigned, By the induction hypothesis, unit propagation will have assigned the propositional variable associated with this value to true. Hence, unit propagation on the clause associated with this nogood will set the other variable (and thus its last bit).

2. Unit propagation generates an empty clause in the log encoding if there is a clause, $l_1 \vee \ldots l_k$ in which unit propagation assigns each literal l_i to false. This means that two CSP variables are effectively assigned values which contradict the nogood associated with this clause. By the first result, enforcing arc-consistency on the direct encoding makes the same assignments. Hence unit propagation on the direct encoding also generates an empty clause. To show that the reverse may not hold, consider a CSP in two variables, the first with one value, the second with four values, all o incompatible with the first value. Then unit propagation on the direct encoding generates the empty clause, but not on the log encoding.

With equivalent branching heuristics, the forward checking algorithm FC applied to the original problem strictly dominates DP applied to the log encoding. To simplify the proof, we assume that the branching heuristic in FC enumerates values in (numerical) order. The ability of FC to assign values in any order gives it an even greater edge over DP applied to the log encoding.

Theorem 16. *Given equivalent branching heuristics, FC applied to the original problem strictly dominates DP applied to the log encoding.*

Proof. We map the search tree explored by FC onto a proof tree for DP with at least as many branches. The proof proceeds by induction on the number of CSP variables. In the step case, consider the first variable x_1 branched upon by FC. We assume FC orders the values for this variable numerically. We branch in DP on x_{i0} then x_{i1}, $\ldots x_{i\lceil \log_2(m) \rceil}$. We now have a CSP subproblem with one fewer variable, and appeal to the induction hypothesis. To show strictness, consider a CSP in two variables, both with 3 values, in which all pairs of assignments are nogood. FC will take 3 branches to show that the problem is insoluble. DP on the log encoding will take 8 branches since both bits for one variable and one bit for the second variable must be set before we generate the empty clause.

6 Related work

Bennaceur studied the literal encoding for encoding SAT problems as CSPs [Ben96]. He proved that enforcing arc-consistency on the literal encoding is equivalent to unit propagation. We re-prove this result and extend it to arc-inconsistency. Bennaceur also proved that a CSP is arc-consistent iff its literal encoding has no unit clauses, and strong path-consistent iff it has no unit or binary clauses. The direct encoding of a CSP into a SAT problem appears in [dK89]. Génisson and Jégou proved that, with suitable branching heuristics, DP is equivalent to FC applied to the direct encoding [GJ96].

Apt has also looked at propagation rules for Boolean constraints [A99]. He proves an equivalence between Boolean constraint propagation and unit propagation, and between Boolean constraint propagation and generalized arc-consistency. Our results complete the triangle, characterizing the relationship between generalized arc-consistency and unit propagation.

Frisch and Peugniez studied the performance of local search procedures like WalkSAT on encodings of non-Boolean formulae into propositional satisfiability [FP99]. The unary and binary encodings studied there are closely related to the direct and log encodings of CSPs into SAT problems studied here.

Bacchus and van Beek present a study of encodings of non-binary CSPs into binary CSPs [BvB98]. The dual and hidden variable encodings studied here can be constructed by composing the non-binary encoding of SAT problems into non-binary CSPs, with the dual and hidden variable encodings of non-binary CSPs into binary CSPs. Bacchus and van Beek's work is limited to the FC algorithm and a simple extension called FC+. Stergiou and Walsh look at the maintenance of higher levels of consistency, in particular arc-consistency within these encodings [SW99]. They prove that arc-consistency on the dual encoding is strictly stronger than arc-consistency on the hidden variable, and this itself is equivalent to generalized arc-consistency on the original non-binary CSP.

7 Conclusions

We have performed a comprehensive study of mappings between constraint satisfaction problems (CSPs) and propositional satisfiability (SAT). We analysed four different mappings of SAT problems into CSPs: the dual, hidden variable, literal and non-binary encodings. We proved that achieving arc-consistency on the dual encoding does more work than unit propagation on the original SAT problem, whilst achieving arc-consistency on the hidden variable and literal encodings does essentially the same work. We then extended these results to algorithms that maintain some level of arc-consistency during search like FC and MAC, and DP which performs unit propagation at each search node. DP strictly dominates FC applied to the dual encoding, is incomparable to MAC applied to the dual encoding, explores the same number of branches as MAC applied to the hidden variable encoding, and strictly dominates MAC applied to the literal encoding. We also analysed two different mappings of CSPs into SAT problems: the direct and log encodings. We proved that unit propagation on the direct encoding does less work than achieving arc-consistency on the original problem, but more work than unit propagation on the log encoding. DP on the direct encoding explores the same size search tree as FC applied to the original problem, but is strictly dominated by MAC. By comparison, DP on the log encoding is strictly dominated by both FC and MAC applied to the original problem.

What general lessons can be learned from this study? First, the choice of encoding can have a large impact on the level of consistency achieved. For instance, the dual encoding allows us to achieve higher levels of consistency than the literal encoding. Second, the choice of encoding also has a large impact on the branching structure of our search trees. In particular, the dual and literal encodings require us to branch using a variable ordering based upon the clauses. DP applied to the original SAT problem can therefore sometimes beat MAC applied to the dual encoding. Fourth, whilst a clearer picture of the relationship between SAT problems and CSPs is starting to emerge, there are several questions that remain unanswered. For example, how do local search methods like GSAT and Min-Conflicts compare on these different encodings?

References

[A99] K. Apt. Some Remarks on Boolean Constraint Propagation. In New Trends in Constraints, Papers from the Joint ERCIM/Compulog-Net Workshop. Springer, 1999.

[Ben96] H. Bennaceur. The satisfiability problem regarded as a constraint satisfaction problem. In W. Wahlster, editor, *Proc. of the 12th ECAI*, pages 155–159. European Conference on Artificial Intelligence, Wiley, 1996.

[BMFL99] C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On Forward Checking for Non-binary Constraint Satisfaction. In A. Brodsky and J. Jaffar, editors, *Proc. of Fifth International Conference on Principles and Practice of Constraint Programming (CP99)*. Springer, 1999.

[BvB98] F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proc. of 15th National Conference on Artificial Intelligence*, pages 311–318. AAAI Press/The MIT Press, 1998.

[dK89] J. de Kleer. A comparison of ATMS and CSP techniques. In *Proc. of the 11th IJCAI*. International Joint Conference on Artificial Intelligence, 1989.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[FP99] A.M. Frisch and T.J. Peugniez. Solving non-Boolean satisfiability problems with stochastic local search: A comparison of encodings., 1999. Unpublished manuscript, available from `http://www.cs.york.ac.uk/aig/publications.html`.

[Gas79] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Carnegie-Mellon University, 1979. PhD thesis.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[GJ96] R. Genisson and P. Jegou. Davis and Putnam were already forward checking. In W. Wahlster, editor, *Proc. of the 12th ECAI*, pages 180–184. European Conference on Artificial Intelligence, Wiley, 1996.

[MM88] R. Mohr and G. Masini. Good old discrete relaxation. In *Proc. of the 8th ECAI*, pages 651–656, European Conference on Artificial Intelligence, 1988.

[SW99] K. Stergiou and T. Walsh. Encodings of non-binary constraint satisfaction problems. In *Proc. of the 16th National Conference on AI*. American Association for Artificial Intelligence, 1999.