# Encodings of Non-Binary Constraint Satisfaction Problems

**Kostas Stergiou** and **Toby Walsh**
Department of Computer Science
University of Strathclyde
Glasgow G1 1XL
Scotland
{ks,tw}@cs.strath.ac.uk

**Abstract**

We perform a detailed theoretical and empirical comparison of the dual and hidden variable encodings of non-binary constraint satisfaction problems. We identify a simple relationship between the two encodings by showing how we can translate between the two by composing or decomposing relations. This translation suggests that we will tend to achieve more pruning in the dual than in the hidden variable encoding. We prove that achieving arc-consistency on the dual encoding is strictly stronger than achieving arc-consistency on the hidden variable, and this itself is equivalent to achieving generalized arc-consistency on the original (non-binary) problem. We also prove that, as a consequence of the unusual topology of the constraint graph in the hidden variable encoding, inverse consistencies like neighborhood inverse consistency and path inverse consistency collapse down onto arc-consistency. Finally, we propose the "double encoding", which combines together both the dual and the hidden variable encodings.

## Introduction

Many constraint satisfaction problems (CSPs) can be compactly formulated using non-binary relations. We can solve a non-binary CSP either by using one of the algorithms like forward checking (FC) which have been generalized to non-binary constraints or by translating it into a binary CSP. There exist two well known methods for translating non-binary CSPs into binary CSPs: the dual encoding (sometimes call the "dual graph method") and the hidden variable encoding. Recently, Bacchus and van Beek have started to compare how backtracking algorithms like FC perform on the two encodings and on the original non-binary problem (Bacchus & van Beek 1998). Their ultimate aim is to provide guidance on when to translate. We continue this research programme, focusing on higher levels of consistency like arc-consistency (AC) and on the comparison of the two encodings. Bacchus and van Beek remark "…An important question that we have not addressed here is the relationship between the two binary translations. When is the dual representation to be preferred to the hidden representation and vice versa? Are there any theoretical results that can be proved about their relative behaviour? …" p.317-8 of (Bacchus & van Beek 1998). In this paper, we provide answers to many of these questions. In addition, we propose a new encoding which combines together both the dual and the hidden variable encodings.

## Formal background

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 solution 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 (Debruyne & Bessiere 1997) for references). A problem is $(i, j)$-consistent iff it has non-empty domains and any consistent instantiation of $i$ variables can be extended to a consistent instantiation involving $j$ additional variables. A problem is arc-consistent (AC) iff it is $(1, 1)$-consistent. A problem is path-consistent (PC) iff it is $(2, 1)$-consistent. A problem is strong path-consistent iff it is $(j, 1)$-consistent for $j \leq 2$. A problem is path inverse consistent (PIC) iff it is $(1, 2)$-consistent. A problem is neighborhood inverse consistent (NIC) iff any value for a variable can be extended to a consistent instantiation for its immediate neighborhood. A problem is restricted path-consistent (RPC) iff it is arc-consistent and if a variable assigned to a value is consistent with just a single value for an adjoining variable then for any other variable there exists a value compatible with these instantiations. A problem is singleton arc-consistent (SAC) iff it has non-empty domains and for any instantiation of a variable, the problem can be made arc-consistent.

Many 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 (Mohr & Masini 1988). We can also maintain a level of consistency at every node in a search tree. For example, the MAC algorithms for binary CSPs maintains arc-consistency at each node in the search tree (Gaschnig 1979). As a second example, the forward checking algorithm (FC) maintains a restricted form of arc-consistency that ensures that the most recently instantiated variable and those that are uninstantiated are arc-consistent.

Following (Debruyne & Bessiere 1997), we call a consistency property $A$ stronger than $B$ ($A \geq B$) iff in any problem in which $A$ holds then $B$ holds, and strictly stronger ($A > B$) iff it is stronger and there is at least one problem in which $B$ holds but $A$ does not. We call a local consistency property $A$ incomparable with $B$ ($A \sim B$) iff $A$ is not stronger than $B$ nor vice versa. Finally, we call a local consistency property $A$ equivalent to $B$ iff $A$ implies $B$ and vice versa. The following identities summarize results from (Debruyne & Bessiere 1997) and elsewhere: strong PC $>$ SAC $>$ PIC $>$ RPC $>$ AC, NIC $>$ PIC, NIC $\sim$ SAC, and NIC $\sim$ strong PC.

## Encodings of non-binary problems

### Dual encoding

The dual encoding simply swaps the variables for constraints and vice versa. There is a dual variable $v_c$ for each $k$-ary constraint $c$. Its domain is the set of consistent tuples for that constraint. For each pair of constraints $c$ and $c'$ in the original problem with variables in common, we introduce a compatibility constraint between the dual variables $v_c$ and $v'_c$. This constraint restricts the dual variables to tuples in which the variables that are shared take the same value.

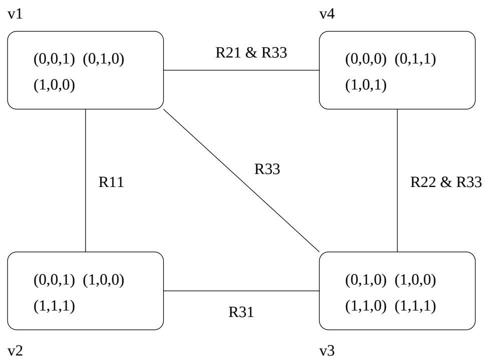

**Fig. 1.** Dual encoding of a non-binary CSP. $Rij$ is the binary relation on a pair of tuples that is true iff the $i$th element of the 1st tuple equals the $j$th element of the 2nd tuple.

Consider an example with six 0-1 variables, and four arithmetic constraints: $x_1+x_2+x_6 = 1$, $x_1-x_3+x_4 = 1$, $x_4 + x_5 - x_6 \geq 1$, and $x_2 + x_5 - x_6 = 0$. The dual encoding represents this problem with 4 dual variables, one for each constraint. The domains of these dual variables are the tuples that satisfy the respective constraint. For example, the dual variable associated with the third constraint $v_3$ has the domain $\{(0,1,0),(1,0,0),(1,1,0),(1,1,1)\}$ as these are the tuples of values for $(x_4, x_5, x_6)$ which satisfy $x_4 + x_5 - x_6 \geq 1$. The dual encoding of the problem is shown in Figure 1.

### Hidden variable encoding

The hidden variable encoding also introduces a dual variable $v_c$ for each (nonbinary) constraint $c$. Its domain is again the set of consistent tuples for the variables in the constraint $c$. For each tuple in the domain of the dual variable $v_c$, we introduce compatibility constraints between $v_c$ and each variable $x_i$ in the constraint $c$. Each constraint specifies that the tuple assigned to $v_c$ is consistent with the value assigned to $x_i$. Consider again the example with four arithmetic constraints. There are, in addition to the original six 0-1 variables, four dual variables with the same domains as in the dual encoding. For example, the dual variable associated with the third constraint $v_3$ again has the domain $\{(0,1,0),(1,0,0),(1,1,0),(1,1,1)\}$. There are now compatibility constraints between $v_3$ and $x_2$, between $v_3$ and $x_5$ and between $v_3$ and $x_6$, as these are the variables mentioned in the third constraint. For example, the compatibility constraint between $v_3$ and $x_2$ is the relation that is true iff the first element in the tuple assigned to $v_3$ equals the value of $x_2$. The hidden variable encoding is shown in Figure 2.

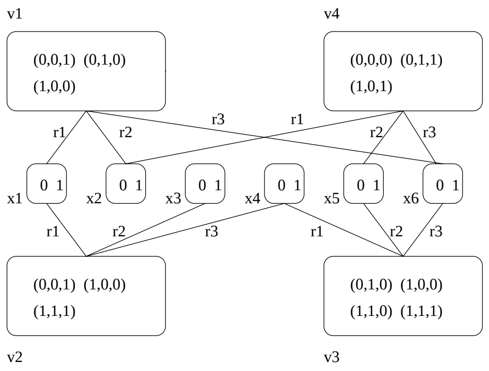

**Fig. 2.** Hidden variable encoding of a non-binary CSP. The binary relation $ri$ applies to a tuple and a value and is true iff the $i$th element of the tuple equals the value.

## Mapping between the encodings

We can construct the dual encoding by performing a (solution preserving) simplification of the hidden variable encoding. This mapping compiles out the role of the original variables by composing binary relations. Given the constraint graph of a hidden variable encoding, we collect the set of paths of length 2 between each pair of dual variables, $v_c$ and $v_c'$. We delete these paths and replace them with a single constraint formed from the composition of the relations on the deleted paths. When any of the original variables becomes isolated from the rest of the constraint graph, or connected just to a single dual variable, we can safely delete it. Applying these simplifying transformations to every pair of dual variables transforms the hidden variable encoding into the dual.

As an example, consider the hidden variable encoding in Figure 2. We take a pair of dual variables, $v_1$ and $v_2$. The variable $x_1$ lies on the only path of length 2 between $v_1$ and $v_2$. We delete the two $r1$ constraints on this path and add a new relation $R11$ that is the composition of the two $r1$ relations. As $x_1$ is now isolated from the rest of the constraint graph, we can safely delete it. We then take another pair of dual variables, say $v_1$ and $v_4$. There are two paths of length 2 between $v_1$ and $v_4$. We delete both of these paths. The path between $v_1$ and $v_4$ via $x_2$ had $r2$ and $r1$ constraints on it so induces a constraint $R21$ between $v_1$ and $v_4$. Similarly, the path between $v_1$ and $v_4$ via $x_6$ had two $r3$ constraints on it so induces a constraint $R33$ between $v_1$ and $v_4$. We therefore add their union $R21\&R33$ as the new induced constraint between $v_1$ and $v_4$. We can now delete $x_2$ and $x_3$ as they are isolated from the rest of the problem. The intermediate CSP constructed at this point is shown in Figure 3. If we continue this simplification on the remaining pairs of dual variables, we construct the dual encoding shown in Figure 1.

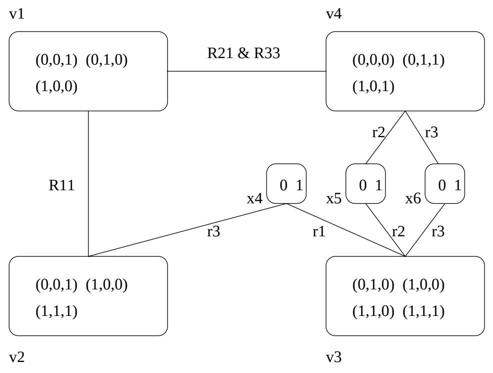

**Fig. 3.** An intermediate point in the transformation of the hidden variable encoding into the dual variable encoding.

The transformation of the hidden variable encoding into the dual can easily be reversed. We simply take constraints between pairs of dual variables and decompose them into paths of constraints which take in the original variables. Note that, at any point in the transformation, we still have a binary CSP. It is therefore possible to have "hybrid" encodings, in which some parts of the problem are represented by a dual encoding and others by a hidden variable encoding.

It is also possible to construct a "double" encoding in which both the dual and the hidden variable encoding are present in their entirety. In the double encoding, we have both the original variables and the dual variables. We also have both the constraints between dual variables (as in the dual encoding), and the constraints between dual variables and the original variables (as in the hidden variable encoding). In the double encoding, we will have the extra pruning achievable in the dual encoding. We will also be able to branch on the original variables as in the hidden variable encoding; branching heuristics may be able to perform better on these than on the dual variables with their potentially large domains.

We believe that this may be the first time this close relationship between the hidden variable and the dual encoding has been identified. This observation has several practical and theoretical consequences. For example, our ability to propagate efficiently and effectively through the compatibility constraints in the hidden variable encoding is likely to be a major cause of difference between the dual and the hidden variable encodings. As a second example, enforcing the same level of consistency in the two encodings will inevitably do more pruning in the dual than in the hidden variable encoding.

## Theoretical comparison

To compare constraint propagation in a non-binary problem and its encoding, we must tackle a variety of problems.

First, a static analysis in which we simply compare the levels of consistency in a problem and in its encoding is not very informative. The problem is that the translation into the encoding builds in a significant level of consistency. For example, the dual variables only have consistent tuples in their domains. Hence, by construction, the dual encoding has a level of consistency at least equal to GAC in the original problem. A solution to this problem is to perform a more dynamic analysis in which we compare the levels of consistency *achieved* by constraint propagation during search.

Second, constraint propagation may infer nogoods involving the dual variables, and these cannot be directly compared with nogoods inferred in the original problem. A solution to this problem is simply to translate nogoods involving dual variables into nogoods involving the original variables and values. For example, if we prune a $k$-ary tuple from the domain of a dual variable, we translate this into a $k$-ary nogood on the original variables.

Third, if we prune values or instantiate variables in the original problem, we cannot perform the same simplification on the dual encoding as the original variables have been discarded. Any solution to this problem should ensure that, when all the variables in a constraint $c$ have been instantiated, the dual variable $v_c$ is reduced to an unitary domain. The solution we adopt is to remove any tuples from the domains of dual variables that contain the value pruned or that are not consistent with the variable instantiation in the original problem. Note that the reverse direction is not problematic. For instance, if we instantiate a dual variable $v_c$, we can simply instantiate all variables $x_i$ in the original problem which appear in $c$.

Fourth, constraint propagation in the dual encoding will infer nogoods which, when translated back into the original problem, have a large arity. Constraint propagation in the original problem may infer much fewer but much smaller nogoods which can be derived from these larger arity nogoods. For example, constraint propagation in the dual encoding may remove all tuples from a dual variable which assign the value $a_i$ to a variable $x_i$. ¿From this, we can derive an unitary nogood that removes $a_i$ from the domain of $x_i$. A solution to this problem is to compare the nogoods that can be derived from the translated nogoods with those that can be derived in the original problem.

We will therefore call achieving a consistency property $A$ stronger than achieving $B$ iff the nogoods identified when achieving $B$ are derivable from those identified when achieving $A$, and strictly stronger if it is stronger and there exists at least one problem on which one nogood identified when achieving $A$ is not derivable from those identified when achieving $B$.

## Hidden variable encoding

If we ignore pruning of values in the dual variables, enforcing AC on the hidden variable encoding is equivalent to enforcing GAC on the original problem.

**Theorem 1** *Achieving AC on the hidden variable encoding is equivalent to achieving GAC on the variables in the original problem.*

**Proof:** Assume that, after removing some values from the domains of variables in the original problem, we make the problem GAC and this prunes the value $a_i$ from a variable $x_i$. Then there exists some constraint $c$ mentioning $x_i$ and the assignment of $a_i$ to $x_i$ cannot be consistently extended to the other variables in $c$. In the hidden variable encoding, enforcing AC between these variables and $v_c$ will remove all tuples that assign $a_i$ to $x_i$. Hence, considering the arc between $x_i$ and $v_c$, if we assign $a_i$ to $x_i$, there are no tuples in the domain of $v_c$ that are consistent. Hence, achieving AC on the hidden variable encoding will prune $a_i$ from the domain of $x_i$.

Assume that we make the problem AC and this prunes the value $a_i$ from a variable $x_i$. Then there exists a dual variable $v_c$ in the hidden variable encoding where $c$ mentions $x_i$ and none of the tuples left in the domain of $v_c$ assigns $a_i$ to $x_i$. Hence, in the original representation of the problem, the assignment of $a_i$ to $x_i$ cannot be consistently extended to the other variables in $c$. We will therefore prune $a_i$ from the domain of $x_i$. □

This theorem shows that we can achieve GAC by means of a simple AC algorithm and the hidden variable encoding. Whether this is computationally effective will depend on the tightness and arity of the non-binary constraint. The best AC algorithm has worst-case time complexity of $O(d^2)$ where $d$ is the domain size. For the hidden variable encoding of very loose $k$-ary constraints, this may be $O(m^{2k})$ where $m$ is the domain size in the original problem. By comparison, we may be able to achieve GAC at much less cost. For example, GAC on all different constraints takes just $O(m^2k^2)$ worst-case time (Régin 1994).

In practive, we may see different results with a hidden variable encoding as we can now branch on the hidden variables and reason explicitly about the valid tuples in their domains. For example, consider a parity constraint even$(x_1 + x_2 + x_3)$ on three 0-1 variables. If we remove 1 from the domain of $x_1$ then the problem remains GAC, and we do will not *explicitly* perform any pruning. However, in the hidden variable encoding, achieving AC will prune two of the four values from the dual variable leaving just the tuples, $(0, 0, 0)$ and $(0, 1, 1)$. In other words, in the hidden variable encoding, we identify the additional constraint that $x_2 = x_3$.

The constraint graph of a hidden variable encoding has a star-shaped topology in which constraints "radiate" out from the original variables. Because of this topology, certain consistency techniques fail to achieve any additional pruning over AC. In particular, NIC collapses down onto AC. Other lesser levels of consistency between NIC and AC (like PIC and RPC) therefore collapse down onto AC. Note that, as we are comparing levels of consistency in the hidden variable encoding, stronger than or equal to AC, we can perform a static analysis that does not worry about the level of consistency built into the encoding.

**Theorem 2** *On a hidden variable encoding, NIC is equivalent to AC.*

**Proof:** Consider a hidden variable encoding that is AC. The proof divides into two cases. In the first case, consider a hidden variable and its immediate neighborhood. As the problem is AC, the hidden variable has a non-empty domain. Take any tuple in this domain, say $a_0 \times \ldots \times a_k$. Then the neighboring (non-hidden) variables, $x_0$ to $x_k$ can consistently take the values $a_0$ to $a_k$. In the second case, consider a non-hidden variable $x_0$ and its immediate neighborhood. Pick any value $a_0$ from the domain of $x_0$. Now, as the problem is AC, we can pick values for any of the neighboring hidden variables. As these are not connected directly to each other, these values are consistent with each other. Hence the problem is NIC. □

Whilst we do not get any more pruning over AC by enforcing inverse consistencies like NIC and PIC, there are levels of consistency stronger than AC that it can be useful to enforce.

**Theorem 3** *On a hidden variable encoding, strong PC is strictly stronger than SAC, which itself is strictly stronger than AC.*

**Proof:** Consider a problem with a single parity constraint, even$(x_1 + x_2 + x_3)$ with variable $x_1$ set to 0, and variables $x_2$ and $x_3$ having 0-1 domains. The hidden variable encoding of this problem is SAC but enforcing strong PC adds the additional constraint that $x_2 = x_3$.

Consider a problem with three parity constraints: even$(x_1 + x_2 + x_3)$, even$(x_1 + x_3 + x_4)$, and even$(x_1 + x_4 + x_2)$. If $x_1$ is assigned to 1, and every other variable has a 0-1 domain then the hidden variable encoding is AC but it is not SAC. Enforcing SAC will show that the problem is insoluble. □

## Dual encoding

The dual encoding binds together the (non-binary) constraints much more tightly than the hidden variable encoding. As a consequence, constraint propagation in the dual can achieve high levels of consistency in the original (non-binary) problem.

**Theorem 4** *Achieving AC on the dual encoding is strictly stronger than achieving GAC on the original problem.*

**Proof:** Assume that, after removing some values from domains of variables in the original problem, we make the problem GAC and this prunes the value $a_i$ from variable $x_i$. Then there exists some constraint $c$ mentioning $x_i$ and the assignment of $a_i$ to $x_i$ cannot be consistently extended to the other variables in $c$. In the dual encoding, we remove tuples from the domains of the dual variables that assign values to variables in the original problem that have been removed. This will remove all tuples in $v_c$ that assign $a_i$ to $x_i$. Hence, we can derive the nogood that $a_i$ cannot be assigned to $x_i$. To show strictness, consider two parity constraints, even$(x_1 + x_2 + x_3)$ and even$(x_2 + x_3 + x_4)$ with $x_1$ assigned to 1, $x_4$ assigned to 0, and all other variables having 0-1 domains. Each constraint is GAC. However, achieving AC on the dual encoding will prove that the problem is insoluble since $x_2 + x_3$ cannot be both even and odd. □

This extra pruning may come at computational cost if the non-binary constraints have a large arity and are loose. As predicted earlier, AC on the dual encoding is more pruningful than AC on the hidden variable encoding.

**Theorem 5** *Achieving AC on the dual encoding is strictly stronger than achieving AC on the hidden variable encoding.*

**Proof:** The proof follows from Theorems 1 and 4. □

These results can be extended to rank algorithms that maintain AC and GAC during search, using arguments similar to (Kondrak & van Beek 1997). For example, under a suitable static variable and value ordering, MAC on the dual encoding strictly dominates MAC on the hidden variable encoding, which itself will be equivalent to an algorithm that maintains GAC on the non-binary representation.

## Experimental results

To support our theoretical results, we experimented with two domains that involve non-binary constraints: Golomb rulers and crossword puzzle generation.

Table 1 compares three encodings of some standard crossword puzzles. The worst-time complexity of GAC on the non-binary encoding of the puzzles is in the order of $O(m^k)$, where $m$ is the number of letters in the alphabet and $k$ is the length of the words. In the puzzles we generated, $k$ was up to 10. This obviously makes the non-binary encoding completely impractical, so we did not consider it in the experiments. The small domain size of the original variables compared to the dual variables makes the hidden representation better. Note, that because of the large size of the dual variables, AC is expensive and it may be the case that forward checking is enough to solve these problems in reasonable time.

| Size<br>n - m | Dual<br>Br.-CPU | Hidden<br>Br.-CPU | Double<br>Br.-CPU |
|---|---|---|---|
| 68 - 135 | 18 - 488.75 | 2 - 53.85 | 2 - 552.66 |
| 88 - 180 | 11 - 550.4 | 0 - 78.15 | 0 - 632.03 |
| 86 - 177 | 50 - 451 | 5 - 73.52 | 5 - 564.88 |
| 80 - 187 | 34 - 900.95 | 19 - 93 | 19 - 1272 |
| 64 - 128 | 3 - 298.5 | 11 - 53.4 | 11 - 309.24 |
| 12 - 36 | 346 - 901.16 | 138 - 60.88 | 138 - 485.18 |

Table 1: Branches and CPU time when generating crossword puzzles. Fail First was used for variable ordering. $n$ is the number of variables and $m$ is the number of constraints.

A *Golomb Ruler* can be represented by a set of $n$ variables of domain size $m$, such that $x_1 < x_2 < \ldots < x_n$, $x_1 = 0$, and the $n(n-1)/2$ differences $x_j - x_i$, $1 \leq i < j \leq n$, are distinct. Such a ruler has $n$ marks and is of length $m$. The constraints can be encoded by adding an auxiliary variable $x_{ji}$ for each difference $x_j - x_i$, such that $x_j - x_i = x_{ji}$, and then constraining all of the auxiliary variables to be distinct. This gives $n(n-1)/2$ ternary constraints and a clique of binary `not equals' constraints. Table 2 compares MGAC on the ternary representation to MAC on the hidden and double representations. As Theorem 1 predicted, MGAC in the non-binary encoding explores the same number of branches as MAC in the hidden. The extra filtering in the double encoding reduces the number of branches. In terms of CPU time, though, the non-binary encoding is the clear winner with the double performing poorly. The dual encoding for this problem is impractical because of the large domain size ($O(m^4)$) of the dual variables needed to represent the not-equals constraints. Such constraints are redundant in the double so they can be ignored.

| Ruler | Ternary | | Hidden | Double | |
|---|---|---|---|---|---|
| n-m | Branches | CPU | CPU | Branches | CPU |
| 7-25 | 12 | 0.2 | 0.45 | 12 | 2.42 |
| 7-24 | 436 | 1.36 | 3.37 | 382 | 9.03 |
| 8-34 | 35 | 0.7 | 1.75 | 35 | 14.84 |
| 8-33 | 2585 | 12.82 | 31.3 | 2139 | 94.06 |
| 9-44 | 283 | 4.26 | 9.71 | 257 | 80.09 |
| 9-43 | 15315 | 111.97 | 261.68 | 11170 | 824.17 |
| 10-55 | 1786 | 27.63 | 60.24 | 1455 | 444.66 |
| 10-54 | 73956 | 862.56 | 1861.93 | * | * |

Table 2: Branches explored and CPU time (seconds) used to find an optimal golomb ruler or prove that none exists. The variables were ordered lexicographically. The numbers of branches in the hidden representation are not given because they are always equal to the corresponding numbers in the non-binary representation. A * means that there was a cut off after 1 hour of CPU.

## Related work

Bacchus and van Beek present one of the first detailed experimental and theoretical studies of the hidden variable and dual encodings (Bacchus & van Beek 1998). However, their analysis is restricted to the FC algorithms (and a simple extension called FC+). Our analysis identifies the benefits of enforcing higher levels of consistency. Such analysis is valuable as toolkits like ILOG's Solver enforce these higher levels of consistency during search. Bacchus and van Beek also do not study the relationship between the two encodings. Our results identify a simple mapping between the two. This mapping motivates many of our theoretical results.

Dechter has studied the trade-off between the number of hidden variables and their domain size (Dechter 1990). She shows that any $n$-ary constraint $R$ can be represented by $(|R|-2)/(k-2)$ hidden variables of domain size $k$ where $|R|$ is the number of allowed tuples in the constraint. As required, when $k = |R|$, this degenerates to a single hidden variable for each $n$-ary constraint.

## Conclusions

We have performed a detailed theoretical and empirical comparison of the dual and hidden variable encodings of non-binary constraint satisfaction problems. We have shown how the hidden variable encoding can be transformed into the dual encoding by composing relations. Motivated by this observation, we proved that achieving arc-consistency on the dual encoding is strictly stronger than achieving arc-consistency on the hidden variable, and this itself is equivalent to achieving generalized arc-consistency on the original (non-binary) problem. We also proved that, as a consequence of the unusual topology of the constraint graph in the hidden variable encoding, inverse consistencies like neighborhood inverse consistency and path inverse consistency collapse down onto arc-consistency. Finally, we proposed the double encoding, which combines together both the dual and the hidden variable encodings.

What general lessons can be learnt from this study? First, there is a simple relationship between the dual and the hidden variable encoding based on the composition and decomposition of the binary relations. Second, this relationship suggests that enforcing the same level of consistency in the two encodings will do more pruning in the dual. We were able to prove this conjecture theoretically. Third, it may pay to encode a non-binary constraint satisfaction problem into a binary form. We can, for instance, achieve generalized arc-consistency on a non-binary problem by enforcing arc-consistency on the hidden variable encoding. It remains to be seen if these lessons can be translated back into the solution of non-binary problems without the overhead of encoding.

## References

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

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

Dechter, R. 1990. On the expressiveness of networks with hidden variables. In *Proceedings of the 8th National Conference on AI*, 555–562. American Association for Artificial Intelligence.

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

Kondrak, G., and van Beek, P. 1997. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence* 89:365–387.

Mohr, R., and Masini, G. 1988. Good old discrete relaxation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-88)*, 651–656.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on AI*, 362–367. American Association for Artificial Intelligence.