On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems

Fahiem Bacchus
Department of Computer Science
University Of Waterloo
Waterloo, Ontario, Canada, N2L 3G1
fbacchus@logos.uwaterloo.ca

Peter van Beek
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada, T6G 2H1
vanbeek@cs.ualberta.ca

Abstract

It is well known that any non-binary discrete constraint satisfaction problem (CSP) can be translated into an equivalent binary CSP. Two translations are known: the dual graph translation and the hidden variable translation. However, there has been little theoretical or experimental work on how well backtracking algorithms perform on these binary representations in comparison to their performance on the corresponding non-binary CSP. We present both theoretical and empirical results to help understand the tradeoffs involved. In particular, we show that translating a non-binary CSP into a binary representation can be a viable solution technique in certain circumstances. The ultimate aim of this research is to give guidance for when one should consider translating between non-binary and binary representations. Our results supply some initial answers to this question.

Introduction

The lion's share of work on constraint satisfaction problems (CSPs) has restricted its attention to binary CSPs, where all constraints are between two variables. This work has generated a great deal of knowledge about the theory and practice of solving CSPs. Unfortunately, it is not always straightforward to generalize this knowledge to non-binary CSPs. The well known fact that any non-binary discrete CSP can be converted into an equivalent binary CSP is usually used as a justification for restricting attention to binary CSPs. Implicitly, the assumption has been that when faced with a non-binary CSP we can simply convert it into a binary CSP, and then apply the best techniques for solving the binary equivalent.

The field has not completely ignored the issue of non-binary CSPs, however, as there has been work in both the constraint programming and the traditional CSP communities that addresses direct solution techniques for non-binary CSPs. In particular, two of the most successful techniques for solving binary problems, backtracking combined with forward checking and backtracking combined with arc consistency, have been generalized to the non-binary case (Mac77; VH89).

Hence, there are at least two options when it comes to dealing with non-binary CSPs: apply one of the standard translations to convert it to a binary CSP and then solve it using binary CSP techniques, or apply one of the direct solution techniques for non-binary CSPs. A potential advantage of translating to the binary is that much more is known about solving binary CSPs: more useful heuristics are known, more polynomial-time special cases have been identified, and more algorithms are known[1]. On the other hand it is unknown whether or not these techniques are useful when applied to the CSPs that arise from translating non-binary CSPs. Surprisingly little work has been done on examining the effectiveness of the translation technique, or on comparing these two options. The work presented here addresses this problem.

There are at least two good reasons for looking more carefully at the issue of translating non-binary CSPs into binary CSPs. First, non-binary CSPs appear quite frequently when modeling real problems. In this case the issue is how to solve these problems most efficiently, and as we show there are certain cases where translating to the binary produces significant performance gains and cases where it produces a degradation in performance. The second reason is that, as noted above, a common justification for focusing solely on binary CSPs is the fact that non-binary CSPs can be translated into binary CSPs. Hence, it is important to study the properties of these translations so as to better understand the legitimacy of focusing on the binary case.

Two general methods are known for converting non-binary CSPs to binary CSPs: the dual graph method and the hidden variable method. The dual graph representation comes from the relational database community and was introduced to the CSP community by Dechter and Pearl (DP89). Earlier, Freuder (Fre78) had used an incremental version of the method in an algorithm for finding all solutions to a CSP. The hidden variable translation has an even longer history. Rossi et al. (RPD89) credit Peirce (Pei33) with first showing that binary relations have the same expressive power as non-binary relations. Peirce's method for representing non-binary relations with a collection of binary relations forms the foundation of the hidden variable method. Dechter (Dec90) shows how to represent any non-binary relation with binary relations using hidden variables

[1] For example, algorithms such as minimal forward checking (DM94) and lazy arc consistency (SRGV96) are currently only applicable to binary CSPs.

that have bounded domain sizes. Rossi et al. (RPD89) discusses both the hidden and the dual conversion methods and examined whether a non-binary CSP and its binary representation are equivalent under various definitions of equivalence.

We present both empirical and theoretical results that help us understand the properties of the dual graph and hidden variable representations. Our results compare the number of nodes visited and the number of constraint checks executed by the forward checking algorithm (FC), when applied to a non-binary CSP and to its binary equivalents. The results indicate that for most problems the non-binary representation is the most efficient representation. However, there is a class of problems, when the constraints are tight or restrictive, for which the binary translations can be more efficient by orders of magnitude. Also, we examine a specialized algorithm for the hidden representation, which we call FC$^+$. This algorithm has the advantage that it provably never performs more than a polynomial factor worse than FC on the non-binary representation, and it can often perform exponentially better.

The ultimate aim of this research is to provide guidance for efficiently solving a CSP representation of a problem. Given a problem, there is always the question of how to formulate it as a CSP. Ideally, the problem is modeled in the most natural way and the machine automatically solves it in the most efficient way. When presented with a particular instance of a non-binary CSP, should the machine convert it to a binary representation before solving? To answer this question we need a better understanding of the tradeoffs involved in such a conversion.

Background

We first define constraint satisfaction problems (CSP) and then briefly review backtrack search, the dual graph translation, and the hidden variable translation.

Definition 1 [CSPs] A constraint satisfaction problem consists of a finite set of *variables*, $\mathcal{V} = \{V_1, \ldots, V_n\}$; for each variable $X \in \mathcal{V}$ a finite domain of *values*, $Dom(X) = \{x_1, \ldots, x_k\}$; and a finite collection of *constraints*, $\mathcal{C} = \{C_1,\ldots,C_m\}$. Each constraint $C \in \mathcal{C}$ is a constraint over some set of variables $Vars(C)$. The size of this set is known as the arity of the constraint. Non-binary CSPs are CSPs that contain constraints with arity greater than 2. Every constraint C can be viewed as being a subset of the product of the domains of the variables in $Vars(C)$ (i.e., C is the set of tuples that satisfy the constraint).

We say that a set of assignments to variables $\mathcal{A} = \{X_1 \leftarrow x_1,\ldots, X_\ell \leftarrow x_\ell\}$ is *consistent* with a constraint C if (i) it assigns a value to all of the variables of C (i.e., $Vars(C) \subseteq \{X_1, \ldots, X_\ell\}$) and (ii) the tuple of values assigned by $\mathcal{A}$ to the variables of C is a member of C (i.e., the assignment satisfies the constraint). A *solution* to a CSP is a set of assignments to all n variables $\{V_1 \leftarrow v_1, \ldots, V_n \leftarrow v_n\}$ that is consistent with each of the m constraints. The notation $\|C\|$ is used to denote the size of a set C.

Example 1 Consider the 3-SAT problem, $(X_1 \vee X_2 \vee X_6) \wedge (\neg X_1 \vee X_3 \vee X_4) \wedge (\neg X_4 \vee \neg X_5 \vee X_6) \wedge (X_2 \vee X_5 \vee \neg X_6)$.

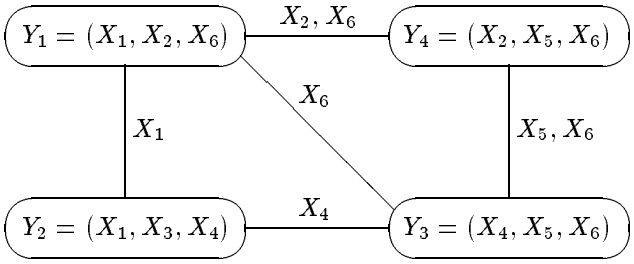

Figure 1: Binary CSP resulting from the dual method

In the non-binary CSP representation of the 3-SAT problem there is a variable for each boolean variable $X_1, \ldots, X_6$, each variable has the domain of values $\{0, 1\}$, and there is a 3-ary constraint for each clause in the formula to ensure that each clause evaluates to 1. For example, the constraint on the first clause contains the tuples, $R_{\{X_1,X_2,X_6\}} = \{(0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$, where the tuple $(0,0,0)$ does not appear in the constraint since the assignment $X_1 \leftarrow 0$, $X_2 \leftarrow 0$, $X_6 \leftarrow 0$ does not satisfy the clause.

CSPs are often solved using a backtracking algorithm. Here we restrict our attention to the widely used forward checking backtracking algorithm (FC) (HE80; McG79), which has been generalized to handle non-binary CSPs. Following Van Hentenryck (VH89) we say that a k-ary constraint, $k \geq 2$, is *forward checkable* if $k-1$ of its variables have been instantiated and the remaining variable is uninstantiated. At a node in the search tree, the new variable assigned at that node causes some (possibly empty) set of constraints to become forward checkable. For each newly forward checkable constraint, FC forward checks the remaining unassigned variable. For each unpruned value of that unassigned variable FC checks whether or not that value along with the node's assignments is consistent with the constraint, pruning those values that are inconsistent. If this process causes the unassigned variable to have all of its domain values pruned, FC backtracks.

We now present the dual graph and hidden variable translations for converting non-binary CSPs into binary ones. In the dual translation, the constraints of the original problem become variables in the new representation. We refer to these variables, which represent the constraints, as *c-variables* and the original variables simply as variables. The domain of each c-variable is exactly the set of tuples that satisfy the original constraint and there is a binary constraint between two c-variables iff the original constraints share some variables. The binary constraints prohibit pairs of tuples in which shared variables receive different values.

Example 2 In the dual graph representation of the CSP in Example 1, there are four variables $Y_1, \ldots, Y_4$, one for each 3-ary constraint (or clause) in the original problem (see Figure 1). For example, the variable Y_1 corresponds to the non-binary constraint $R_{\{X_1,X_2,X_6\}}$ and the domain of Y_1 contains the tuples (0, 0, 1), ..., (1, 1, 1). The binary constraints enforce that the ordinary variables appearing in more than

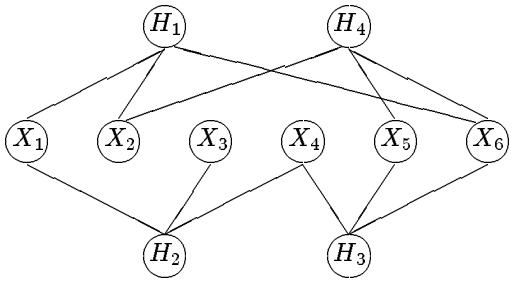

Figure 2: Binary CSP resulting from the hidden method

one c-variable have the same value.

In the hidden representation, the set of variables includes all of the variables of the original problem (with no changes to their domains) plus a new set of "hidden" or *h-variables* variables. For each constraint C_i in the original problem we add an h-variable H_i. The domain of H_i consists of a unique identifier for every tuple in C_i. The new representation contains only binary constraints, and these are constructed as follows. For every h-variable H_i we impose a binary constraint between H_i and each of the variables in $Vars(C_i)$. Say that H_i and X_k are thus constrained. Every value of H_i corresponds to a tuple of values for the variables in $Vars(C_i)$ and thus defines a unique value for X_k. Hence the binary constraint between H_i and X_k consists of a unique value for X_k for every value of H_i. (Note that the constraint is not functional in the other direction as a value for X_k may be compatible with many values of H_i.)

Example 3 In the hidden variable representation of the CSP in Example 1 there are ten variables: the six original variables $X_1, \ldots, X_6$ and four hidden variables, one for each constraint in the original problem (see Figure 2). For example, the constraint $R_{\{X_1,X_2,X_6\}}$ has a corresponding h-variable, H_1, whose domain can be the set $\{1, 2, \ldots, 7\}$ (a unique identifier for each of the seven tuples in the constraint). We can define a correspondence between the values of H_1 and the tuples in $R_{\{X_1,X_2,X_6\}}$ as follows:

$1 \mapsto (0,0,1), 2 \mapsto (0,1,0), 3 \mapsto (0,1,1), 4 \mapsto (1,0,0),$
$5 \mapsto (1,0,1), 6 \mapsto (1,1,0), 7 \mapsto (1,1,1).$

We then impose a constraint between the pairs of variables $\{X_1, H_1\}$, $\{X_2, H_1\}$, and $\{X_6, H_1\}$, giving the binary constraints,

$C_{\{X_1,H_1\}} = \{(0,1),(0,2),(0,3),(1,4),(1,5),(1,6),(1,7)\},$
$C_{\{X_2,H_1\}} = \{(0,1),(1,2),(1,3),(0,4),(0,5),(1,6),(1,7)\},$
$C_{\{X_6,H_1\}} = \{(1,1),(0,2),(1,3),(0,4),(1,5),(0,6),(1,7)\}.$

For example, for $C_{\{X_1,H_1\}}$, the value 3 for H_1 corresponds to the tuple $(0,1,1)$ in which $X_1 = 0$. Hence, $H_1 = 3$ is only compatible with $X_1 = 0$.

Theoretical Comparisons

We first consider the space requirements of the dual and hidden representations. Most CSP algorithms can deal with constraints represented either intensionally, as a function, or extensionally, as a list of compatible tuples or as a boolean array that stores for each possible assignment to the constrained variables a flag indicating whether or not that assignment is compatible. However, the more effective backtracking algorithms, such as FC, use storage proportional to the size of the domains of the variables to keep track of which domain values have been pruned during search. This means that when the dual and hidden representations "make the constraints into variables" they require extra storage of size equal to the total number of tuples in all of the constraints: $\sum_{i=1}^{m} \|C_i\|$.

As well, the dual and hidden representations require additional space to store their binary constraints. Fortunately, the constraints in both can be represented as simple functions, and thus impose only a small additional space requirement. For example, in the hidden representation, to check if $H_i \leftarrow h$ is compatible with $X_k \leftarrow x$, we simply find the tuple of assignments corresponding to h and then check to see if this tuple assigns x to X_k. The pair of assignments are compatible if and only if this is the case. If the original constraints are represented extensionally, as a list of satisfying tuples, then this operation can be done in constant time. However, if the original constraints are intensionally represented we have a space-time tradeoff. We can convert the intensional representation to an extensional one, and pay the space required to store the list of satisfying tuples. Or we can dynamically compute the tuple corresponding to h by iterating over the possible assignments of the constrained variables to find the h'th satisfying tuple. Since this has to be done every time we check a constraint, it will usually not be practical. Hence, we may also assume that we have an extensional representation of the original constraints. Of course, this will be an *additional* space requirement only when the original constraints are represented intensionally.

We now show analytic bounds on the differences in the number of nodes visited and consistency checks performed by the FC algorithm applied to a non-binary CSP and to the corresponding binary CSPs. For ease of exposition, we assume that FC runs until all solutions have been found or it is proven that no solution exists. One issue to address is that of properly accounting for checking k-ary constraints. Clearly, checking if a set of assignments $\{X_1 \leftarrow x_1, \ldots, X_k \leftarrow x_k\}$ satisfies an k-ary constraint must take at least k operations. This is true whether or not the constraint is represented intensionally as a function (where the function must consider all k values), or extensionally as an boolean array (where we will require k operations to index into an k-dimension array). Hence, we will charge k constraint checks for every check of an arity k constraint. To be consistent with this measure we charge the check of a binary constraint as 2 constraint checks (such a check requires at least 2 operations). Note that our counts for binary constraint checks are thus twice what is traditionally counted as a constraint check. This way of accounting for constraint checks allows us to properly compare the fundamental operations performed when solving the non-binary CSP and the corresponding binary CSPs.

Dual Graph Representation. The relative cost of FC on the non-binary CSP and the dual CSP depends on the cardinalities of the constraints and on the structure of the underlying constraint graph.

Example 4 shows that when the constraints have many satisfying tuples, the dual can be exponentially worse in terms of number of consistency checks. However, as Example 5 shows, when the constraints have few satisfying tuples, the dual can also be exponentially better. In such cases, FC on the original may have to visit a large number of assignments prior to being able to check a constraint, whereas FC on the dual only examines assignments that are known to be consistent with some constraint.

Example 4 Consider the non-binary CSP with n Boolean variables $X_1, \ldots, X_n$ and n constraints given by $\{X_1, \neg X_1 \vee X_2, \neg X_1 \vee \neg X_2 \vee X_3, \cdots, \neg X_1 \vee \cdots \vee \neg X_{n-1} \vee X_n\}$. In this CSP, nodes in the backtrack tree for the dual representation have exponentially many children and there is exactly one solution node. FC applied to this problem would visit n nodes and perform $O(n^2)$ consistency checks, whereas FC applied to the dual of this problem would visit n nodes and perform $O(2^n)$ consistency checks.

Example 5 Consider the non-binary CSP with n Boolean variables $X_1, \ldots, X_n$ and n constraints given by $\{X_1 \wedge \cdots \wedge X_{n-1}, X_1 \wedge \cdots \wedge X_{n-2} \wedge X_n, \cdots, X_2 \wedge \cdots \wedge X_n\}$; i.e., all n possible ways of forming a conjunction of $n-1$ variables. In this CSP, nodes in the backtrack tree for the dual have a single child and there is exactly one solution node. FC applied to this problem would visit 2^{n-1} nodes and perform $O(n2^n)$ consistency checks, whereas FC applied to the dual of this problem would visit n nodes and perform $O(n^2)$ consistency checks.

When visiting a node in the search tree for the original CSP, FC ensures that there exists a *single* extension of the current set of assignments that satisfies all of the forward checkable constraints. With FC on the dual, we have a different guarantee, that for every remaining constraint there exists *some* extension of the current set of assignments (to the original variables) that is consistent with it; there need not be a single extension satisfying multiple constraints. This means that FC on the original CSP checks for a stronger condition on a smaller number of constraints, while FC on the dual checks for a weaker condition on a larger set of constraints. An analysis of examples suggests that as the number of constraints m grows, it becomes increasingly likely that FC on the original CSP will detect deadends in the search earlier than FC on the dual, and that as m decreases, the converse situation becomes increasingly likely. How this works out in practice is an experimental question which we examine in the next section.

Hidden Variable Representation. We now turn our attention to the hidden representation. First, we demonstrate that, as with the dual, FC on the original CSP and FC on its hidden representation are incomparable: the algorithm can perform exponentially better or worse depending on the particular problem. Examples 6 and 7 illustrate this point.

Example 6 Consider a CSP containing the constraint C over some set of variables $\{X_1, X_2, \ldots\}$. Furthermore, say that there is no tuple in C in which $X_1 = 0$ and $X_2 = 0$. FC applied to the non-binary CSP is unable to detect that every node containing these assignments is a dead end: it can only forward check on the constraint C when all but one of C's variables have been instantiated. On the other hand, FC applied to the hidden is able to detect all of these dead ends: at every such node the h-variable corresponding to C will experience a domain wipe out.

Example 7 Consider a CSP containing two constraints C_1 and C_2 both over the set of variables $\{X_1, X_2, X_k\}$. Say that $C_1 = \{(0,0,0),(1,1,1)\}$ and $C_2 = \{(0,0,1),(1,1,1)\}$. FC applied to the non-binary CSP is able to detect that every node containing the assignments $X_1 = 0$ and $X_2 = 0$ is a dead end: at such nodes the domain of X_k will experience a domain wipe out when we forward check both C_1 and C_2. FC applied to the hidden, on the other hand, is unable to detect a dead end at every such node: assignments to ordinary variables can prune the domains of h-variables but not the domains of other ordinary variables.

These two examples can be used to construct CSPs where FC applied to the non-binary representation performs exponentially better than FC applied to the hidden and vice versa. There is a way, however, of improving FC on the hidden so that it can still perform exponentially better but can only be outperformed by a bounded amount. The intuition behind the improvement comes from Example 7. When FC on the hidden CSP visits a node in which $X_1 = 0$ and $X_2 = 0$ it will reduce the domains of the two h-variables H_1 and H_2 (corresponding to constraint C_1 and C_2 respectively) to the singleton set $\{1\}$, where 1 corresponds to the first satisfying tuple of the constraints. At this point if we continue constraint propagation so that we restore arc-consistency between H_1 and X_k and between H_2 and X_k we would detect the same dead end that FC on the non-binary does.

We can define the following enhancement to FC.

Definition 2 [FC$^+$] FC$^+$ is a backtracking algorithm designed to run on the hidden representation. It operates exactly like FC, except that after forward checking prunes the domain of any h-variable we additionally prune the domains of any uninstantiated variables constrained by that h-variable so as to remove values whose support has been lost. As usual we backtrack if any future variable experiences a domain wipe out.

This enhancement to FC is similar in spirit to those developed by Nadel (Nad89). It fits between standard forward checking and full maintenance of arc-consistency in terms of the amount of constraint propagation it performs. However, we can make the algorithm more efficient than the generic algorithms presented by Nadel because every value for an h-variable functionally determines the values of all the ordinary variables it is constrained with. When we instantiate an ordinary variable we forward check any h-variables it is constrained with in the normal manner. This operation requires a binary constraint check for every domain value of the h-variable. Say that h-variable H_i has had some of its values

pruned. FC$^+$ must then check the domains of all of the unassigned ordinary variables H_i is constrained with. Say that $\{X_1, \ldots, X_k\}$ are the k unassigned variables constrained by H_i. We can restore arc-consistency between H_i and each of these variables by iterating *once* over the remaining domain of H_i. Every unpruned value of H_i supports a unique value for each of the X_j, and in $2k$ operations per value (k binary checks each requiring 2 operations) of H_i we can accumulate the set of still supported values for each of the X_j. Finally, in a second phase we iterate through the domains of the X_j pruning all values not marked as still supported by the first phase.

Counting all of these operations as primitive constraint checks, and using the simplifying assumption that each of the variables in the original non-binary CSP has an identical domain size, we obtain the following result.

Proposition 1 *Given any variable ordering strategy for the non-binary CSP, there exists an ordering strategy such that FC$^+$ applied to the hidden representation will never visit more nodes than FC applied to the non-binary CSP, and it will perform at most* $\max_{i=1}^{m}\big((\mathit{Arity}(C_i)+1)\|C_i\| + \mathit{Arity}(C_i)\big)$ *as many checks.*

The variable ordering employed is exactly the same as that used by FC on the non-binary representation. In particular, we delay instantiating all the h-variables until all of the ordinary variables are instantiated. Once all of the ordinary variables have been instantiated, and we have not experienced a domain wipe-out, each h-variable will have its domain reduced down to one value. In fact the values assigned to the other variables constitute a solution, so search can be terminated prior to visiting any of the h-variables.

These results shed some light on the hidden representation. We see that using the hidden imposes an overhead over direct use of the non-binary representation. Although this overhead is only a multiplicative factor, it can be orders of magnitude: non-binary constraints can often contain a large number of satisfying tuples. On the other hand, if we employ the FC$^+$ algorithm we can potentially save an exponential amount of work by visiting exponentially fewer nodes. When this potential is realized the savings can outweigh the multiplicative overhead. In the next section we show empirically that both outcomes are possible, and we provide some guidelines as to when conversion to the hidden might be effective.

Experimental Comparisons

We now show some experimental results comparing FC on the representations. Throughout this section, FC refers to an implementation of the forward checking algorithm which dynamically orders the variables by selecting as the next variable to instantiate the variable with the minimum remaining values (ties are broken by choosing the variable that participates in the most constraints) and FC$^+$ refers to an implementation of the algorithm of Definition 2 for the hidden variable method. In FC$^+$ we also employ the minimum remaining values heuristic. In particular, we allow h-variables to be instantiated prior to ordinary variables, if they are selected by the heuristic[2]. Finally, we count the checks performed by FC and FC$^+$ in the manner described in the previous section with one refinement: when solving the dual, we charged one constraint check for each shared variable. For example, with reference to Figure 1, checking the constraint between Y_1 and Y_2 costs one and checking the constraint between Y_1 and Y_4 costs two.

To systematically examine the effect of the cardinalities of the constraints and the number of constraints on the cost of solving a non-binary CSP and its corresponding binary representations, we use the following model of a random non-binary CSP. A random CSP has n variables each with domain size of d, and m constraints each with arity k and t satisfying tuples. Each constraint is over a subset of variables chosen with uniform probability from the $\binom{n}{k}$ possible subsets, and each constraint contains $0 \leq t \leq d^k$ tuples chosen at random.

Dual Graph Representation. Figure 3 shows the effect of the number of tuples in the constraints and the number of constraints on the cost of solving a random non-binary CSP and its dual. Specifically, for each parameter settings we generated and solved an ensemble of non-binary problems and their dual representations (a minimum of 30 problems in each ensemble). We then took the ratio of the median consistency checks needed to solve the dual representations over the median consistency checks needed to solve the non-binary CSPs. Finally, we constructed the contour lines shown in the figures using cubic spline interpolation on our data points. For example, the left most contour lines in the figures represent the points in the space of random problems where solving the dual was $10\times$ faster (in terms of consistency checks) than solving the non-binary CSP, and everything to the left of the left most line means the dual was *at least* $10\times$ faster. The experiments show that the dual can be an efficient representation.

We now consider two classes of problems, random 3-SAT and crossword puzzles, and show that the maps constructed from our experimental results over the space of random non-binary problems have predictive power and so provide guidance for selecting between the non-binary and dual CSP models of a problem. The results for random 3-SAT are shown in Table 1. It can be seen that the cost ratios for solving the original CSP and its dual representation fit well with the predictions of the order of magnitude curves (see Figure 3, $k = 3$, at the point 7/8 on the x-axis (each of the constraints has 7 out of the 8 possible tuples)). This provides some evidence that the experimental predictions scale for larger n. The results for 20 crossword puzzles (Gin93) are shown in Table 2. In the non-binary formulation of crossword puzzles there is a variable for each letter to be filled in and the constraints are the words in the Unix dictionary. There are few constraints and each constraint allows few of the possible tuples: at most 4000 out of the 26^k possible tuples for $k = 3, ..10$, where 10 is the length of the longest word in the puzzles to be filled. The order of magnitude

[2]We delay instantiating any h-variables all of whose constrained variables have already been assigned. When this happens the h-variable has become redundant and need not be further considered.

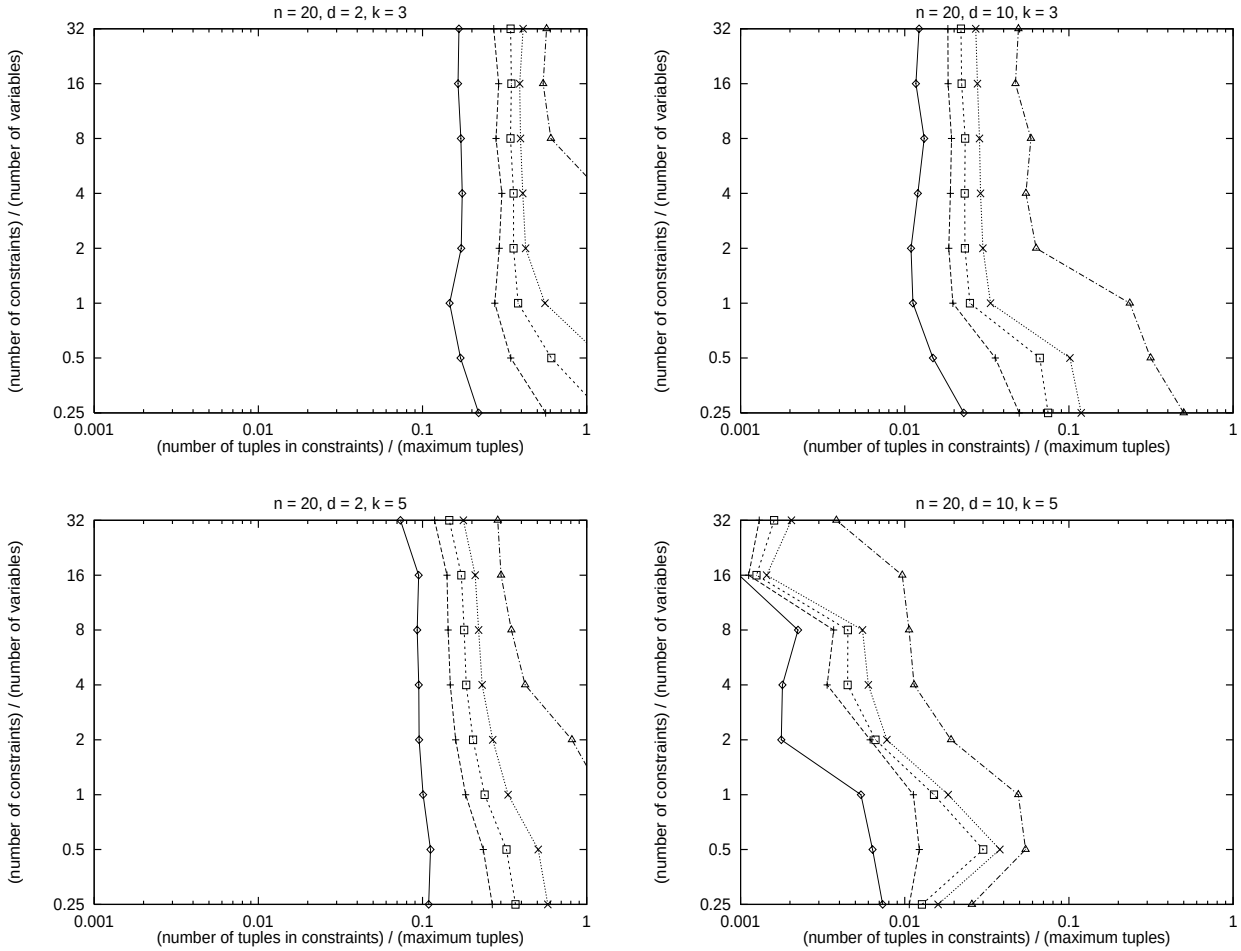

Figure 3: Order of magnitude curves for median cost of dual and original CSP when $n = 20$, $d = 2, 10$, and $k = 3, 5$. From left to right the curves mean: (i) dual is $10\times$ better, (ii) dual is $2\times$ better, (iii) dual and non-binary cost the same, (iv) dual is $2\times$ worse, and (v) dual is $10\times$ worse.

Table 1: Effect of the ratio of clauses to variables (m/n) on the ratio of the average number of consistency checks performed to solve the dual representation over the average number of consistency checks performed to solve the non-binary representation (cost rat.), when finding first solution to random 3-SAT problems with 100 Boolean variables.

m/n	1/4	1/2	1	2	4	8	16	32
cost rat.	0.9	1.5	2.5	4.2	31	33	38	44

curves predict that the dual representation should be more and more efficient as the length of the words in the puzzle grows. This is indeed the case and on the larger problems the dual is at least 1000 times faster.

Hidden Variable Representation. Figure 4 shows the basic behavior of the hidden variable representation on random CSPs. Over a wide range of different values for n, d, k, and m, we have found that it is the number of satisfying tuples in the constraint that determines the relative effectiveness of the hidden variable CSP. To determine which representation is superior, the hidden variable or the original non-binary, we plot the ratio of the average number of constraint checks performed by each algorithm. We run a sufficient number of problems at each data point to obtain averages that have two statistically significant digits. To present the data most effectively we plot the log (base 10) of this ratio: at zero the representations have approximately the same performance, positive numbers represent the number of orders of magnitude the hidden outperforms the non-binary, while negative number similarly represent the number of orders of magnitude the non-binary outperforms the hidden.

The plot shows three sample problem classes, each specified by the numbers $\langle n, d, k, m, t\rangle$. For each problem class we vary the number of satisfying tuples in each constraint from near 0% to near 100%. In the first two problem classes the number of constraints is 10% or less the total possible, while for the last class the number of constraints is 90% the total possible. The graphs show what we have also seen in

Table 2: Number of consistency checks performed when finding one solution to crossword puzzles. The absence of an entry indicates that the problem could not be solved within 5×10^8 consistency checks. For the dual problem, n is the number of variables and m is the number of constraints; for the non-binary problems, m is the number of variables and n is the number of constraints.

	original	dual	size	
puzzle	cc's	cc's	n	m
1	52	292	4	3
2	138	3,267	6	8
3	185	11,404	8	14
4	208	10,481	12	14
5	12,509	18,686	10	19
6	12,636	19,876	14	21
7	556,660	28,962	14	23
8	5,780,710	27,656	20	30
9	____	47,167	26	41
10	____	224,258	18	35
11	____	29,777	24	40
12	____	729,125	18	38
13	28,446,460	50,281	28	52
14	____	72,307	34	65
15	____	254,381	28	62
16	____	258,780	46	95
17	____	178,885	68	135
18	____	248,877	88	180
19	____	587,826	86	177
20	____	____	80	187

other problem classes: if the constraints have few satisfying tuples the hidden variable CSP can outperform the non-binary CSP by many orders of magnitude; the performance advantage of the hidden variable CSP decreases as we increase the number of satisfying tuples in the constraints and there is some threshold beyond which the non-binary becomes more effective; and finally, as our theory predicts the potential gain of the non-binary over the hidden is much less than the other way around. These results makes sense in terms of what FC$^+$ is doing. As the constraints have fewer and fewer satisfying tuples the FC$^+$ is able to detect a larger and larger number of extra deadends over FC running on the non-binary.

It is important to note that hard problems exist at all values of t. In particular, the hidden variable representation can be superior on hard problems: when the constraints contain a small number of satisfying tuples we can generate hard problems by increasing n or by decreasing m, both of these changes decrease the number of constraints each variable participates in. In Figure 5 we plot the average number of constraint checks required by FC$^+$ to solve the hidden variable CSP and the average number of checks required by FC to solve the equivalent non-binary CSP. In this plot we vary the number of constraints m. The graph shows something like the classic easy/hard/easy regions (smoothed out to some extent by our use of a log scale) as the varying number of constraints change the problems from solvable to unsolvable, and we see that although FC$^+$ on the hidden

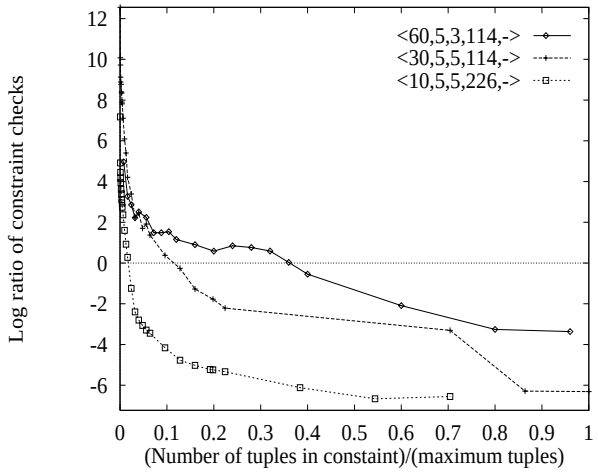

Figure 4: Log of the number of constraints checks performed by FC$^+$ on the hidden divided by the number of constraint checks performed by FC on the non-binary.

variable CSP never finds this particular problem class very difficult it becomes quite difficult for FC on the non-binary: at the peak (when we have a total of 10 constraints) it requires about 30 million checks on average to solve a problem vs. an average of 16 thousand checks for the hidden. More importantly is the fact that in this peak area FC on the non-binary requires in excess of 500 million checks on 3% of the problems while using the hidden the *same problems* were solved using less than 35 thousand checks. This phenomenon seems to be related to the exceptionally hard problems reported by Smith and Grant in (SG95). In particular, it indicates that for this problem class the distribution of constraint checks performed by FC$^+$ displays a lower variance as well as a lower mean.

Conclusions and Future Work

We examined how well the FC algorithm performs on a non-binary CSP in comparison to its performance on binary translations of the CSP.

Our experiments show that the dual graph representation can be more efficient by orders of magnitude, when the number of constraints is low relative to the number of variables, and the constraints are restrictive. As well, for the hidden variable representation, we showed that a modified forward checking algorithm which we call FC$^+$, can sometimes perform exponentially better than simply using FC on the non-binary, and sometimes it can be outperformed by a bounded (but sometimes large) amount. We have also provided better insights into the behavior and nature of the two binary translations. Translating a non-binary CSP involves some overhead, and we view this work as providing some initial intuitions as to when such a translation is worthwhile. Empirically, we have shown that the number of satisfying tuples in the constraints is perhaps the most important factor in determining how worthwhile the translation is.

An important question that we have not addressed here is

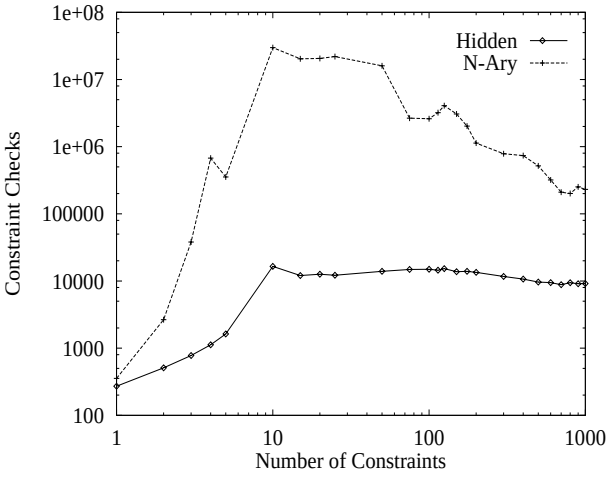

Figure 5: Number of constraints checks performed by the two algorithms on the problem class $\langle 30, 5, 5, -, 40 \rangle$.

the relationship between the two binary translations. When is the dual representation to be preferred to the hidden variable representation and vice versa? Are there any theoretical results that can be proved about their relative behaviour? We intend to address these questions in future work.

One thing, however, we feel that our data has demonstrated is that the translation to the binary has promise as a solution technique for non-binary CSPs. In the end, however, it could well be that insights about when these translations perform better, can be carried over directly to the non-binary case, so that improved methods for solving non-binary CSPs can be developed that avoid the overhead of the translation entirely. It should be clear, however, that studying and understanding these binary translations is an essential prerequisite to achieving such insights.

Acknowledgments

This work was supported by the Canadian Government through their NSERC and IRIS programs.

References

[Dec90] R. Dechter. On the expressiveness of networks with hidden variables. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 556–562, Boston, Mass., 1990.

[DM94] M. J. Dent and R. E. Mercer. Minimal forward checking. In *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, pages 432–438, New Orleans, LA, 1994.

[DP89] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[Fre78] E. C. Freuder. Synthesizing constraint expressions. *Comm. ACM*, 21:958–966, 1978.

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

[HE80] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Mac77] A. K. Mackworth. On reading sketch maps. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 598–606, Cambridge, Mass., 1977.

[McG79] J. J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Inform. Sci.*, 19:229–250, 1979.

[Nad89] B. A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[Pei33] C. S. Peirce. In C. Hartshorne and P. Weiss, editors, *Collected Papers, Vol. III*. Harvard University Press, 1933. Cited in: F. Rossi, C. Petrie, and V. Dhar, 1989.

[RPD89] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. Technical Report ACT-AI-222-89, MCC, Austin, Texas, 1989. A longer version of (RPD90), with more details and results on the hidden variable method.

[RPD90] F. Rossi, C. Petrie, and V. Dhar. On the equivalence of constraint satisfaction problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, Stockholm, Sweden, 1990.

[SG95] B. M. Smith and S. A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 646–651, Montreal, 1995.

[SRGV96] T. Schiex, J.-C. Régin, C. Gaspin, and G. Verfaillie. Lazy arc consistency. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 216–221, Portland, Oregon, 1996.

[VH89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.