Reformulating propositional satisfiability as constraint satisfaction

Toby Walsh

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

Abstract. We study how propositional satisfiability (SAT) problems can be reformulated as constraint satisfaction problems (CSPs). We analyse four different mappings of SAT problems into CSPs. For each mapping, we compare theoretically the performance of systematic algorithms like FC and MAC applied to the encoding against the Davis-Putnam procedure applied to the original SAT problem. We also compare local search methods like GSAT and WalkSAT on a SAT problem against the Min-Conflicts procedure applied to its encoding. Finally, we look at the special case of local search methods applied to 2-SAT problems and encodings of 2-SAT problems. Our results provide insight into the relationship between propositional satisfiability and constraint satisfaction, as well as some of the potential benefits of reformulating problems as constraint satisfaction problems.

1 Introduction

A number of different computational problems have been solved by reformulating them as propositional satisfiability (SAT) problems. Surprisingly, even problems for higher complexity classes than SAT can be efficiently solved by reformulating them as (a sequence of) SAT problems. For example, Kautz and Selman's BLACKBOX system won the AIPS-98 planning competition by reformulating STRIPS planning problems as a sequence of SAT problems [KS98a,KS98b]. Other computational problems as diverse as quasigroup existence, hardware diagnosis and spacecraft control have been translated into SAT problems and solved efficiently. But is SAT the best choice as a target language for such reformulation?

One possible weakness of SAT is that variables have only two possible values (true or false). Constraint satisfaction, by comparison, offers a target language in which variables can take larger domains. Such domains might allow us to model problems more naturally and reason about them more efficiently. Another possible weakness of SAT is the limited number of systematic solvers available, most of which are based upon the (now elderly) Davis-Putnam procedure. Constraint satisfaction, by comparison, offers a vast array of systematic solvers (e.g. BT, FC, MAC, BJ, CBJ, FC-CBJ, MAC, MAC-CBJ, DB, ...). To explore when reformulating problems into CSPs is worthwhile, and to understand the relationship between SAT and CSPs, we are studying mappings between SAT problems and CSPs. Bennaceur has previously looked at reformulating SAT problems as CSPs

[Ben96]. However, this study was limited to a single mapping. Since the choice of mapping can have a very large impact on our ability to solve problems, it is 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, as well as of the potential benefits of reformulating problems into constraint satisfaction problems.

2 Constraint satisfaction

A constraint satisfaction problem (CSP) is a triple (X, D, C). X is a set of variables. For each $x_i \in X$, D_i is the domain of the variable. Each k-ary constraint $c \in C$ is defined over a set of variables $(x_1, \ldots x_k)$ by the subset of the cartesian product $D_1 \times \ldots D_k$ which are consistent values. A binary CSP has only binary constraints. A non-binary CSP has larger arity constraints. A solution for a CSP is an assignment of values to variables that is consistent with all constraints. Many lesser levels of consistency have been defined for binary constraint satisfaction problems (see [DB97] for references). 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. When enforcing arc-consistency, any value assigned to a variable that cannot be extended to a second variable can be removed from the variable's domain. If all values for a variable are removed, a domain wipeout occurs, and the problem is insoluble. Other stronger local consistencies have shown promise, including singleton arc-consistency. A problem is singleton arc-consistent (SAC) iff it has non-empty domains and for any assignment of a variable, the problem can be made arc-consistent. Singleton arc-consistency provides useful extra pruning compared to arc-consistency at a moderate additional computational expense [DB97].

Most of these definitions can be extended to non-binary constraints. For example, 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. Systematic 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. 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 variables, whilst the nFC1 algorithm maintains generalized arc-consistency on those constraints and constraint projections involving one uninstantiated variable [BMFL99]. Local search methods can also be used to solve CSPs. For example, the Min-Conflicts procedure (MC) repairs a complete assignment by randomly choosing a variable that is in an unsatisfied

constraint, and giving it a new value which minimizes the number of violated constraints.

3 Propositional satisfiability

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. One of the best systematic 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 (which deletes any tautologous clause) as neither are needed for completeness or soundness, nor usually for efficiency. Note that the unit propagation rule is effectively the "singleton" empty rule. That is, if we assign the complement of an unit clause, the empty rule shows that the resulting problem is unsatisfiable; we can therefore delete this assignment. Local search methods can also be used to solve SAT problems. There are two popular families of local search procedures based upon GSAT and WalkSAT. The GSAT procedure repairs a complete truth assignment by flipping the truth value of a variable that minimizes the number of unsatisfied clauses (sideways moves are allowed). The WalkSAT procedure repairs a complete truth assignment by flipping the truth value of a variable that occurs in an unsatisfied clause. The variable is either chosen at random or using a greedy heuristic based on the number of satisfied clauses.

4 Reformulating SAT problems as CSPs

There are several different ways that a SAT problem can be reformulated 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. This encoding appears in [Ben96].

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 $\langle F, F, T \rangle$.

Note that the literal encoding using 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.

5 Systematic procedures

We now compare the performance of the Davis-Putnam (DP) procedure against some popular systematic CSP algorithms like FC and MAC on these different encodings. 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.

5.1 Dual encoding

There are several difficulties in comparing DP against algorithms like FC and MAC applied to the dual encoding. 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 1. *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 (and implicitly an equivalent branching heuristic for DP) 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 1 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 that that maintained by FC? Consider, for example, all possible 2-SAT clauses in two variables. Enforcing arc-consistency on the dual encoding shows that this problem is unsatisfiable. However, as the problem does not contain any unit clauses, unit propagation does not show it is unsatisfiable. Hence enforcing arc-consistency on the dual encoding can do more work than unit propagation. This might suggest that MAC (which enforces arc-consistency

at each node) might outperform DP (which only 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 2. *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 outperforms DP in this case. 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.

5.2 Hidden variable encoding

We will 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 3. *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 in a similar way, 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 4. *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. With such a heuristic, FC explores a tree with the same number of branches as DP. Hence, using the last result, FC explores a tree with the same number of branches as MAC.

5.3 Literal encoding

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 5. *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 naed 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 (and implicitly a branching heuristic for DP) which has no more branches than the tree explored by MAC. To prove strictness, consider the example in the proof of the next theorem.

Although DP can explore a smaller search tree than MAC applied to the literal encoding, both are exponential in the worst case. However, MAC's worst case behaviour scales with a larger exponent than DP's. The problem with MAC is that the branching factor of its search is governed by the clause size. Branching propositionally (on whether a variable is true or false) can be more efficient. Indeed, we can exhibit a class of problems on which the ratio of the number of branches explored by DP compared to that explored by MAC vanishes to zero as problem size grows.

Theorem 6. *There exists a class of SAT problems in n variables on which the ratio of the number of branches explored by DP compared to that explored by MAC on the literal encoding tends to zero as $n \to \infty$, whatever branching heuristics are used.*

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. However, MAC takes $k!$ branches whatever variable and value ordering we use. As $k \to \infty$, the ratio of the number of branches explored by DP to that explored by MAC is $O(2^k/k!)$. By Stirling's approximation, this tends to zero.

5.4 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 DP on the SAT problem with algorithms that enforce (some level of) generalized arc-consistency on the non-binary encoding. 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 7. *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 8. *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.

6 Local search methods

It is more difficult to compare theoretically the performance of local search procedures like GSAT on a SAT problem with methods like Min-Conflicts (MC) applied to an encoding of this problem. For example, whilst the assignments for the dual variables will often not be consistent with each other, the only values allowed are those that satisfy the clauses. MC applied to the dual encoding cannot therefore be in a part of the search space in which clauses are not satisfied. By comparison, GSAT's search is almost exclusively over states in which some of the clauses are not satisfied. A similar observation applies to the literal encoding.

It is easier to make comparisons with the hidden variable and non-binary encodings. With both these encodings, MC will have a complete assignment to the (propositional) variables which, as in GSAT and WalkSAT, may not satisfy all the clauses. One remaining difficulty is that most of the local search methods have a stochastic component. Our comparison of search methods is therefore of the form: if method A moves from state X to state Y, is there a non-zero probability that method B can move between corresponding states in its search space? If this is the case, we say that method B can simulate method A. This means that, in theory at least, method B can follow the same trajectory through the search space as method A. It does not mean that method B is necessarily any more efficient than method A (or vice versa) as the probability that method B can follow method A's trajectory to a solution could be very small. However, if method A cannot simulate method B and vice versa, it is likely that there will be significant differences in their performance.

Theorem 9. *MC on the non-binary encoding can neither simulate GSAT on the original SAT problem nor vice versa.*

Proof. Suppose we cannot increase the number of satisfied clauses by flipping a single variable (this is a very common situation in GSAT's search). Then it is possible that GSAT will pick a variable to flip that only occurs in satisfied clauses. MC, on the other hand, must pick a variable in one of the unsatisfied clauses. Hence, MC cannot simulate GSAT. Suppose MC picks a variable in an unsatisfied clause, and flipping it decreases the number of satisfied clauses (again this is a very common situation in MC's search). GSAT, on the other hand, cannot pick this variable. Hence, GSAT cannot simulate MC.

Theorem 10. *MC on the non-binary encoding can simulate WalkSAT on the original SAT problem (and vice versa).*

Proof. Suppose WalkSAT picks a variable in an unsatisfied clause and flips it. MC has a non-zero probability of picking the same clause and variable. Although

MC is limited to give this variable a new value which minimizes the number of violated clauses, variables only have two values (true or false) so we flip it the same way as WalkSAT. Hence MC can simulate WalkSAT. To show the reverse, suppose MC picks a variable in an unsatisfied clause and flips it. Then WalkSAT has a non-zero probability of picking the same clause and variable. Hence WalkSAT can simulate MC.

In the hidden variable encoding, we focus on the variable assignments given to the propositional variables (those given to the dual variables must, by construction, satisfy all the clauses). We therefore ignore dual variables flipped by MC and consider instead only those situations where MC flips one of the propositional variables. Note that since each constraint in the hidden variable encoding is between a propositional and a dual variable, every unsatisfied constraint in the hidden variable encoding contains a propositional variable which MC might chose to flip.

Theorem 11. *MC on the hidden variable encoding can neither simulate GSAT on the original SAT problem nor vice versa.*

Proof. Suppose we have two disjoint sets of clauses, one of which is satisfied and the other not. GSAT can pick a variable to flip that occurs in the satisfied set. MC applied to the hidden variable encoding, on the other hand, must pick a variable in the unsatisfied set. Hence, MC applied the hidden variable encoding cannot simulate GSAT. To show that the reverse also does not hold, observe that MC applied to the hidden variable encoding may flip a propositional variable that decreases the number of satisfied clauses. However, GSAT cannot flip such a variable. Hence, GSAT cannot simulate MC.

Theorem 12. *MC on the hidden variable encoding can simulate WalkSAT on the original SAT problem (but not vice versa).*

Proof. Suppose WalkSAT picks a variable in an unsatisfied clause and flips it. MC has a non-zero probability of picking the same propositional variable as the constraint between it and the dual variable associated with the unsatisfied clause cannot be satisfied. As variables only have two values (true or false), we flip the propositional variable in the same way as WalkSAT. Hence MC can simulate WalkSAT. To show that the reverse may not hold, suppose we have two disjoint sets of clauses, and a truth assignment which satisfies only one of the sets. Also suppose that one of the dual variables associated with a clause in the satisfied set has an assignment which contradicts the satisfying propositional assignment. Now MC may flip one of the propositional variables associated with this clause. WalkSAT, however, cannot flip this variable as it is not in an unsatisfied clause. Hence WalkSAT cannot simulate MC. Note that we could modify MC so that dual variables are always set according to the values given to the propositional variables. WalkSAT can simulate this modified MC algorithm (and vice versa).

6.1 2-SAT

For the tractable case of 2-SAT (in which each clause has 2 literals), we can give more precise results comparing the performance of some simple local search methods on the original SAT problem and on its encoding. We consider Papadimitriou's random walk (RW) algorithm which starts from a random truth assignment, picks at random an unsatisfied clause and a variable within this clause, and flips its truth assingment [Pap91]. A straight forward generalization to CSPs is to start from a random assignment of values to variables, pick at random a constraint that is violated and a variable within this constraint, and randomly change this variable's assignment. Papadimitriou has proved that RW applied to a satisfiable 2-SAT problem can be expected to find a model in quadratic time.

Theorem 13. *RW is expected to take at most n^2 flips to find a satisfying assignment for a satisfiable 2-SAT problem in n variables [Pap91].*

Proof. The problem reduces to an one-dimensional random walk with a reflecting and an absorbing barrier (or "gambler's ruin against the sheriff"). We give the details here as a similar proof construction is used in the next proof. Consider a satisfying assignment S for the 2-SAT problem. Let $N(i)$ be the expected number of flips to find a satisfying assignment given that we start i flips away from S. Now $N(0) = 0$. For $i > 0$, we chose one of the literals in an unsatisfied clause. At least one of these literals must be true in S. Hence, we have at least a half chance of moving closer to S. Thus, $N(i) \leq 1/2(N(i-1) + N(i+1)) + 1$ for $0 < i < n$. And for $i = n$, $N(n) \leq N(n-1) + 1$ since we must move nearer to S. Consider the recurrence relation $M(0) = 0$, $M(i) = 1/2(M(i-1) + M(i+1)) + 1$ for $0 < i < n$. and $M(n) = M(n-1) + 1$. We have $M(i) \geq N(i)$ for all i. And a solution for $M(i)$ is $M(i) = 2in - i^2$. The worst case is $i = n$, when $M(n) = n^2$. Hence $N(i) \leq n^2$.

It follows from this result that the probability that RW finds a satisfying assignment after $2n^2$ flips is at least $1/2$. This appeals to the lemma that $prob(x \geq k.\langle x \rangle) \leq 1/k$ for any $k > 0$ where $\langle x \rangle$ is the expected value of x. The (generalized) RW algorithm applied to the literal encoding of a 2-SAT problem also runs in expected quadratic time.

Theorem 14. *RW is expected to take at most l^2 flips to find a satisfying assignment when applied to the literal encoding of a satisfiable 2-SAT problem in l clauses.*

Proof. The problem again reduces to an one-dimensional random walk with a reflecting and an absorbing barrier. However, there are now l variables (one for each clause), each with two possible values. Again, the probability of flipping one of these variables and moving nearer to a (distinguished) satisfying assignment is at least $1/2$. Hence, the expected number of flips is at most l^2.

Note that RW on the literal encoding is expected to take (at most) l^2 flips whilst RW on the original 2-SAT problem is expected to take (at most) n^2 flips. Performance is likely to be similar as l and n for satisfiable 2-SAT problems tend to be closely related. For instance, the phase transition for random 2-SAT problems occurs around $l/n = 1$ [CR92,Goe92]. That is, in the limit random 2-SAT problems are almost always satisfiable for $l/n < 1$, and almost always unsatisfiable for $l/n > 1$.

There is little point in considering the non-binary encoding of the 2-SAT problem as this reduces to a binary CSP which is isomorphic in structure to the original 2-SAT problem. Hence RW will perform in an identical manner on this encoding as on the original 2-SAT problem. Analysing the behaviour of RW on the dual and hidden variable encoding of 2-SAT problems is more problematic as the dual variables have domains of size 3, and correspond to the assignment of values to pairs of variables.

7 Related work

Bennaceur studied the literal encoding for reformulating SAT problems as CSPs [Ben96]. He proved that enforcing arc-consistency on the literal encoding is equivalent to unit propagation. 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. Bacchus and van Beek present one of the first detailed studies 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 study 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 origianl (non-binary) CSP. More recently, van Beek and Chen have shown that reformulating planning problems as constraint satisfaction problems (CSPs) using their CPlan system is highly competitive [vBC99].

8 Conclusions

We have performed a comprehensive study of reformulations of propositional satisfiability (SAT) problems as constraint satisfaction problems (CSPs). We analysed four different mappings of SAT problems into CSPs: the dual, hidden variable, literal and non-binary encodings. We compared theoretically the performance of systematic search algorithms like FC and MAC applied to these encodings against the Davis-Putnam procedure. Given equivalent branching heuristics, 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 compared local search methods like GSAT and WalkSAT against the Min-Conflicts procedure applied to these encodings. On the hidden variable and non-binary encodings, we showed that the WalkSAT and Min-Conflicts procedures could follow similar trajectories through their search space. However, this was not necessarily the case for the GSAT and Min-Conflicts procedures. We also proved that a simple random walk procedure is expected to take quadratic time on the literal encoding of a 2-SAT problem, similar to the performance of the procedure applied directly to the 2-SAT problem.

What general lessons can be learned from this study? First, the choice of encoding can have a large impact on search. For example, despite the higher level of consistency achieved by enforcing arc-consistency in the dual encoding compared to unit propagation on the original SAT problem, DP applied to the original SAT problem can sometimes beat MAC applied to the dual encoding because DP allows more flexible branching heuristics. Second, comparing theoretically the performance of local search procedures on these mappings is problematic. For instance, the state space explored by Min-Conflicts applied to the dual encoding is completely different to that explored by GSAT. Empirical studies may therefore be the only way we can make informative comparisons between such local search procedures. Third, whilst a clearer picture of the relationship between SAT problems and CSPs is starting to emerge, there are several questions which remain unanswered. For example, how do non-chronological backtracking procedures like backjumping [Dec90] and dynamic backtracking [Gin93] compare on these different encodings? What is the practical impact of these theoretical results? And finally, do mappings in the opposite direction (i.e. of CSPs into SAT) support similar conclusions?

Acknowledgements

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.

References

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

[BMFL99] C. Bessiere, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Proceedings of IJCAI-99 Workshop on Non-binary constraints*. International Joint Conference on Artificial Intelligence, 1999.

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

[CR92] V. Chvatal and B. Reed. Mick gets some (the odds are on his side). In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 620–627. IEEE, 1992.

[DB97] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of the 15th IJCAI*, pages 412–417. International Joint Conference on Artificial Intelligence, 1997.

[Dec90] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decompositio. *Artificial Intelligence*, 41(3):273–312, 1990.

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

[Gin93] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Goe92] A. Goerdt. A theshold for unsatisfiability. In I. Havel and V. Koubek, editors, *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 264–274. Springer Verlag, 1992.

[KS98a] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search*, 1998. Held in conjunction with AIPS-98, Pittsburgh, PA, 1998.

[KS98b] H. Kautz and B. Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings of AIPS-98, Pittsburgh, PA*, 1998.

[Pap91] C.H. Papadimitriou. On selecting a satisfying truth assigment. In *Proceedings of the Conference on the Foundations of Computer Science*, pages 163–169, 1991.

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

[vBC99] P. van Beek and X. Chen. Cplan: a constraint programming approach to planning. In *Proceedings of 16th National Conference on Artificial Intelligence*. AAAI Press/The MIT Press, 1999.