A new encoding of AllDifferent into SAT

Ian P. Gent and Peter Nightingale

University of St Andrews, Fife, Scotland
{ipg, pn}@dcs.st-and.ac.uk

Abstract. For propositional satisfiability (SAT) solvers to be used for solving the constraint satisfaction problem (CSP), efficient encodings are required. The AllDifferent constraint is well used in constraint programming. This paper presents a new encoding for the AllDifferent constraint. This is compared, theoretically and empirically, to a simple and common encoding. The new encoding presented scales better than the common (or any other known) encoding, however in the empirical evaluation we found it does not perform as well as the common encoding on feasibly sized instances.

1 Introduction

Solvers for propositional satisfiability (SAT) are highly developed and efficient, incorporating advanced techniques such as conflict learning, backjumping and tuned heuristics, implemented using techniques such as watched literals. Constraint solvers are not typically optimized to the same level, but they offer many useful types of constraints. Some of these constraints encode neatly into SAT, such as arbitrary binary constraints. However, others do not, such as the global cardinality constraint described below.

Our motivation for developing an encoding of the AllDifferent constraint is to make use of the efficiency of SAT solvers on problems with AllDifferent constraints. The usual approach to encoding the AllDifferent is to decompose it into pairwise binary constraints $X \neq Y$, then encode these using the direct encoding [2]. This is referred to below as the pairwise encoding. The new encoding presented here maintains the propagation properties of the pairwise encoding while scaling better in formula size.

In the following discussion, n is the number of variables contained in a constraint, d is the largest domain size of the variables in the constraint, and e is the total number of constraints.

Bailleux and Boufkhad [1] considered the cardinality constraint on a set of Boolean variables (i.e. constraining the number of variables that can be assigned the value 1), and produced an encoding with $O(n^2)$ clauses (of length at most 3) and $O(n \log n)$ variables. Unit propagation over this encoding restores global arc-consistency (GAC) in $O(n^2)$ time although only setting $O(n \log n)$ variables. There is a GAC algorithm which runs in $O(n)$ time, so this encoding is suboptimal in that sense. However, it may be the optimal SAT encoding.

For arbitrary binary constraints, Gent [2] described the support encoding and showed that unit propagation on the encoding establishes arc-consistency. Unit propagation takes $O(ed^2)$ time, which is the same as the optimal algorithm running on an extensional representation of the constraints. We call this a *propagation optimal* encoding.

There are several consistency notions for the AllDifferent constraint [3]. The three most common are global arc-consistency (GAC), range consistency (RC) and arc-consistency over the pairwise binary decomposition (AC). GAC is the strongest, followed by RC, followed by AC. With the example $A, B \in \{1, 3\}, C \in \{1, 2\}, \mathsf{AllDifferent}(A, B, C)$, GAC would remove 1 from C, since 1 must be used by either A or B. However RC would not. With the slightly different example $A, B \in \{1, 2\}, C \in \{1, 3\}, \mathsf{AllDifferent}(A, B, C)$, RC can identify the range $1 \ldots 2$ and thus prune 1 from C. AC can only prune when variables become fully instantiated.

We consider the simplest consistency notion, AC. When the domain of a variable contains only one value, that value is removed from the domain of all the other variables in the constraint. This is repeated as long as possible. If two variables are set to the same value, the constraint fails. This algorithm performs $O(n^2)$ value removals before reaching the fixed point. Storing the constraint as a list of variables requires $O(n)$ space.

In this paper we present an encoding of the AllDifferent constraint with $O(nd)$ clauses (of length no more than 3). When unit propagation is performed on the encoding, it does the same work as the simple propagation algorithm outlined above, setting $O(n^2)$ Boolean variables. We assume that setting a variable in SAT and removing a value from a domain in CSP have the same time complexity. If $n \geq d$ the encoding is propagation optimal. This arises firstly with permutations ($n = d$, $\forall x, y \bullet D_x = D_y$) and secondly when the domains are not equal. In this second situation, if all domains were equal and $n > d$, then the constraint would be false by the pigeonhole principle. However, the pigeonhole principle does not apply when the domains are not equal. For example, $A, B \in \{1, 3\}, C \in \{1, 2\}, \mathsf{AllDifferent}(A, B, C)$ has maximum domain size $d = 2$, but there is a satisfying assignment.

However, since the size is $O(nd)$, when $d > n$ (or, more accurately, when d is not $O(n)$, so $\lim_{d \to \infty} n/d = 0$) the encoding should not be considered propagation optimal.

2 Encodings

The main concept in these encodings is constraining the number of Boolean variables set to true. For some set S of Boolean variables, we need to constrain them to have at least one, exactly one, or at most one set to true, so correspondingly we describe at-least-one (ALO), exactly-one (EO) and at-most-one (AMO) encodings. For this section, members of S are referred to as x_i, where $i \in 1 \ldots |S|$.

An ALO encoding for the set S is simply the following.

$$\left(\bigvee_{|S|}^{i=1} x_i \right) \tag{1}$$

To create an EO clause set for a set of size n, we could combine ALO and AMO. Another possibility is to use a structure used by Gent, Prosser and Smith [4,5] and independently by Ansótegui and Manyà [6] in their regular and half regular mappings. The structure (which is referred to as *ladder* from here on) consists of a sequence of $p = n-1$ additional Boolean variables, $y_1 ... y_p$ (referred to as the ladder variables) and a set of clauses (defined below).

A complete *valid* assignment of the ladder variables has no adjacent pair of variables y_r, y_{r+1} where $y_r = False \wedge y_{r+1} = True$. It consists of a sequence of zero or more true assignments, and all following variables are assigned false. Hence if $\exists r \bullet y_r = False \wedge y_{r+1} = True$, the sequence must be *invalid*.

An assignment to a ladder variable can be unit-propagated appropriately with $p-1$ binary clauses, shown below. Setting a variable to false propagates to all variables following it in the sequence, and setting a variable to true propagates to all variables preceding it, by unit propagation. Hence the following set of clauses forbids all invalid states. These are referred to as the ladder validity clauses.

$$\bigwedge_{p-1}^{i=1} (\neg y_{i+1} \vee y_i) \tag{2}$$

The set of y variables has $|S|$ valid states. Each valid state can be mapped to a single variable in S, such that the variable is assigned true iff $y_1 \ldots y_p$ takes the matching state. In constraint programming terms, this is *channelling* between two representations. This channelling must be propagated in both directions by unit propagation on the clause set. The channelling constraints are as follows.

$$\bigwedge_{|S|}^{i=1} [(y_{i-1} \wedge \neg y_i) \iff x_i]$$

These constraints encode to the following set of clauses (referred to as the channelling clauses).

$$\bigwedge_{|S|}^{i=1} [(\neg y_{i-1} \vee y_i \vee x_i) \wedge (\neg x_i \vee y_{i-1}) \wedge (\neg x_i \vee \neg y_i)] \tag{3}$$

Clauses containing y_0 or $y_{|S|}$ are simplified by unit propagation as if $y_0 = True$ and $y_{|S|} = False$.

The number of channelling clauses plus ladder validity clauses is $O(n)$ (where $n = |S|$), and they are all size 3 or smaller. The number of ladder variables is also $O(n)$. Therefore the time taken to achieve consistency between the two sets by unit propagation must also be $O(n)$. However, while performing search on the

variables in S, the total cost of unit-propagation down one branch of the search tree is also $O(n)$. Since descending the branch involves setting all n variables in S, the mean cost of unit-propagation during search is $O(1)$.

To form an encoding for AMO, we could take the EO encoding described above and add a variable to the set S. The additional variable would indicate that no variables are set true. This is referred to as the ladder AMO encoding.[1]

Another way is to disallow pairs of variables, with the following set of clauses.

$$\bigwedge_{x \in S} \bigwedge_{y \in S \atop y \neq x} (\neg x \vee \neg y) \tag{4}$$

We will refer to this as the *pairwise* AMO encoding. The number of clauses is $O(n^2)$ and the cost of achieving consistency is $O(n)$. This encoding is clearly simpler than the ladder encoding, and much more widely used, but it has worse space complexity. Unit propagation on this encoding achieves the same level of consistency as unit propagation on the ladder AMO encoding.

The combination of the pairwise AMO encoding and the ALO encoding gives a second EO encoding, which is referred to as the pairwise EO encoding.

2.1 AllDifferent from AMO and EO

Throughout we use the same way of encoding of the finite-domain CSP variables into Boolean variables: for a variable v with domain size d and domain D_v, the unary encoding [17] is a set of Boolean variables $x_1^v \ldots x_d^v$ where $\forall i \bullet v \mapsto i \iff x_i^v$. For a variable v contained in the AllDifferent constraint, and a value $i \in D_v$, the Boolean variable x_i^v is assigned true iff $v = i$. Hence we have a two-dimensional table of Boolean variables, shown below for the case where all domains are equal, $d = 4$, and there are 3 variables in the AllDifferent.

		v		
		1	2	3
i	1	x_1^1	x_1^2	x_1^3
	2	x_2^1	x_2^2	x_2^3
	3	x_3^1	x_3^2	x_3^3
	4	x_4^1	x_4^2	x_4^3

Intuitively, an AllDifferent constraint can be formed as follows: each value can be used at most one times, and each variable takes exactly one value. This corresponds to an EO encoding for each column in the table, and an AMO encoding for each row. Using the ladder, this gives us an encoding with $O(nd)$ clauses and $O(nd)$ extra variables. Alternatively, using the pairwise encoding for AMO, and ALO clauses, we have an EO encoding with $O(n^2 d + d^2 n)$ clauses and no extra variables.

[1] In practice the additional variable $x_{|S|}$ is given the last index $|S|$, and by reasoning on the clause set it can be seen that $x_{|S|} = y_{|S|-1}$, so $x_{|S|}$ is redundant and is omitted, along with the relevant channelling clauses (all those that contain $x_{|S|}$).

More formally, the ladder encoding for the AllDifferent constraint can be defined as follows. For each CSP variable v within the AllDifferent constraint, there is one ladder EO structure on the set of Boolean variables representing v: $\forall i \bullet x_i^v$. For each value i where i is in the domain of more than one variable, i.e. $\exists v, w \bullet v \neq w \wedge i \in D_v \wedge i \in D_w$, there is a ladder AMO structure containing the set of Boolean variables $\forall v \bullet x_i^v$. That is, all CSP variables whose domain contains i are represented in the AMO structure.

The pairwise encoding can be defined similarly. In the place of ladder EO structures we have ALO and pairwise AMO combined. In place of the ladder AMO structure we have a pairwise AMO structure.

This scheme also works when the variables do not have equal initial domains. Each value i that is shared between more than one domain must have an AMO structure, and this includes all Boolean variables x_i. Similarly, each CSP variable v has an EO structure covering all Boolean variables x^v.

As a side issue, it is not necessary for the CSP variable to take exactly one value. It is sufficient to use ALO rather than EO. If a solution to the SAT problem encodes a CSP variable with more than one value, this indicates multiple solutions to the underlying CSP. However, with complete SAT solvers, propagation tends to work better if each variable is constrained to take exactly one value (as shown by Kautz et. al. in their comparison between their 2D and 3D encodings [8]).

If all variables share the same domain, and $n = d$, then each value must be used exactly once. Therefore we can use EO encodings for both rows and columns. This gives slightly stronger propagation. We use this optimization in our experimental evaluation, for both ladder and pairwise encodings.

2.2 Propagation in the ladder encoding

It is important that propagation works correctly in the ladder encoding. A change to the unary variables propagates to the ladder variables, then some propagation can occur with the ladder validity clauses. This then propagates back to the unary variables. The following example demonstrates this.

		v		
		1	2	3
i	1	x_1^1	$x_1^2 = False$	x_1^3
	2	x_2^1	$x_2^2 = False$	x_2^3
	3	x_3^1	x_3^2	x_3^3
	4	x_4^1	x_4^2	x_4^3

Let us say that x_2^1 and x_2^2 are already set $False$. For column 2, there are three ladder variables y^1, y^2 and y^3 and two ladder validity clauses, $(y^1 \vee \neg y^2) \wedge (y^2 \vee \neg y^3)$. The following propagation occurs.

- By unit propagation on the channelling clause $(x_1^2 \vee y^1)$, y^1 is set to $True$. (The clause $(x_1^2 \vee y^1)$ is derived from $(\neg y^0 \vee y^1 \vee x_1^2)$ since y^0 is always $True$.)
- By the channelling clause $(\neg y^1 \vee y^2 \vee x_2^2)$, y^2 is set $True$.

None of the unary variables are set. Next, x_3^2 is set $False$ by the search procedure. This shows what happens when all but one of the x^2 variables are set $False$.

- By the channelling clause $(\neg y^2 \vee y^3 \vee x_3^2)$, y^3 is set $True$.
- By the channelling clause $(\neg y^3 \vee x_4^2)$, x_4^2 is set $True$. (This clause is derived from $(\neg y^3 \vee y^4 \vee x_4^2)$, because y^4 is considered to be $False$.)

We now have this situation.

		v		
		1	2	3
i	1	x_1^1	$x_1^2 = False$	x_1^3
	2	x_2^1	$x_2^2 = False$	x_2^3
	3	x_3^1	$x_3^2 = False$	x_3^3
	4	x_4^1	$x_4^2 = True$	x_4^3

Propagation can now happen along row 4. This shows what happens when an x variable is set $True$. There are three ladder variables z^1, z^2 and z^3 and two ladder validity clauses, $(z^1 \vee \neg z^2) \wedge (z^2 \vee \neg z^3)$.

- By unit propagation on the channelling clauses $(\neg x_4^2 \vee z^1) \wedge (\neg x_4^2 \vee \neg z^2)$, z^1 is set to $True$ and z^2 is set to $False$.
- By the ladder validity clause $(z^2 \vee \neg z^3)$, z^3 is set $False$.
- By unit propagation on the channelling clauses $(\neg x_4^1 \vee \neg z^1) \wedge (\neg x_4^3 \vee z^2)$, x_4^1 and x_4^3 are set to $False$.

		v		
		1	2	3
i	1	x_1^1	$x_1^2 = False$	x_1^3
	2	x_2^1	$x_2^2 = False$	x_2^3
	3	x_3^1	$x_3^2 = False$	x_3^3
	4	$x_4^1 = False$	$x_4^2 = True$	$x_4^3 = False$

2.3 Correctness of the ladder encoding

We prove that unit propagation is strong enough to maintain arc-consistency on the pairwise decomposition of the AllDifferent constraint (i.e. its decomposition into $n(n-1)/2$ not-equal constraints). Firstly there are three lemmas about the ladder and channelling, which together show that it functions as a GAC exactly-one constraint. Following that are two theorems which use the lemmas to show the correctness and propagation properties of the ladder encoding. The same two theorems are included for the pairwise encoding, proving that the two encodings are equivalent in propagation. The following proofs refer to variables x_i^v, and to row and column ladders as used in section 2.1.

Lemma 1. *If a set of variables S is channelled to a ladder with the channelling clauses in formula 3, and the ladder validity clauses in formula 2, and more than one of S are set $True$, unit propagation generates the empty clause.*

Proof. Suppose that $p_i, p_j \in S$, $p_i = p_j = True$, $n = |S|$, and the ladder consists of $z_1 ... z_{n-1}$. There are many possibilities for generating an empty clause, of which it suffices to give one:

By the clauses $(\neg p_i \vee z_{i-1}) \wedge (\neg p_i \vee \neg z_i)$, $z_{i-1} = True$ and $z_i = False$. (The end cases $i = 1$ and $i = n$ each lack one of these clauses, and in these cases the ladder becomes entirely $True$ or entirely $False$.) These assignments are then propagated up and down the ladder by the ladder validity clauses, $(z_v \vee \neg z_{v+1})$. So $z_v = True$ for $v \leq i-1$ and $z_v = False$ for $v \geq i$. If $j < i$ in the sequence, the clause $(\neg p_j \vee \neg z_j)$ becomes empty. If $j > i$ in the sequence, the clause $(\neg p_j \vee z_{j-1})$ becomes empty.

Lemma 2. *If a set of variables S is channelled to a ladder with the channelling clauses in formula 3, and the ladder validity clauses in formula 2, and $|S| - 1$ variables $p_i \in S$ are set $False$ with the remaining variable p_j unassigned, then p_j becomes $True$ by unit propagation.*

Proof. Suppose that $n = |S|$ and the ladder consists of $z_1 ... z_{n-1}$. The first variable p_1 is contained in the channelling clause $(z_1 \vee p_1)$ and the last variable in $(\neg z_{n-1} \vee p_n)$. The clause $(\neg z_{i-1} \vee z_i \vee p_i)$ (for every other $False$ variable p_i) simplifies to $(\neg z_{i-1} \vee z_i)$. At least one of p_1 and p_n must be false, so unit propagation begins at the top, bottom or both ends of the ladder and proceeds through the $(\neg z_{i-1} \vee z_i)$ clauses until all the ladder variables are set. So $z_v = True$ for $v \leq j - 1$ and $z_v = False$ for $v \geq j$. p_j is contained in the channelling clause $(\neg z_{j-1} \vee z_j \vee p_j)$ and is thus set $True$. All the channelling and validity clauses are now satisfied.

Lemma 3. *If a set of variables S is channelled to a ladder with the channelling clauses in formula 3, and the ladder validity clauses in formula 2, and a variable $p_i \in S$ is set $True$ with the rest unassigned or $False$, an unassigned variable $p_j \in S$ will be set $False$ by unit propagation.*

Proof. Suppose that $n = |S|$ and the ladder consists of $z_1 ... z_{n-1}$. By the clauses $(\neg p_i \vee z_{i-1}) \wedge (\neg p_i \vee \neg z_i)$, $z_{i-1} = True$ and $z_i = False$. (The end cases $i = 1$ and $i = n$ each lack one of these clauses, and in these cases the ladder becomes entirely $True$ or entirely $False$.) These assignments are then propagated up and down the ladder by the ladder validity clauses, $(z_v \vee \neg z_{v+1})$. So $z_v = True$ for $v \leq i - 1$ and $z_v = False$ for $v \geq i$. If $j < i$ in the sequence, the clause $(\neg p_j \vee \neg z_j)$ causes $p_j = False$. If $j > i$ in the sequence, the clause $(\neg p_j \vee z_{j-1})$ causes $p_j = False$. When all possible variables p_j are set, all the channelling and validity clauses are satisfied.

Theorem 1. *Using the ladder encoding of AllDifferent, when unit propagation stops with no empty clause, the pairwise decomposition not-equal constraints are arc-consistent.*

Proof. Consider the variables x_i^v where v is the CSP variable and i is the value. Suppose we have a set of current domains of the x variables in which no unit propagation is possible, and no domain is empty. Consider any v, w, j such that

there is no support in v for $w = j$. That is, for each possible supporting value $i \in 1..(w-1), (w+1)..d$, $x_i^v = False$.

In this situation the column ladder causes $x_j^v = True$ by lemma 2, then the row ladder causes $x_j^w = False$ by lemma 3. That is, the value j is not in the domain of variable w, and the domains are pairwise arc-consistent as required.

Theorem 2. *Using the ladder encoding of AllDifferent, in a situation where all the pairwise decomposition not-equal constraints are arc-consistent, construct a SAT partial assignment as follows. Variable $x_i^v = True$ if $v = i$ in the CSP, and variable $x_i^v = False$ if i is not in the domain of v. x_i^v is left unassigned if i is in the domain of v but other values remain in the domain of v. Performing unit propagation, some ladder variables y will be set in a valid state. Following this, every clause in the SAT encoding is either satisfied or contains two or more literals and the propagation on ladder variables does not set any x variables.*

Proof. We work by case analysis, first considering the case where $v = i$ in the CSP, corresponding to $x_i^v = True$. In the column ladder, x_j^v will be set $False$ for all $j \neq i$ by lemma 3, and in the proof of lemma 3 it can be seen that both validity and channelling clauses in the column ladder are all satisfied. In the row ladder, again x_i^w will be set $False$ for all $w \neq v$ by lemma 3, and again it can be seen that both validity and channelling clauses in the row ladder are satisfied, and the value i is not present in the domains of variables w as required by arc-consistency. This case also covers the possibility that $x_j^v = False$ for all $j \neq i$, since this implies that $x_i^v = True$ by lemma 2.

The only other case to consider is where $v \neq i$ in the CSP, but at least two values remain in the domain of v. This corresponds to $x_i^v = False$, and there are at least two remaining unset variables in the column. Of the channelling clauses for the column ladder, $(\neg x_i^v \vee y_{i-1}^v) \wedge (\neg x_i^v \vee \neg y_i^v)$ are satisfied and $(\neg y_{i-1}^v \vee y_i^v \vee x_i^v)$ simplifies to $(\neg y_{i-1}^v \vee y_i^v)$. There are three possibilities, (1) if $x_{i-1}^v = False$ or x_{i-1}^v does not exist, then $y_{i-1}^v = True$ hence $y_i^v = True$ and the clause is satisfied, or (2) $x_{i+1}^v = False$ or x_{i+1}^v does not exist, then $y_i^v = False$ hence $y_{i-1}^v = False$ and the clause is satisfied, or (3) the clause remains with two literals. The ladder validity clauses are not relevant here since they cannot become unit. The ladder variables are set in sequence from the top and bottom, but since there are at least two unset x^v variables, not all ladder variables can be set by the clause above. Hence there is no j such that $y_{j-1}^v = True$ and $y_j^v = False$, therefore no unit propagation occurs on the clause $(\neg y_{j-1}^v \vee y_j^v \vee x_j^v)$, and no x variables are set. Other channelling clauses $(\neg x_j^v \vee y_{j-1}^v) \wedge (\neg x_j^v \vee \neg y_j^v)$ are satisfied or non-unit because either $x_j^v = False$ or y_{j-1}^v and y_j^v are both unset. A similar, but simpler, argument applies to the channelling clauses for the row ladder: the argument is simpler because all variables may be set false, and so we do not need to appeal to the existence of two unset variables.

Theorem 3. *Using the pairwise encoding of AllDifferent, formed from the AMO encoding in formula 4 and the ALO in formula 1, when unit propagation stops with no empty clause, the pairwise decomposition not-equal constraints are arc-consistent.*

Proof. Consider the situation described in theorem 1. By the ALO clause $\left(\bigvee_d^{i=1} x_i^v\right)$, $x_j^v = True$. Hence, by the pairwise AMO clause $(\neg x_j^v \vee \neg x_j^w)$, $x_j^w = False$. That is, the value j is not in the domain of variable w, and the domains are pairwise arc-consistent as required.

Theorem 4. *Using the pairwise encoding of AllDifferent, including the AMO encoding in formula 4, in a situation where all the pairwise decomposition not-equal constraints are arc-consistent, construct a SAT partial assignment as follows. Variable $x_i^v = True$ if $v = i$ in the CSP, and variable $x_i^v = False$ if i is not in the domain of v. x_i^v is left unassigned if i is in the domain of v but other values remain in the domain of v. Every clause in the SAT encoding is either satisfied or contains two or more literals.*

Proof. We work by case analysis, first considering the case where $v = i$ in the CSP, corresponding to $x_i^v = True$. By the relevant pairwise clauses $(\neg x_i^v \vee \neg x_i^w)$ where $v \neq w$, $x_i^w = False$, as required by arc-consistency. By the other set of pairwise clauses $(\neg x_i^v \vee \neg x_j^v)$ where $i \neq j$, $x_j^v = False$ as required. The ALO clause for v is satisfied by $x_i^v = True$. The only other case to consider is where $v \neq i$ in the CSP, but at least two values remain in the domain of v. This corresponds to $x_i^v = False$, and there are at least two remaining unset variables in the column. The ALO clause for v is not unit since it contains two unset literals. The pairwise clauses $(\neg x_i^v \vee \neg x_i^w)$ where $v \neq w$, and $(\neg x_i^v \vee \neg x_j^v)$ where $i \neq j$, are all satisfied. No other clauses contain x_i^v.

3 Experimental evaluation

To make a comparison between the encodings presented above, we used the quasigroup completion problem. A quasigroup[2] is an $n \times n$ table of symbols contained in alphabet Σ, where $|\Sigma| = n$. n is the *order* of the quasigroup completion problem. Each row and column of the table contains a permutation of the symbols in Σ. The completion problem is to fill in blank entries in such a table, maintaining the permutation property. It is NP-complete [7].

We generated the instances using the method suggested by Achlioptas et. al., which is to create a random complete quasigroup and punch holes to create a quasigroup with holes (QWH) problem [7]. The method used to generate a random quasigroup is a Markov chain Monte Carlo approach proposed by Jacobson and Matthews [18]. The quasigroups generated this way are uniformly distributed. The second step is to punch a set number of holes in the quasigroup. The positions of the holes are chosen with uniform distribution. This does not give a uniform distribution over all satisfiable QCP instances, because some satisfiable QCP instances can be generated from more than one complete quasigroup. A weakness of this approach is that the problems are all satisfiable.

There exists a trivial algorithm for completing an empty quasigroup, and a full quasigroup is also trivial to complete, hence the difficult region for this

[2] Or, more properly, the multiplication table of a quasigroup, which is a Latin square.

NP-complete problem must lie between these extremes. Achlioptas et. al. show that with randomly placed holes, the computational cost peak for QWH (for an incomplete algorithm (WalkSat) and a complete algorithm (Satz)) occurs when the number of holes is $1.6 \times n^{1.55}$. This corresponds to the backbone covering 50% of the variables. We use only instances from the cost peak (rounding the number of holes to the nearest integer).

To encode these instances into SAT, we use a 3D table of Boolean variables, with size $n \times n \times n$. This is the 2D quasigroup table extended in a third dimension to provide a Boolean variable for each symbol in Σ. For each entry in the quasigroup table, exactly one symbol is required. This is achieved using n^2 EO structures. Similarly, each symbol occurs exactly once on each row of the quasigroup table, and again for the columns, making a total of $3n^2$ EO structures.

Two ways of forming an EO structure were described above: the ladder, and the combined ALO clause and AMO pairwise clauses. We do not mix the two encodings for an instance of QWH, because there is no readily apparent reason why it would be beneficial. From here we will refer to these as *ladder* and *pairwise* encodings. Note that the pairwise encoding is the 3D encoding proposed by Kautz et. al. [8]. Recall that the pairwise EO structure has $O(n^2)$ clauses and the ladder structure has $O(n)$: the pairwise encoding has $O(n^4)$ clauses in total and the ladder has $O(n^3)$. Both encodings have $O(n^3)$ Boolean variables, although the pairwise encoding has n^3 and the ladder encoding has $4n^3 - 3n^2$.

The ladder encoding has a better size bound, but it does not necessarily show an improvement in solution time for instances we can feasibly solve.

3.1 Experimental results

We used a selection of recent SAT solvers to evaluate the encodings: ZChaff [11], SATO version 4.1 [12], and Siege version 4 [13]. Unfortunately, GRASP [15] and 2clseq [14] exceeded the available memory (1 GB) on instances of order 30 and above, so we have omitted them from the evaluation. All these solvers are based on the Davis-Putnam-Logemann-Loveland procedure [9,10]. Since the ladder encoding is designed for unit propagation, we did not consider local search algorithms. We also use a simpler solver to gain some insight into the encodings: BT+lex. This is backtracking with unit propagation and pure literal propagation, with a static (lexicographic) variable ordering and static branch ordering. The internal data structures are described in [16].

All experiments were carried out on a Pentium 4 3.06 GHz machine with 1 GB of RAM, and all runtimes are measured in seconds. The time to the first solution is measured. Sizes of the QWH problems were chosen so that runtimes did not exceed 10 000 seconds.

Figure 1 shows runtimes and node counts for SATO. The line indicates equal runtime. Initially these results are somewhat surprising, because they show that neither encoding is consistently better, even though they have identical propagation characteristics. However, the different clause set and the additional variables of the ladder encoding affect the variable ordering heuristic of SATO in such a

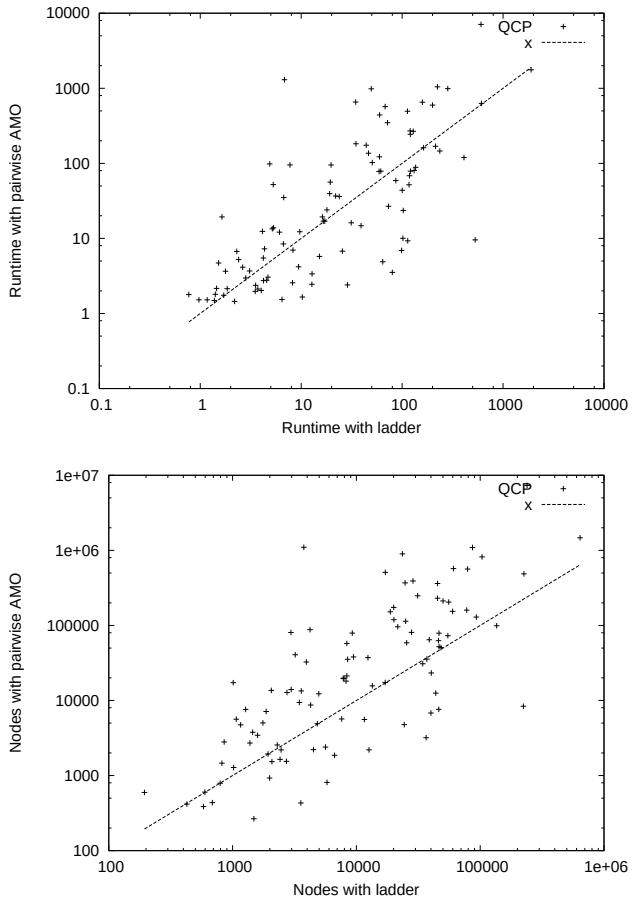

Fig. 1. 100 × order 33 QWH problems, with randomly placed holes and on the phase transition. Runtime and nodes for SATO v4.1.

way that the runtimes are not as well correlated as we expected. The correlation coefficient of the log of the runtimes is 0.74, indicating that the two variables are moderately well correlated. The ladder encoding is better for 60/100 instances, although the median run time for the ladder encoding is 19.29s whereas for the pairwise encoding it is 17.02s. Interestingly, the node counts for the ladder encoding are mostly lower, indicating that the SATO heuristics work better with ladder than pairwise.

ZChaff and Siege can be run for larger instances. Figure 2 shows data for ZChaff with instances of order 35. The correlation coefficient is 0.70 for the log of the runtimes, similar to the previous experiment. 49/100 instances performed better with the ladder encoding. The median run time for the ladder encoding is 25.975s, and 19.795s for the pairwise encoding. The plot of node counts

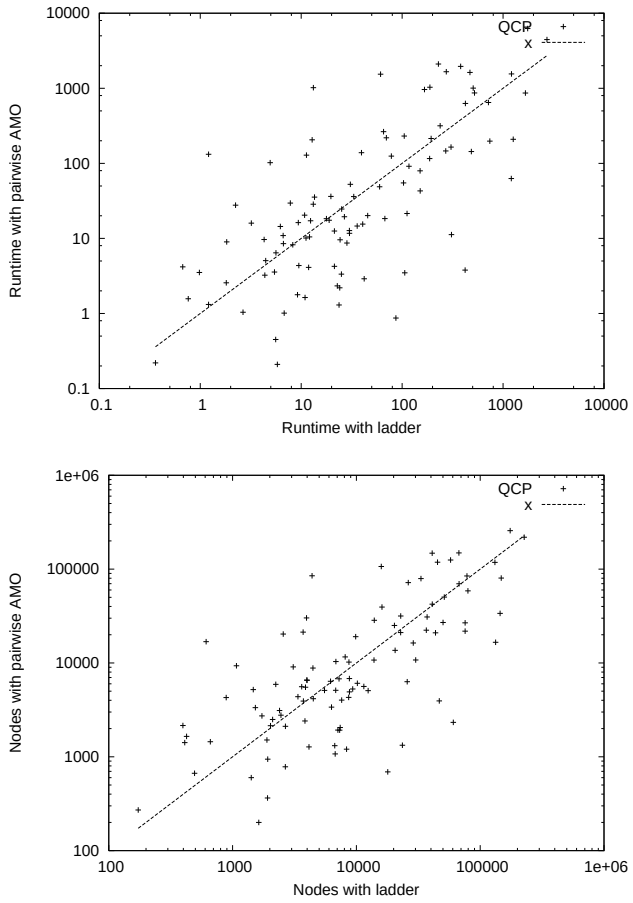

Fig. 2. 100 × order 35 QWH problems, with randomly placed holes and on the phase transition. Runtime and nodes for ZChaff.

shows that neither encoding has a significant advantage in terms of the ZChaff heuristics.

Figure 3 shows data for the Siege solver for the same instances of order 35. The correlation coefficient is 0.78 for the log of the runtimes, which is stronger than previous experiments. Siege seems less well suited to the ladder encoding, with only 12/100 instances performing better with the ladder encoding. The median run time for the ladder encoding is 8.40s, and 1.855s for the pairwise encoding. Although the node count plot is omitted for space reasons, it shows no advantage for either encoding. Therefore the difference in runtime is caused by increased unit propagation cost.

From these results, it appears that SATO is best suited to the ladder encoding. It is also generally the worst-performing solver. For ZChaff and Siege,

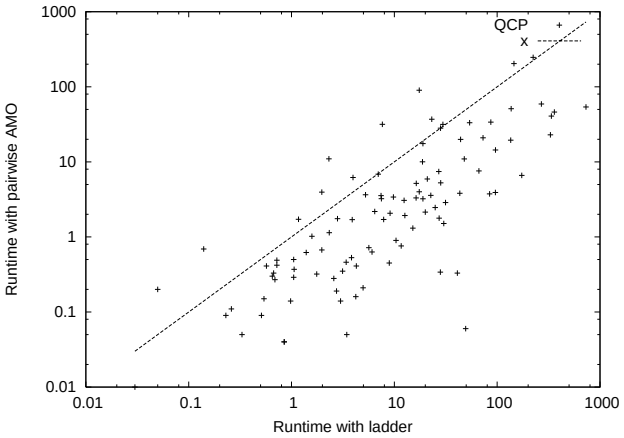

Fig. 3. 100 × order 35 QWH problems, with randomly placed holes and on the phase transition. Runtime for Siege v4.

order (n)	clauses				literals	variables
pairwise encoding						
	AMO (size 2) ($\frac{3}{2}n^3(n-1)$)	ALO (size n) ($3n^2$)	Units	Total		
25	562500	1875	390	564765	1172265	15625
33	1724976	3267	728	1728971	3558491	35937
35	2186625	3675	829	2191129	4502704	42875
ladder encoding						
	Size 2 ($3(3n-2)n^2$)	Size 3 ($3(n-2)n^2$)	Units	Total		
25	136875	43125	390	180390	403515	60625
33	316899	101277	728	418904	938357	140481
35	378525	121275	829	500629	1121704	167825

Table 1. Comparing sizes of the encodings

perhaps we cannot scale high enough to find the benefit of the ladder encoding. Table 1 shows that the ladder encoding is approximately a quarter the size of the pairwise encoding (comparing either the literals or clauses) at order 35. However, it appears that the state of the art solvers can efficiently handle problems with over 5 million literals, and the pairwise encoding allows cheaper unit propagation with Siege.

3.2 Propagation performance

The experiments above do not control the variable and branch ordering, and as a result the ladder and pairwise run times can be several orders of magnitude different, due to different heuristic choices. To remove the effect of the ordering heuristics, it is necessary to fix the variable and branch ordering. If the unary

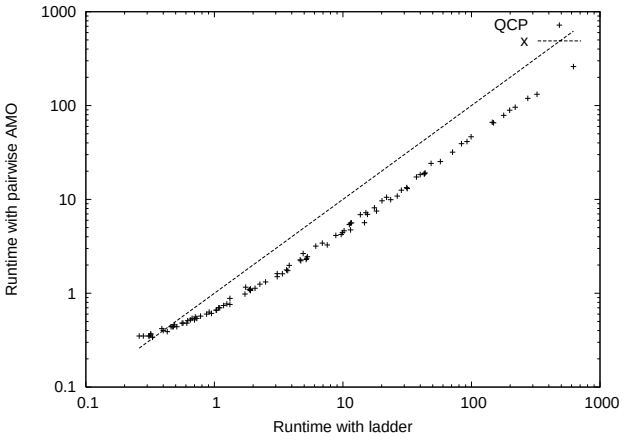

Fig. 4. 100 × order 25 QWH problems, with randomly placed holes and on the phase transition. Runtime for BT+lex.

variables are all set, the additional ladder variables will be set completely by unit propagation, so it is possible to traverse the same search tree with both encodings. This allows us to compare the encodings purely on their unit propagation efficiency.

Algorithms such as conflict backjumping and clause learning can also cause differences in the search tree, so we use a simple backtracking algorithm, with pure literal elimination and unit propagation. Unfortunately, the state-of-the-art SAT solvers do not allow us to use a fixed variable ordering, or to turn off conflict backjumping or clause learning. Therefore we use the QBF solver BT+lex, which has efficient data structures based on literal and clause watching. The implementation is described by Gent et. al. [16]. The fact that BT+lex can accept the QBF problem does not affect its behaviour on a SAT problem, since QBF is a direct generalization of SAT.

Figure 4 shows a comparison between the encodings using BT+lex, and order 25 QWH instances. The pairwise encoding is performing better for most instances. The exception is the easiest instances, which seem to show the overhead of the larger encoding. (Even at this size, the pairwise encoding is over twice the size of the ladder; see table 1.) The median run time for the ladder encoding is 3.77s, compared to 1.875s for the pairwise encoding. Hence at this scale, unit propagation over the pairwise encoding is approximately two times faster with this particular implementation.

To compare BT+lex to a more capable SAT solver, figure 5 shows run times for SATO on the same instances. The median run times here are 0.38s and 0.44s for ladder and pairwise respectively. The lower bounds on the run times are similar to BT+lex, but the upper bound is much lower.

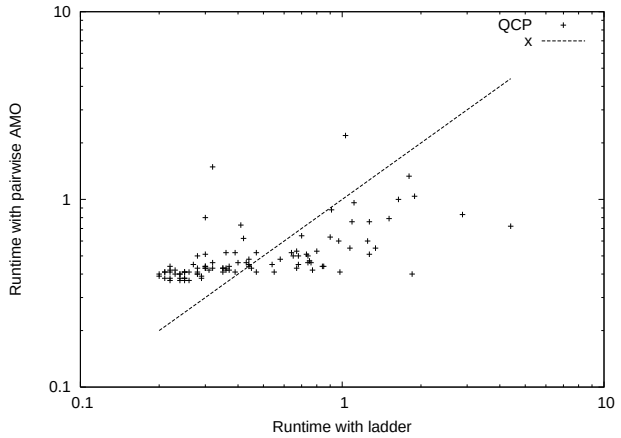

Fig. 5. 100 × order 25 QWH problems, with randomly placed holes and on the phase transition. Runtime for SATO.

4 Conclusions

This article has introduced the ladder encoding for the AllDifferent constraint into SAT. The AllDifferent is commonly used in constraint programming. The new encoding scales well in the number of clauses. However, empirical evaluation on feasibly sized instances of the quasigroup completion problem shows that the new encoding does not perform as well as the pairwise encoding, restricting its use to cases where the formula is so large that size becomes a more important consideration than solution speed. This may be the case for large, easy problems away from the phase transition. Finding encodings of AllDifferent which provide more powerful propagation remains an open research question.

Finally, although the ladder encoding was not very successful here, the ladder structure is a useful encoding trick in other situations [4,5,6]. There are not many such tricks for encoding into SAT at the moment. It would be beneficial to the SAT community to develop and publish more encoding techniques.

Acknowledgments

We thank Andrew Rowley for the use of his BT+lex solver, and Carla Gomes for her random complete quasigroup generator. The first author is supported by a Royal Society of Edinburgh SEELLD Support Fellowship, and the second by an EPSRC Doctoral Training Grant. We also thank the referees for helpful comments.

References

1. Efficient CNF Encoding of Boolean Cardinality Constraints, Olivier Bailleux and Yacine Boufkhad, Proceedings of CP 2003, pages 108-122, 2003.

2. Arc Consistency in SAT, Ian P. Gent, Proceedings of ECAI 2002: the 15th European Conference on Artificial Intelligence, pages 121-125, 2002.
3. The alldifferent Constraint: A Survey, W. J. van Hoeve, 6th Annual workshop of the ERCIM Working Group on Constraints, 2001.
4. A 0/1 encoding of the GACLex constraint for pairs of vectors, Ian Gent, Patrick Prosser and Barbara Smith, in ECAI 2002 workshop W9 Modelling and Solving Problems with Constraints, 2002.
5. SAT Encodings of the Stable Marriage Problem with Ties and Incomplete Lists, Ian P. Gent and Patrick Prosser, Proceedings of 5th International Symposium on Theory and Applications of Satisfiability Testing (SAT2002), 2002.
6. Mapping Problems with Finite-Domain Variables into Problems with Boolean Variables, Carlos Ansótegui and Felip Manyà, To appear in 7th International Conference on Theory and Applications of Satisfiability Testing (SAT04), 2004.
7. Generating Satisfiable Problem Instances, Dimitris Achlioptas, Carla Gomes, Henry Kautz and Bart Selman, Proceedings of 17th National Conference on Artificial Intelligence (AAAI2001), pages 256-261, 2001.
8. Balance and Filtering in Structured Satisfiable Problems, Henry A. Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla P. Gomes, Bart Selman and Mark E. Stickel, Proceedings of IJCAI2001, pages 351-358, 2001.
9. A Computing Procedure for Quantification Theory, Martin Davis, Hilary Putnam, Journal of the ACM (JACM), vol. 7 no. 3, pages 201-215, July 1960.
10. A machine program for theorem-proving, Martin Davis, George Logemann, Donald Loveland, Communications of the ACM, vol. 5 no. 7, pages 394-397, July 1962.
11. Chaff: Engineering an Efficient SAT Solver, Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang and Sharad Malik, Proceedings of the 38th Design Automation Conference (DAC'01), 2001.
12. SATO: an Efficient Propositional Prover, Hantao Zhang, Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI, pages 272-275, 1997.
13. Efficient Algorithms for Clause-learning SAT Solvers, Lawrence Ryan, MSc thesis, Simon Fraser University, February 2004.
14. Enhancing Davis Putnam with Extended Binary Clause Reasoning, F. Bacchus, Proceedings of 18th National Conference on Artificial Intelligence (AAAI-2002), 2002.
15. GRASP: A Search Algorithm for Propositional Satisfiability, J. P. Marques-Silva and K. A. Sakallah, IEEE Transactions on Computers, vol. 48, no. 5, pages 506-521, May 1999.
16. Watched Data Structures for QBF Solvers, Ian Gent, Enrico Giunchiglia, Massimo Narizzano, Andrew Rowley and Armando Tacchella, Proceedings of 6th International Symposium on Theory and Applications of Satisfiability Testing (SAT2003), pages 348-355, 2003.
17. Solving Non-Boolean Satisfiability Problems with Stochastic Local Search, Alan M. Frisch and Timothy J. Peugniez, Proceedings of IJCAI 2001, pages 282-290, 2001.
18. Generating uniformly distributed random latin squares, M. T. Jacobson and P. Matthews, Journal of Combinatorial Designs, vol. 4, no. 6, pages 405-437, 1996.