International Journal of the Physical Sciences Vol. 6 (25), pp. 5937-5950, 23 October, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.880
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

Overlapping Alldiff constraints: SAT versus CSP encoding application to the Sudoku

Eric Monfroy[1,2]*, Broderick Crawford[1,3], Ricardo Soto[3,4] and Fernando Paredes[5]

[1]Universidad Técnica Federico Santa María, Department, de Informática, Av. España 1680, Valparaíso, Chile.
[2]CNRS, LINA, Université de Nantes, France.
[3]Pontificia Universidad Católica de Valparaíso, Chile.
[4]Universidad Autónoma de Chile, Chile.
[5]Escuela de Ingeniería Industrial, Universidad Diego Portales, Santiago, Chile.

Accepted 18 August, 2011

Constraint satisfaction problems (CSP) or Boolean satisfiability problem (SAT) are two well-known paradigms to model and solve combinatorial problems. Modeling and resolution of CSP is often strengthened by global constraints (e.g., Alldiff constraint). This paper highlights two different ways of handling specific structural information: a uniform propagation framework to handle (interleaved) Alldiff constraints with some CSP reduction rules and a SAT encoding of these rules that preserves the reduction properties of CSP. We illustrate our approach on the well-known Sudoku puzzle which presents 27 overlapping Alldiff constraints in its 9 × 9 standard size. We also present some preliminary results we obtained in CHR, GeCode and Zchaff.

Key words: Boolean satisfiability problem (SAT), computer science, decision support, constraint programming, global constraint, automated reasoning.

INTRODUCTION

During the last decades, two closely related communities have focused on the resolution of combinatorial problems.

On one hand, the constraint programming community has focused on the resolution of discrete constraint satisfaction problems (CSP). This paradigm provides the user with a general modeling framework, in which problems are classically expressed by a set of decision variables which values belong to finite integer domains. Then, constraints can be defined to model the relationships that exist between these variables. Concerning the resolution algorithms, complete methods aim at exploring a tree by enumerating variables and reducing the search space using constraint propagation techniques, while incomplete methods can be used to explore this search space, according to specific or more general heuristics (metaheuristics). Again, much work has been achieved to define very efficient propagation algorithms and search heuristics.

On the other hand, the SAT community has developed very efficient algorithms and techniques to handle the seminal Boolean satisfaction problem, which was the first problem to be proved NP-complete (Cook, 1971). The model is very simple, but limits the user to Boolean variables and conjunctive normal form (CNF) propositional formulas to model its problems. The dedicated complete resolution techniques (that is, able to decide if an instance is satisfiable or not) mainly rely on the Davis-Putnam-Logemann-Loveland (DPLL) procedure (Davis et al., 1962). Incomplete algorithms have also been proposed to overcome the problem of size of the instance, mostly based on local search procedures (Selman et al., 1994; Hoos, 1999). In addition, very sophisticated techniques, including symmetries and structure detection, learning techniques, hybrid heuristics, etc., have been proposed in order to build very efficient solvers, which are able to handle huge and very difficult benchmarks, as highlighted during the annual SAT competitions (Hirsch et al., 2010).

Concerning this resolution aspect, the two paradigms

*Corresponding author. E-mail: ericmonfroy@gmail.com.

share some common principles (Bordeaux et al., 2006). Here, we will only focus on complete methods that aim at exploring a tree by enumerating variables (finite domain variables or Boolean ones) and reducing the search space using propagation techniques (constraint propagation or unit propagation).

Now, if we look closer to the modeling facilities, on the CSP side, the identification of typical constraints (so-called global constraints that arise in several real-world problems) has increased the declarative expressiveness. Moreover, the development of very specialized and efficient algorithms has considerably improved the resolution performances. The first example is certainly the Alldiff constraint of Regin (1994) expressing that a set of n variables have all different values. On one hand, this constraint is very useful since it naturally appears in the modeling of many problems, such as timetabling, planning, resource allocation, etc. On the other hand, it is well known that usual constraint propagation techniques are inefficient for this kind of constraint (due to limited domain reduction when decomposing the constraint into $n \times \frac{n-1}{2}$ disequalities) and specific algorithms have been proposed to boost resolution (van Hoeve and Katriel, 2006).

On the SAT side, no such high level modeling feature is offered: the user must translate his problem into propositional logic, involving often, ad-hoc techniques. As a consequence, up-to-date benchmarks (e.g., industrials ones) are often incomprehensible by users and sophisticated preprocessing tools should be used to simplify their structures or detect particular structural properties (e.g., equivalences, implications). Hence, it could be useful to provide a more declarative approach for modeling problems in SAT by adding an intermediate layer to translate high level constraints into propositional formulas. Systematic basic transformations from CSP to SAT have been proposed (Gent, 2002; Walsh, 2000; Gavanelli, 2007; Kasif, 1990; Sinz, 2005) to ensure some consistency properties to the Boolean encodings. Even if some specific relation-ships between variables (e.g., equivalences) are handled specifically by some SAT solvers, global constraints must be transformed into clauses and properties can then be established according to the chosen encodings(Bacchus, 2007; Bailleu and Boufkhad, 2003; Gent and Nightingale, 2004; Marques Silva and Lynce, 2007).

At this time, global constraints have been extensively studied and a lot of them are available in modern solvers, with associated reduction algorithms. Nevertheless, when modeling problems, the user often take into account several global constraints, which may share common variables. Therefore, while the global constraint approach was a first step from basic atomic constraints to more powerful relations (improving then the strength of the local consistencies to reduce the variables domains), the next step would consist of handling efficiently multiple global constraints. Recent works have begun to deal with such combinations of several global constraints (van Hoeve et al., 2008; van Hoeve and Regin, 2006; Regin and Gomes, 2004).

In this paper, we focus on the Alldiff constraint mentioned earlier. From the previous considerations, one may notice that handling sets of distinct variables was often a more general problem and that, in some cases, such Alldiff constraints could be interleaved, leading to a high computational complexity (Elbassioni et al., 2005). For instance, consider two Alldiff constraints on two sets of variables $\mathcal{V}$ and $\mathcal{W}$ such that $\mathcal{V} \cap \mathcal{W} \neq \emptyset$. Many Alldiff constraints overlap in various problems such as Latin squares and Sudoku (Simonis, 2005). Therefore, in this paper, our purpose is twofold:

1. Boolean satisfiability problem: we want to provide, on the CSP solving side, a uniform propagation framework to handle Alldiff constraints and, in particular, interleaved Alldiff. From an operational point of view, we define propagation rules to improve resolution efficiency by taking into account specific properties induced by interleaved Alldiff. Our purpose is to define rules independent from existing solvers and which could be easily integrated in them.
2. We also want to generalize possible encodings of Alldiff and multiple Alldiff constraints in SAT (that is, by a set of CNF formulas). Our purpose is to keep the reduction properties of the previous propagation rules. Therefore, our encodings are fully based on these rules.

Our goal is not to compare the efficiency of CSP reductions versus their SAT encodings (nor to compete with existing solvers), but to generate CSP rules and SAT encodings that are solver independent: no specific features are required for the solvers (just adding new propagators for the cathodic protection (CP) system) and no new global constraints (but it is also possible to add new global constraints) have to be integrated in the modeling language (only the Alldiff is necessary on the CP side). Thus, our techniques can easily be integrated in standard solvers (both CSP and SAT solvers); if one is interested in better efficiency, the solvers can then be improved (based on the CSP rule structures or on the SAT formulas structures) to take advantage of their own facilities.

This paper highlights two different ways of handling specific structural information when faced to multiple Alldiff constraints. On the CSP side, one may use global constraints and study the propagation properties to design new propagation rules or specific algorithms to insure consistency. Note that treating several interleaved Alldiff constraints may perform more reduction (that is, reduce more and more quickly the search space) than using each Alldiff separately. On the SAT side, the resolution process is fixed but one may embed information within encoding itself and take advantage of the structure through unit propagation.

ENCODING CSP VERSUS SAT

CSP basic notions

A CSP (X, C, D) is defined by a set of variables $X = \{x_1, \cdots, x_n\}$ taking their values in their respective domains $D = \{D_1, \cdots, D_n\}$. A constraint $c \in C$ is a relation $c \subseteq D_1 \times \cdots \times D_n$. A tuple $d \in D_1 \times \cdots \times D_n$ is a solution if and only if $\forall c \in C, \quad d \in c$.

Note that we consider C as a set of constraint (equivalent to a conjunction) and that we will use set notations. Usual resolution processes (Apt, 2003; Bordeaux et al., 2006) are based on two main components, such as reduction and search strategies. Search consists in enumerating the possible values of a given variable in order to progressively build a variables assignment and reach a solution. Reduction techniques are added at each node to reduce the search tree (local consistency mechanisms): the idea is to remove values of variables that cannot satisfy the constraints. This approach requires an important computational effort, and performances can be improved by adding more specific techniques, such as efficient constraint propagation algorithms for global constraints. We recall a basic consistency notion (the seminal arc consistency is the binary sub-case of this definition).

Definition 1

Generalized arc consistency (GAC);a constraint c onvariables $[(x]_1, \cdots, x_m)$ is a generalized arc-consistent iff $\forall k \in 1..m, \forall d \in D_k$, $\exists (d_1, \cdots, d_{k-1}, d_{k+1}, \cdots, d_n) \in D_1 \times \cdots \times D_{k-1} \times D_{k+1} \times \cdots \times D_n$, s.t. $[(d]_1, \cdots, d_m) \in c$. A CSP is GAC if all its constraints are GAC.

The reduction/enumeration approach requires an important computational effort and thus, encounters difficulties with large scale problems. Performances can be significantly improved by adding specific techniques such as efficient propagation algorithms, global constraints, etc. Here, we are concerned with global constraints, and more specifically, with reduction rules for overlapping Alldiff.

Modeling the problems: Sudoku as a CSP

The Sudoku (a more constrained Latin square) is a puzzle (e.g., Sudopedia) which can be easily encoded as a CSP: it is played on a 9×9 partially filled grid which must be completed using numbers from 1 to 9 such that the numbers in each row, column and major 33 blocks are different. A $n \times n$ Sudoku puzzle (with $n = m^2$) can be modeled by $3.n$ Alldiff over n^2 variables with domain $[1..n]$:

1. A set of n^2 variables $\mathcal{X} = \{x_{i,j} | i \in [1..n], j \in [1..n]\}$
2. A domain function D such that $\forall x \in \mathcal{X}, D(x) = [1..n]$
3. A set of $3.n$ variables subsets $\mathcal{B} = \{C_1, \cdots, C_n, L_1, \cdots, L_n, B_{1,1}, B_{1,2}, \cdots, B_{m,m}\}$ defined by $\forall x_{i,j} \in \mathcal{X}$, $x_{i,j} \in C_j$, $x_{i,j} \in L_i$, $x_{i,j} \in B_{((i-1) \div m)+1, ((i-1) \div m)+1}$
4. A set of $3.n$ Alldiff constraints $\forall i \in [1..n] Alldiff(C_i)$ and $\forall j \in [1..n] Alldiff(L_j)$ and $\forall k, k' \in [1..m] Alldiff(B_{k,k'})$

Domain reduction rules

Inspired by Apt (2003), we use a formal system to precisely define reduction rules to reduce domains with respect to constraints. We abstract constraint propagation as a transition process over CSPs. A domain reduction rule is of the form:

$$(X, C, D) | \Sigma \rightarrow (X, C, D') | \Sigma'$$

where $D' \subseteq D$ and Σ and Σ' are first order formulas (that is, conditions for the application of the rules) such that $\Sigma \wedge \Sigma'$ is consistent. We canonically generalize $\subseteq$ to sets of domains as $D' \subseteq D$ iff $\forall x \in X, D'_x \subseteq D_x$.. Given a set of variables V, we also denote D_V the union is the set cardinality.

Given a CSP $(X^\dagger k, \quad C^\dagger k, D^\dagger k)$, a transition can be performed to get a reduced CSP $(X^\dagger(k+1), C^\dagger(k+1), D^\dagger(k+1))$ if there is an instance of a rule (that is, a renaming without variables' conflicts):

$$(X^k, C^k, D^k) | \Sigma^k \rightarrow (X^{k+1}, C^{k+1}, D^{k+1}) | \Sigma^{k+1}$$

such that $D^k \vDash \bigwedge_{x \in X} x \in D_x^k \wedge \Sigma^k$ and D^{k+1} is the greatestsubset of D^k such that $D^{k+1} \vDash \bigwedge_{x \in X} x \in D_x^{k+1} \wedge \Sigma^{k+1}$.

In the conclusion of a rule (in Σ), we use the following notations: $d \notin D_x$ means that d can be removed from the domain of the variable x (without loss of solution);

similarly, $d \notin D_V$ means that d can be removed from each domain variable of V and $d1, d2 \notin D_x$ (respectively D_V) is a shortcut for $d1 \notin D_x \wedge d2 \notin D_x$ (resp. $d1 \notin D_V \wedge d2 \notin D_V$).

Since we only consider here rules that do not affect constraints and variables, the sets of variables will be omitted and we highlight the constraints that are required to apply the rules by restricting our notation to $< C, D >$. We will say that $< C, D >$ is GAC if C is GAC with respect to D. For example, a very basic rule to enforce basic node consistency (Apt, 2003) on equality could be:

$$d' \notin D' \_$$

This rule could be applied on $< X = 2, [\![D]\!]_X = \{1,2,3\} >$ with $3 \in D_x, 3 \neq 2$ to obtain $< X = 2, [\![D]\!]_X = \{1,2\} >$, etc.

The transition relation using a rule R is denoted by $< C, D > \xrightarrow{R} < C, D' >$. $\xrightarrow{R*}$ denotes the reflexive transitive closure of $\xrightarrow{R}$. It is clear that $\xrightarrow{R}$ terminates due to the decreasing criterion on domains in the definition of the rules (Apt, 2003). This notion can obviously be extended to sets of rules $\mathcal{R}$. Note also that we require that the result of $\xrightarrow{R*}$ is independent from the order of application of the rules as shown in Apt K R (2003) (this is obvious with the rules that we use). From a practical point of view, it is generally faster to first sequence rules that execute faster.

SAT basic notions

An instance of the SAT problem can be defined by a pair (Ξ, ϕ) where Ξ is a set of Boolean variables $\Xi = \{\xi_1, \cdots, \xi_n\}$ and ϕ is a Boolean formula $\phi: \{0,1\}^n \rightarrow \{0,1\}$. The formula is said to be satisfiable if there exists an assignment $\sigma: \Xi \rightarrow \{0,1\}$ satisfying ϕ and unsatisfiable otherwise. The formula ϕ is in conjunctive normal form (CNF) if it is a conjunction of clauses (a clause is a disjunction of literals and a literal is a variable or its negation).

In order to transform our CSP (X, D, C) into a SAT problem, we must define how the set Ξ is constructed from X and how ϕ is obtained. Concerning the variables, we use the direct encoding as presented in Walsh (2000): $\forall x \in X, \forall d \in D_x, \exists \xi_x^d \in \Xi$ (ξ_x^d is true when x has the value d, false otherwise).

To enforce exactly one value for each variable, we use the next clauses:

$$\bigwedge_{x \in X} \bigvee_{d \in D_x} \xi_x^d$$

and

$$\bigwedge_{x \in X} \bigvee_{d1, d2 \in D_x, d1 \neq d2} [\![(\neg \xi]\!]_x^{d1} \vee \neg \xi_x^{d2})$$

Given a constraint $c \in C$, one may add for all tuples $d \notin c$, a clause recording this nogood value or use other encodings based on the valid tuples of the constraint (Bacchus, 2007). One may remark that it can be very expensive and it is strongly related to the definition of the constraint itself. Therefore, as mentioned in the introduction, several work have addressed the encodings of usual global constraints into SAT (Bailleu and Boufkhad, 2003), 12 or (Marques Silva and Lynce, 2007). Here, our purpose is to define uniform transformation rules for handling multiple Alldiff constraints, which are often involved in many problems.

From the resolution point of view, complete SAT solvers are basically based on a branching rule that assign a truth value to a selected variable and unit propagation (UP) which allows to propagate unit clauses in the current formula (Bordeaux et al., 2006). This principle is very close to the propagation of constraints achieved by reduction rules to enforce consistency. Therefore, we will study the two encodings CSP and SAT from this consistency point of view. According to Walsh (2000) and Bacchus (2007), we say that a SAT encoding preserves a consistency iff all variables assigned to false by unit propagation have their corresponding values eliminated by enforcing GAC.

More formally, given a constraint c, UP leads to a unit clause $[\![(\neg \xi]\!]_x^d)$ iff d is not GAC with c (d is removed from D_x by enforcing GAC) and if c is unsatisfiable then UP generates the empty clause (enforcing GAC leads to an empty domain).

ALLDIFF CONSTRAINTS: REDUCTION RULES AND TRANSFORMATIONS

In the following, we classically note $Alldiff(V)$ the Alldiff constraint on a subset of variables V, which semantically corresponds to the conjunction of $n * \frac{n-1}{2}$ pairwise disequality constraints $\bigwedge_{x_i, x_j \in V, i \neq j} x_i \neq x_j$.

1,2 3,4	3,4 5	2,3	6,7 8	1,2 3,7	2,3	2,4 8,9	1,5 6,7,9	2,4 5,8,9

Initial row

1,4	4,5	2,3	6,7 8	1,7	2,3	4 8,9	1,5 6,7,9	4,5 8,9

After reduction with [O2]

Figure 1. Application of [O3] on a Sudoku or Latin square row.

1,2 3,4	3,4 5	2,3	6,7 8	1,2 3,7	1,2	1,3	1,5 6,7,9	2,4 5,8,9

Initial row

4	4,5	2,3	6,7 8	7	2,3	1,3	5,6 7,9	4,5 8,9

After reduction with [O3]

Figure 2. Application of [O3] on a Sudoku or Latin square row.

Single Alldiff constraint

We first reformulate a well-known consistency property described (Regin, 1994; van Hoeve and Katriel, 2006) with respect to the number of values remaining in the domain of the variables. This case corresponds of course to the fact that if a variable has been assigned, then the corresponding value must be discarded from other domains.

[O1]:

$$< C \wedge Alldiff(V), D > \mid x \in V \wedge D_x = \{d\} \rightarrow < C \wedge Alldiff(V), D' > \mid d \notin D'(V \setminus \{x\})$$

Property 1

If $< Alldiff(V), D > \xrightarrow{[O1]*} < Alldiff(V), D' >$ then the corresponding conjunction $\bigwedge_{x_i, x_j \in V, i \neq j} x_i \neq x_j$ is GAC with respect to $< D' >$. Note that enforcing GAC on the disequalities with **[O1]** reduces less the domains than enforcing GAC on the global Alldiff constraint.

This rule can be generalized when considering a subset V' of m variables with m possible values, $1 \leq m \leq (\#V - 1)$:

[Om]:

$$< C \wedge Alldiff(V), D > \mid V' \subseteq V \wedge D_{V'} = \{d_1, \cdots, d_m\} \rightarrow < C \wedge Alldiff(V), D' > \mid d_1, \cdots, d_m \notin D'_{V \setminus V'}$$

Consider $m = 2$, and that two variables of an Alldiff only have the same two possible values. Then, it is trivial to see that these two values cannot belong to the domains of the other variables (Figure 1).

Figure 2 shows an application of **[O3]**: the values 1, 2 and 3 only appear in 3 cells, thus they can be removed from the other cells.

Property 2

If $< Alldiff(V), D > \xrightarrow{[Om]*} < Alldiff(V), D' >$ for all m so that $1 \leq m \leq (\#V - 1)$, then $< Alldiff(V), D' >$ has the GAC property.

The proof can be obtained from Regin (1994). Now, the Alldiff constraints can be translated in SAT, by encoding **[O1]** for a variable x with a set of $\#V * (\#V - 1)$ CNF clauses:

[SAT − O1:]

$$\wedge_{x \in V} \wedge_{y \in V \setminus \{x\}} \left([\neg \xi_x^d] \vee \left(\vee_{f \in D_x \setminus \{d\}} \xi_y^f \right) \vee [\neg \xi_y^d] \right)$$

This representation preserves GAC. Indeed, if $\neg \xi_x^d$ is false (that is, when the variable x is valued to d) and $(\vee_{f \in D_x x1 \setminus \{d\}} \xi_{x1}^f)$ is false (that is, when the variable $x1$ is valued to d) then $\neg \xi_{x2}^d$ must be true to satisfy the clause ($x2$ cannot be valued to d). Generalized to a subset V' of m variables $\{x_1, \cdots, x_m\}$ with m possible values $\{d_1, \cdots, d_m\}$, $1 \leq m \leq (\#V - 1)$, the $\#(V \setminus V') * m * (m + 1)$ clauses are:

$$\wedge_{x \in V \setminus V'} \wedge_{d=1}^{m} \wedge_{f_1=1}^{m} \cdots \wedge_{f_m=1}^{m} \left[\left(\vee_{s=1}^{m} \neg \xi_x^{d_s} \right) \vee \left(\vee_{i=1}^{m} \vee_{f \in D_{x_i} \setminus \{f_1, \cdots, f_m\}} \xi_{x_i}^{f} \right) \vee \neg \xi_{x_i}^{d} \right]$$

Property 3

$\bigcup_{1 \leq m \leq (\#V-1)} [SAT - Om]$ preserves the GAC property.

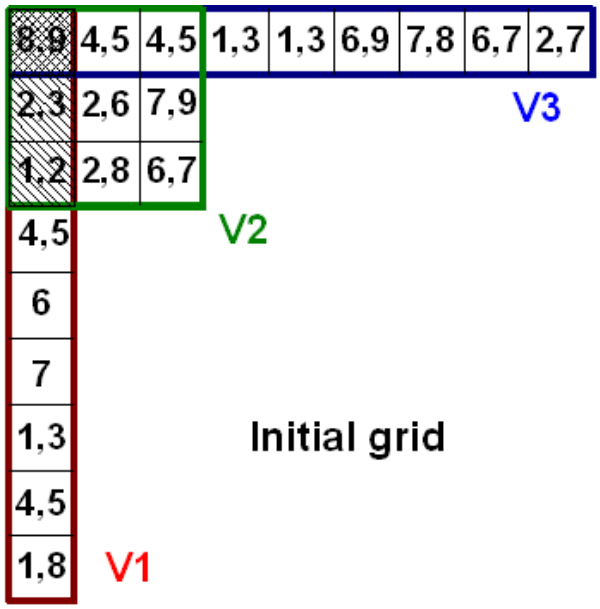

Figure 3. Initial grid.

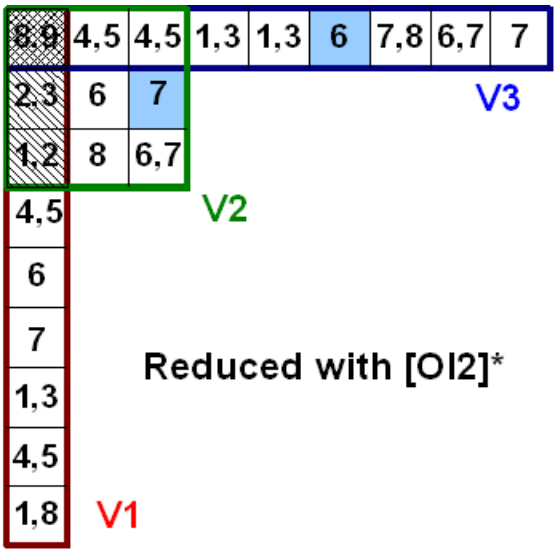

Figure 4. After several reductions by $[OI2]$.

Proof: As mentioned earlier, our transformation is directly based on consistency rules and therefore Property 2 remains valid for the SAT encoding. This can be justified through the propositional rewriting of the direct encoding of $[Om]$ into $[SAT - Om]$ (not given here for lack of space).

MULTIPLE OVERLAPPING ALLDIFF CONSTRAINTS

In the presence of several overlapping Alldiff constraints, specific local consistency properties can be enforced according to the number of common variables, their possible values and the number of overlaps. To simplify, we consider Alldiff constraints $Alldiff(V)$ such that $\#V = \#D_1V$. This restriction could be weakened but it is generally needed in classical problems (especially for Sudoku or Latin squares). We now study typical connections between multiple Alldiff. Therefore, we consider simultaneously several constraints in the design of new rules to achieve GAC as it was the case when considering a global Alldiff instead of a conjunction of pair wise disequalities to improve reduction.

Several Alldiff connected by one intersection

This is a simple propagation rule: if a value appears in variables of the intersection of two Alldiff, and that it does not appear in the rest of one of the Alldiff, then it can be safely removed from the other variables domains of the second Alldiff.

$[OI2]$:

$$d \in D_{V_1 \cap V_2} \wedge d \notin D_{V_2 \setminus V_1} \rightarrow d \notin D'_{V_1 \setminus V_2}$$

Figures 3 and 4 show an example of applications of $[OI2]$: Figure 3 is the intial grid and Figure 4 the grid after reduction by several applications of $[OI2]$. $[OI2]$ is coded in SAT as $\#D_1(V_1 1 \cap V_1 2) * \#(V_1 1 \setminus V_1 2) * \#(V_1 1 \setminus V_1 2)$ clauses:

$[SAT - OI2:]$

$$\bigwedge_{d \in D_{V_1 \cap V_2}} \bigwedge_{x_1 \in V_1 \cap V_2} \bigwedge_{x_2 \in V_1 \setminus V_2} \bigvee_{x_3 \in V_2 \setminus V_1} (\neg s^d_{x_1} \vee s^d_{x_2} \vee \neg s^d_{x_3})$$

$[OI2]$ can be extended to $[OI\,m]$ to handle $m\,(m \geq 2)$ Alldiff constraints connected by one intersection. Let denote by V the set of variables appearing in the common intersection: $V = \bigcap_{i=1}^{m} V_i$

$[OIm]$:

$$d \in D_V \wedge d \notin D_{V_1 \setminus V} \rightarrow d \notin \bigcup_{i=2}^{m} D'_{V_i \setminus V}$$

Figures 3 and 5 show an example of application of $[OIm]$ (with $m = 3$): Figure 3 is the initial grid, and

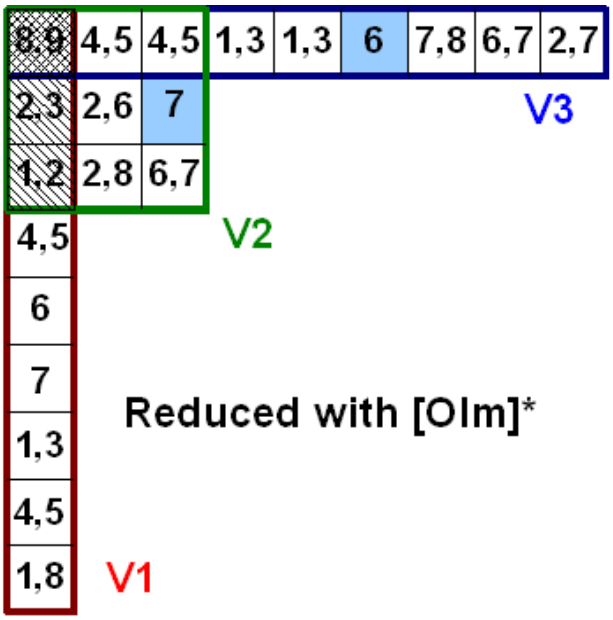

Figure 5. After reductions by $[Olm]$.

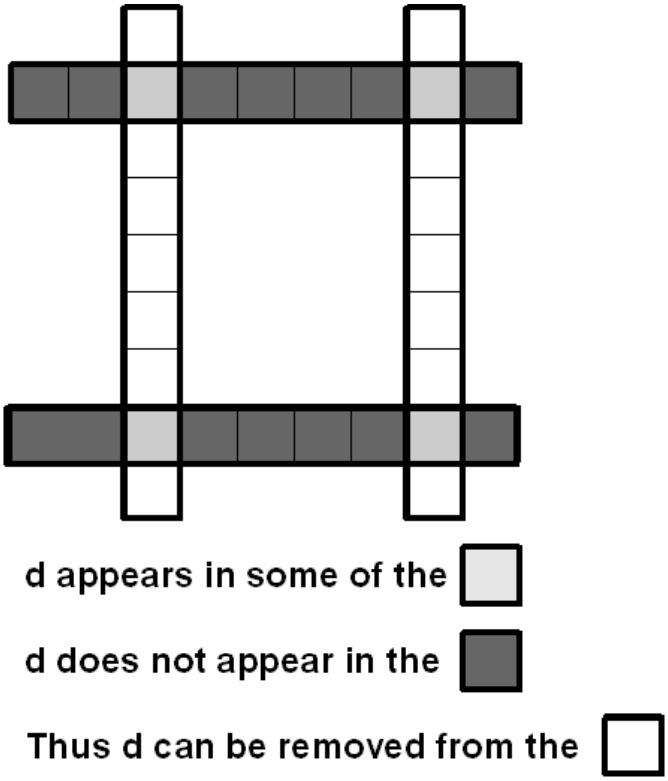

Figure 6. A pattern for applying $[SI4.4]$.

Figure 5 the grid after reduction by $[Olm]$.

Note that this rule can be implicitly applied to the different symmetrical possible orderings of the Alldiff.

$[Olm]$ is translated in SAT as $\#D_1V * \#V * \sum_1(i = 2)^{\lceil m \rceil}(\#(V_1 i \backslash V))$ clauses:

$$[SAT - Olm:]$$

$$\bigwedge_{d \in D_V} \bigwedge_{x_1 \in V_1} \bigwedge_{i=2}^{m} \bigwedge_{x_2 \in V_{i \backslash V}} \bigvee_{x_2 \in V_{1 \backslash V}} (\neg \xi_{x_1}^{d} \vee \xi_{x_2}^{d} \vee \neg \xi_{x_2}^{d})$$

Property 4

Consider $m > 2$ Alldiff with a non empty intersection. Given $< C, D > \xrightarrow{[Olm]*} < C, D' >$ and $< C, D > \xrightarrow{[OI2]*} < C, D'' >$, then $D'' \subseteq D'$.

The proof is straightforward. We illustrate it on Figures 3, 4 and 5. Consider the application of $[Olm]$: 9 is in the intersection of V_1, V_2 and V_3 and not in the rest of V_1; thus, 9 can be removed safely from V_2 and V_3 (except from the intersection of the 3 Alldiff); none other application of $[Olm]$ is possible, leading to the third grid. Now, consider the application of $[OI2]$ on the initial grid: first between of V_1 and V_2; 9 is in the intersection of V_1 and V_2 and not in the rest of V_1; thus, 9 can be removed safely from V_2; the same for 2; applying [OI2] on V_1 and V_3 removes 9 from the rest of V_3; applying $[OI2]$ on V_3 and V_2 does not perform any effective reduction; this leads to the second grid which is smaller than the third one.

Although one could argue that $[Olm]$ is useless (Property 4) in terms of reduction, in practice $[Olm]$ can be interesting in terms of the number of rules to be applied. Moreover, $[Olm]$ can be scheduled before $[OI2]$ to reduce the CSP at low cost.

Several Alldiff connected by several intersections

We first consider 4 Alldiff having four non-empty intersections two by two (Figure 6). V_{ij} (respectively $V_{i,j}$) denotes $V_i \cup V_j$ (respectively $V_i \cap V_j$). V now denotes the union of the four intersections: $V = V_{1,2} \cup V_{2,3} \cup V_{3,4} \cup V_{1,4}$.

$[SI4.4]$:

$$V_{1,2} \neq \emptyset \wedge V_{2,3} \neq \emptyset \wedge V_{3,4} \neq \emptyset \wedge V_{1,4} \neq \emptyset \wedge d \in D_V \wedge d \notin D_{V_{12 \backslash V_{34}}}$$
$$\rightarrow d \notin D'_{V_{24 \backslash V_{13}}}$$

d must at least be an element of 2 opposite intersections

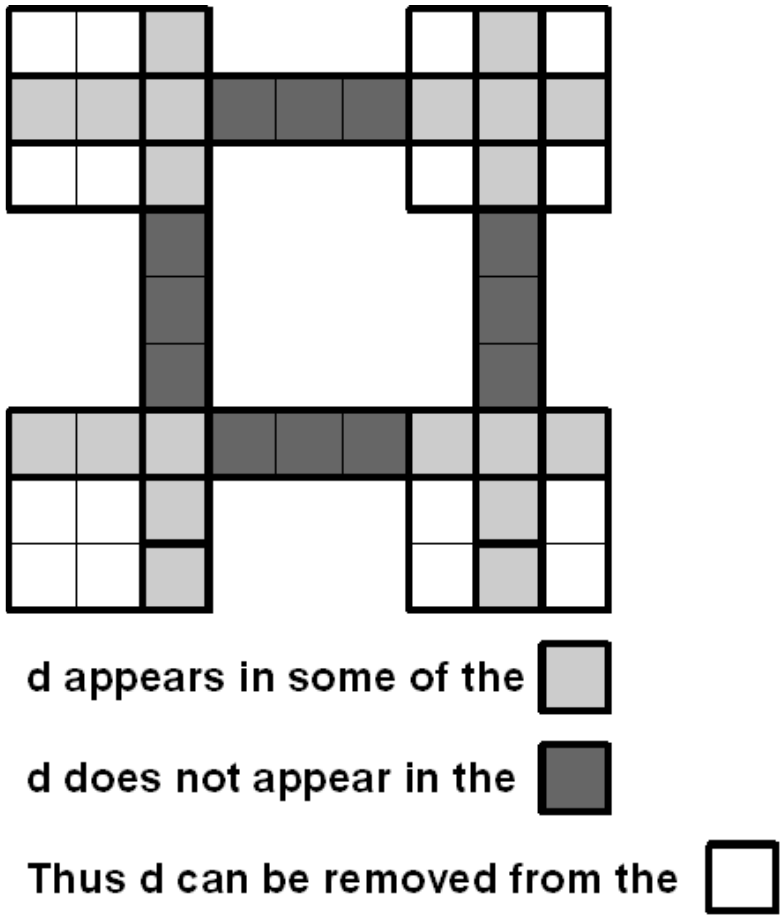

Figure 7. A pattern for applying $[SI2m.2m]$ with $m = 4$.

(at least $d \in V_{1,2} \cap V_{3,4}$ or $d \in V_{2,3} \cap V_{1,4}$) otherwise, the problem has no solution. Our rule is still valid in this case, and its reduction will help showing that there is no solution.

Translated in SAT, we obtain $\#D_1V * \#V\#(V_1 24 \backslash V_1 13)$ clauses with $V_{1,2} \neq \emptyset \wedge V_{2,3} \neq \emptyset \wedge V_{3,4} \neq \emptyset \wedge V_{1,4} \neq \emptyset$:

$$[SAT - SI4.4]:$$

$$\bigwedge_{d \in D_V} \bigwedge_{x_1 \in V} \bigwedge_{x_3 \in V_{24} \backslash V_{13}} \bigvee_{x_2 \in V_{23} \backslash V_{24}} (\neg \xi^d_{x_1} \vee \xi^d_{x_2} \vee \neg \xi^d_{x_3})$$

This rule can be generalized to a ring of $2m$ Alldiff with $2m$ non-empty intersections (Figure 7 for a case of 8 Alldiff). Let V be the union of the variables of the $2m$ intersections: $V = \bigcup_{i=1}^{2m} (V_i \cap V_{(i \bmod 2m)+1})$. V_{odd} (respectively V_{even}) represents the union of the V_k such that k is odd (respectively even):

$\bigcup_{i=0}^{m-1} (V_{(2i+1)})$ (respectively $\bigcup_{i=1}^{m} (V_{(2i)})$.

$$[SI2m.2m]:$$

$$\bigwedge_{i=1}^{2m} (V_i \cap V_{(i \bmod 2m)+1} \neq \emptyset) \wedge d \in D_V \wedge d \in D_{V_{odd} \backslash V}$$

$$\rightarrow$$

$$d \notin D'_{V_{even} \backslash V_{\square\square}}$$

These are $\#D_1V * \#V\#(V_1 even \backslash V)$ SAT clauses, such that $\bigwedge_{i=1}^{2m} (V_i \cap V_{(i \bmod 2m)+1} \neq \emptyset)$:

$$[SAT - SI2m.2m]:$$

$$\bigwedge_{d \in D_V} \bigwedge_{x_1 \in V} \bigwedge_{x_3 \in V_{even} \backslash V} \bigvee_{x_2 \in V_{odd} \backslash V} (\neg \xi^d_{x_1} \vee \xi^d_{x_2} \vee \neg \xi^d_{x_3})$$

The reduction we obtain by applying rules for a single

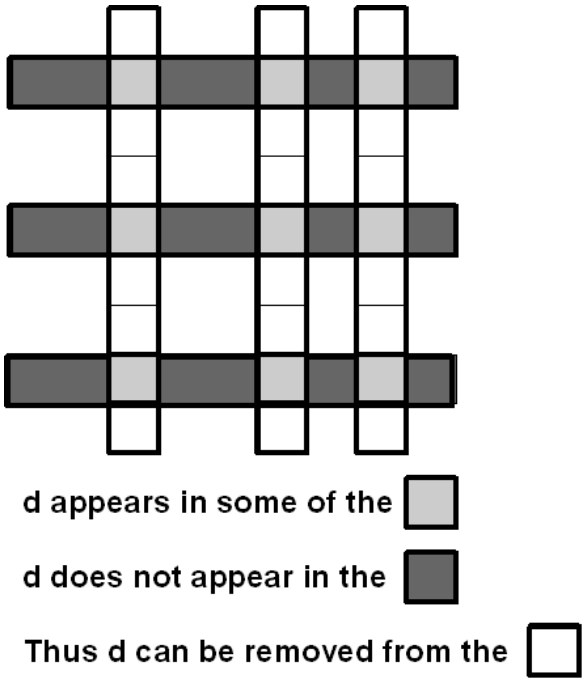

Figure 8. A pattern for applying $[SI6.9]$.

Alldiff and several Alldiff is stronger than enforcing GAC.

Property 5

Given a conjunction of constraints $\bigwedge_{i=1}^{k} Alldiff(V_i)$ and a set of domains D. Given 2 sets of rules $R' \subseteq \bigcup_{i=1}^{\max_1} \{\#V_i\} = \{[Oi]\}$ and $R \subseteq \{[OI2], \cdots, [OIm], [SI4.4], \cdots, [SI2m.2m]\}$.

Consider $< C, D > \xrightarrow{R'*} < C, D' >$ and $< C, D > \xrightarrow{R' \cup R*} < C, D'' >$, then $< C, D' >$ and $< C, D'' >$ are GAC, and moreover $D'' \subseteq D'$.

$\bigcup_{i=1}^{\max_1} \{\#V_i\} \{[Oi]\}$

The proof is based on the fact that already enforces GAC and that the $[OIm]$, $[SI2m.2m]$ preserve GAC. The proof is similar to the proof of $[SAT - Om] \Leftrightarrow [Om]$ of Property 3.

Some more rules

We now give some more rules that are more problem specific, or represent some well-known rules for solving the Sudoku problem. We do not give the SAT translation, as it is more or less automatic, following the technique shown earlier. We exemplify the rules with the Sudoku problem.

$[SI2m.m^2]$: $[SI2m.2m]$ (presented earlier) extends $[SI4.4]$ in terms of ring. Now, we present $[SI2m.m^2]$ which extends the matrix notion of $[SI4.4]$: a set of m Alldiff (e.g., rows for a Sudoku) that cross another set of m Alldiff (e.g., columns of a Sudoku), forming m^2 intersections (Figure 8 contains an example, with $m = 3$, where the rule $[SI6.9]$ could be applied).

Consider V as the union of the variables of the m^2 intersections $V - \bigcup_{i=1..m, j=m+1..2.m} V_{i,j}$; V_r as the union of the first m Alldiff (e.g., rows) deprived of the intersections with the m other Alldiff (e.g., columns): $V_1r = \mathbf{U_1}(i = 1)^{\uparrow} m \setminus [V_1i(\mathbf{U_1}(j = m + 1)^{\uparrow}(2.m) \setminus V_1(i, j))]$ and V_c as the union of the second set of m Alldiff (e.g., columns) deprived of the intersections with the m first Alldiff (e.g., rows): $V_1c = \mathbf{U_1}(j = m + 1)\hat{}(2.m) \setminus [V_1j(\mathbf{U_1}(i = 1)^{\uparrow} m \setminus V_1(i, j))]$.

$[SI2m.m^2]$:

$$< C \wedge \bigwedge_{i=1}^{2m} Alldiff(V_i), D > \Big| d \in D_{V_r} \wedge d \notin D_{V_c} \rightarrow < C \wedge \bigwedge_{i=1}^{2m} Alldiff(V_i), D' > \Big| d \in D'_{V_r}$$

Proof

Consider m Alldiff (rows) and m other Alldiff (columns), and d occurs in the intersections rows/columns but not in rows. To cover the rows, one d from one intersection for each row will be used. Moreover, these d will not appear in the same column. Since we have m rows and m columns, the d of the intersection will be sufficient, and thus, d can be removed from the columns, except at the intersections with the rows.

Note that for $m = 3$, $[SI2m.m^2]$ represents the Swordfish technique for 9×9 Sudoku grids (Figure 8). Similarly to $[SI4.4]$ we can see some conditions that make the problem has or no solution: e.g., in the case of $[SI6.9]$, d must be found in at least 2 intersections of each row and at least 2 intersections of each column; otherwise, if $d \notin D_{V_r}$, the problem has no solution.

Sudoku technique: XY-Wing

This technique considers 4 Alldiff constraints (2 rows and

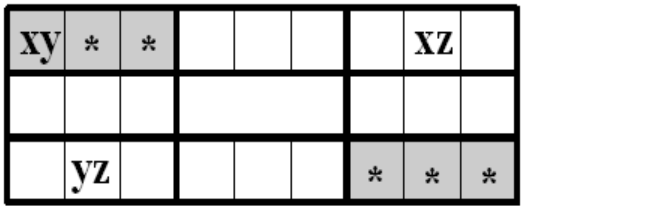

Figure 9. A pattern for XY-Wing.

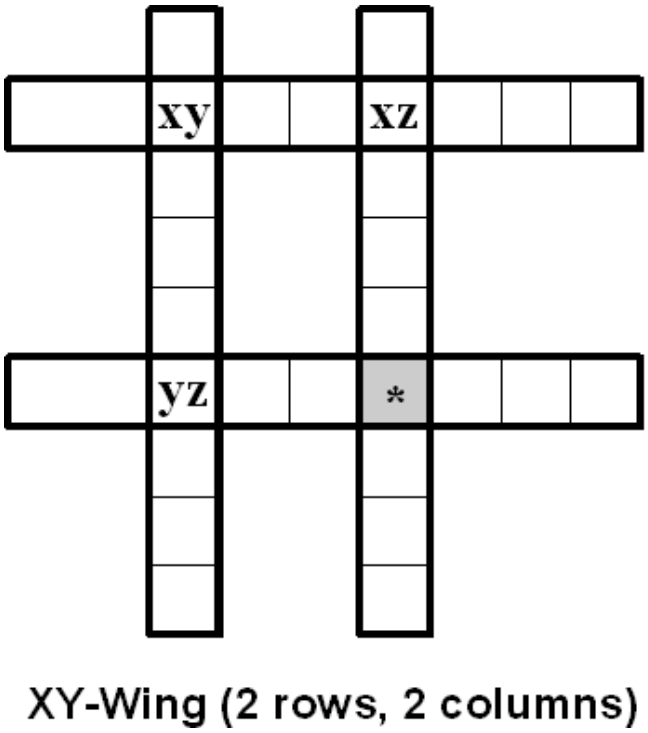

Figure 10. A pattern for XY-Wing.

2 columns, 2 blocks and 2 rows, or 2 blocks and 2 columns) that form a "ring", that is, the four 2 by 2 intersections are non-empty.

The XY-Wing technique (Figures 9 and 10) can be formalized by the following rule:

$$[R.XY-W]:$$

$$\langle C \wedge \bigwedge_{i=1}^{4} Alldiff(V_i), D \rangle$$

$$|V_{1,2}| \neq \emptyset \wedge |V_{2,3}| \neq \emptyset \wedge |V_{3,4}| \neq \emptyset \wedge |V_{4,1}| \neq \emptyset \wedge v_0 \in V_{1,2} \wedge v_1 \in V_{1,2} \wedge v_1 \in V_{1,2} \wedge v_2 \in V_{1,2} \wedge d_3 \in D'_{V_{1,2} \cup V_{3,4}})$$

For 2 blocks and 2 rows (or columns), in the best case this rule removes 5 values for a 9×9 Sudoku. For 2 columns and 2 blocks, this rule removes at most one value for a $n \times n$ Sudoku, or a $n \times n$ Latin square.

Sudoku technique: XYZ-Wing

This is a variant of the XY-Wing, based on the intersection of a block and a column/row: if 3 values x, y and z appear in a variable of the intersection, and the column/row contains a variable with the 2 values x and z, and that the block contains a variable with the 2 values y and z, then, z can be removed from the other variables of the intersection (Figure 11).

$$[R.XYZ-W]:$$

$$\langle C \wedge \bigwedge_{i=1}^{3} Alldiff(V_i), D \rangle | v_0 \in V_{1,2} \wedge v_1 \in V_{1,1} \wedge v_2 \in V_{2,1} \wedge \{d_1, d_2, d_3\} = D_{v_0} \wedge \{d_1, d_2\} = D_{v_1} \wedge \{d_2, d_3\} = D_{v_2} \rightarrow d_3 \in D'_{V_{1,2} \setminus V_0})$$

EVALUATION

To evaluate these rules, we use them with the SAT and CSP approaches on the Sudoku problem. The Sudoku is a well-known puzzle (e.g., Sudopedia) which can be easily encoded as a constraint satisfaction problem: it is generally played on a 9×9 partially filled grid, which must be completed using numbers from 1 to 9 such that the numbers in each row, column and major 3×3 blocks are different.

More precisely, the $n \times n$ Sudoku puzzle (with $n = m^2$) can be modeled by $3.n$ Alldiff constraints over n^2 variables with domain $[1..n]$, as shown earlier in " ENCODING CSP VERSUS SAT" .

The Sudoku puzzle is a special case (that is, more constrained) of Latin squares, which do not require the notion of blocks nor the Alldiff constraints over the blocks.

SAT approach

We now compute the size of the SAT model for a Sudoku of size n by computing the number of generated clauses by each rule (Table 1).

In the following, we consider 9×9 grids. To encode such a Sudoku of size 9, the minimum number of clauses is 20493 (definition of the variables and $[SAT-01]$), whereas encoding all the rules generates approximately 886×10^9 clauses. This increase of the number of clauses is mainly due to $[SAT-Om]$ (Figure 12).

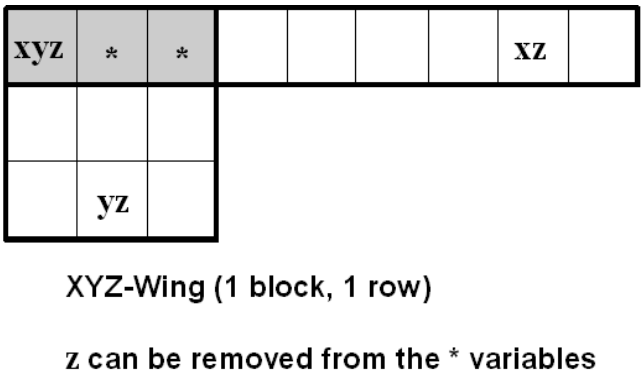

Figure 11. A pattern for XYZ-Wing.

Table 1. The size of the SAT model for a Sudoku of size n.

	Number of clauses	Complexity
Definition of the variables	$n^2 + \frac{n^3(n-1)}{2}$	$O(n^4)$
$[SAT - Om] \forall m \in \{1..n-1\}$	$3n \sum_{m=1}^{n-1} \left(\binom{m}{n}^2 (n-m) m^{m+1} \right)$	$O(n^{3n+3})$
$[SAT - O1m] \forall m \in \{2,3\}$	$n^3(9n - 4\sqrt{n} - 5)$	$O(n^4)$
$[SAT - SI2m.2m]$ with $m = 2$	$6n^6 - 32n^5 + \sqrt{n}(12n^4 - 12n^3) + 34n^4 - 8n^3$	$O(n^6)$

Thus, it is not practicable to generate Sudoku problems in SAT including initially all the rules. To observe the impact of these rules on the behavior of SAT solvers, we run Zchaff (Moskewicz et al., 2001) on 9 Sudoku grids coded with:

1. Definition of the variables $+[SAT - O1]$
2. Definition of the variables $+[SAT - O1] + [SAT - O2]$
3. Definition of the variables $+[SAT - O1] + [SAT - OI2]$
4. Definition of the variables $+[SAT - O1] + [SAT - O2] + [SAT - OI2]$

Figure 13 illustrates the encoding impact on the behavior of Zchaff. For some instances, $[SAT - O2]$ improves the results. We suppose that the performances using the other $[SAT - Om]$ could be better. These results confirm Property 5 because $[SAT - OI2]$ does not improve the behavior if $[SAT - O2]$ is present. We can observe that the worst performances are obtained when $[SAT - OI2]$ is combined to the basic definition of the problem. This rule is probably rarely used, but its clauses may disrupt the heuristics. Nevertheless, the costly but powerful could be added dynamically, during the resolution process in order to boost unit propagation.

CSP approach

From a CSP point of view, we have few rules to manage. However, the combinatoric/complexity is pushed in the rule application, and more especially in the matching: the head of the rule must be tried with all possible configurations of the constraints, and the guard must be tested. Implementing our rules in CHR (SWI-Prolog version) as propagation rules is straightforward but a generic implementation is rather inefficient. The matching of the head of the rule is too weak, and a huge number of conditions have to be tested in the guard. We thus specialized the CHR rules for arrays of variables, which is thus, well suited for problems such as Latin squares and Sudoku. The rules are also scheduled in order to first apply less complex rules, that is, the rules that are faster to apply (strong matching condition and few conditions in the guard), and which have more chance to effectively reduce the CSP. However, we are still working on the implementation to improve the matching. For example, by particularizing $[O7]$ to the Sudoku, we obtained a speed up of up to 1000 for some Sudokus. We also implemented some rules as new propagators in GeCode (Gecode, 2009). The preliminary results are promising,

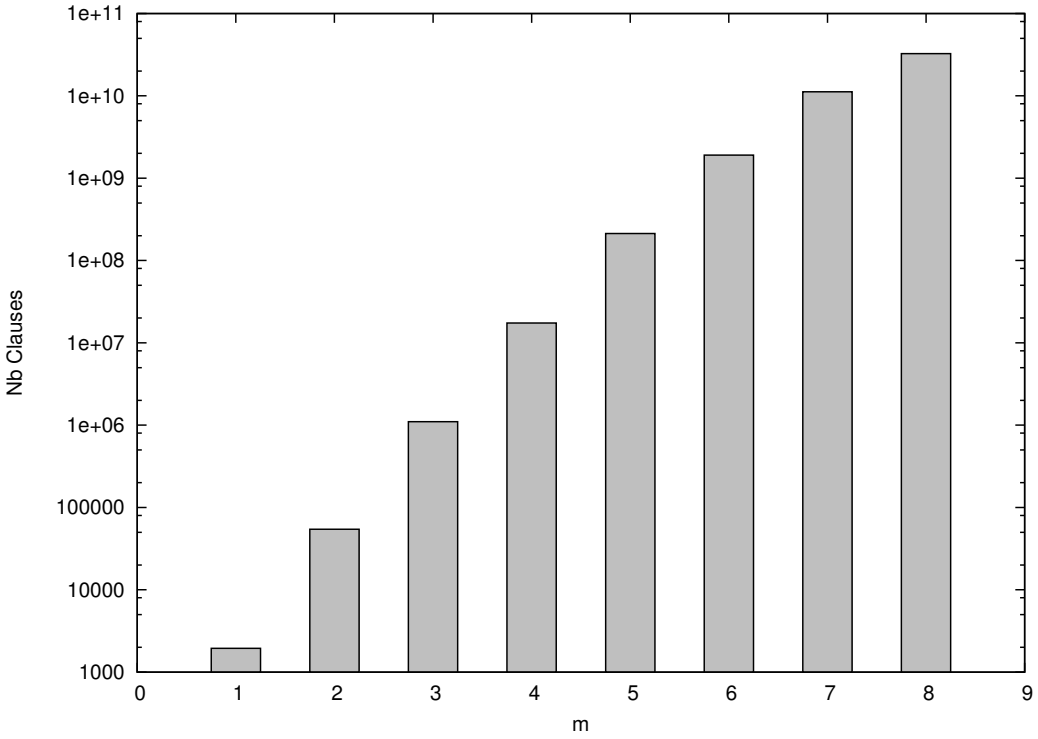

Figure 12. Number of clauses generated by $[SAT - Om]$ for each value of m with $n = 9$.

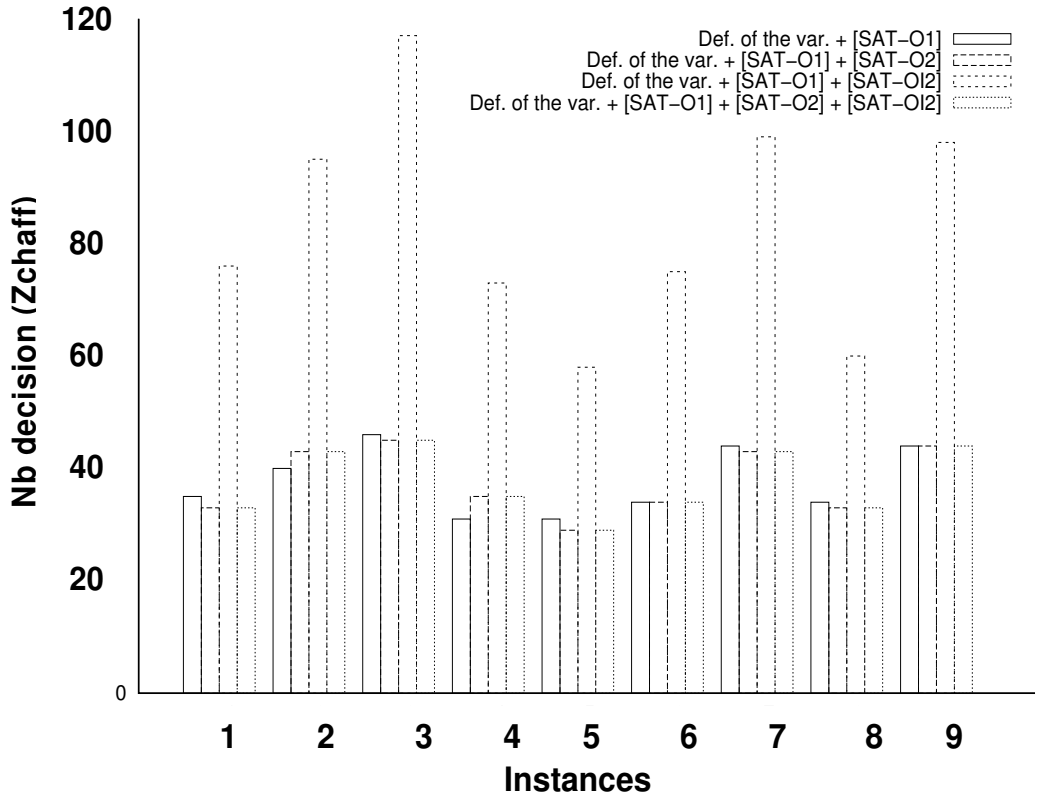

Figure 13. Encoding impact on Zchaff behavior.

and we plan to improve propagation time with a better scheduling of propagators, such as applying complex rules when all standard propagators have already reached a fixed point.

RELATED WORK

Global constraints in CSP

Recent works deal with the combination of several global constraints. van Hoeve et al. (2008) present some filtering algorithms for the sequence constraint and some combinations of sequence. Regin and van Hoeve (2006) studied the conjunction of open global cardinality constraints with specific restrictions. Regin and Gomes (2004) describes the cardinality matrix constraint which imposes that the same value appears p times in the variables of each row (of size n) and q times in the variables of each column (of size m). Consider some Alldiff constraints on the rows and columns of a matrix, this is a special case of the cardinality matrix constraint with $p = q = 1$.

However, this constraint forces each Alldiff to be of size n or m while with our rules, they can be of different sizes. Nevertheless, these approaches require some specialized and complex algorithms for reducing the domains, while our approach allows us to simplify and unify the presentation of the propagation rules and attempts at addressing a wider range of possible combinations of Alldiff.

From the modeling point of view, Simonis (2005) evaluate the difficulty of the Sudoku problem. To this end, various models using different types of constraints are proposed (e.g., the row/column interaction is described by the cardinality matrix global constraint; together with the row/block interaction, this should be compared to the application of our rule **[OI2]** on all intersections of a column and a row, and block and row (or column)). In our approach, we use only the classical model and do not change it; we only add more propagation rules. Moreover, our rules can be used with other problems.

Global constraints in SAT

The basic encodings of CSP into SAT have been fully studied (Bacchus, 2007; Bessiere et al., 2003; Gent, 2002; Walsh, 2000; Sinz, 2005; Gavanelli, 2007; Dimopoulos and Stergiou, 2006; Hoos, 1999) to preserve consistency properties and induce efficient unit propagation in SAT solvers. The specific encodings of global constraint has also been addressed, e.g., cardinality (Bailleu and Boufkhad, 2003; Marques Silva and Lynce, 2007), among (Bacchus, 2007) or Alldiff (Gent and Nightingale, 2004). Our transformation is based on reduction rules and extended to multiple connected Alldiff. As some of these works we proved it correctness with respect to GAC.

CONCLUSION AND FUTURE WORK

We have defined a set of consistency rules for general Alldiff constraints that can easily be implemented in usual constraint solvers. These rules have also been used to encode the same constraints in SAT, preserving some propagation properties through unit propagation. This work provides then a uniform framework to handle interleaved Alldiff and highlights the relationship between CSP and SAT in terms of modeling and resolution when dealing with global constraints.

We now plan to investigate other rules that could be handled in our framework, in order to add new constraints. For instance, if we consider a “ring" of 3 Alldiff constraints having 3 non-empty intersections two by two. $V_{i,j}$ denotes $V_i \cap V_j$, and V now denotes the union of the intersections: $V = V_{1,2} \cup V_{2,3} \cup V_{3,1}$:

$$[SI3.3]: \quad \ll C \wedge \bigwedge_{i=1}^{3} Alldiff(V_i) \mid V_1 1,2 \neq \emptyset \wedge V_1 2,3 \neq \emptyset \wedge V_\perp 3,1 \neq \emptyset \rightarrow \ll C \wedge Alldiff(V) \wedge_\perp (i = 1)^\top 3 \, Alldiff(V_1 i), D \gg \mid true$$

This rule is different from the others since it does not achieve any reduction, but adds a new Alldiff constraint over the union of the intersection of the 3 Alldiff. However, this new constraint will enable more reductions.

We also plan to add a meta mechanism such as in Crawford et al. (2011) for computing efficient strategies for applying rules.

REFERENCES

Apt KR (2003). Principles of Constraint Programming. Cambridge University. Press.

Bacchus F (2007). Gac via unit propagation. In CP 2007, volume 4741 of LNCS, pp. 133-147. Springer, 2007.

Bailleu O, Boufkhad Y (2003). Efficient cnf encoding of boolean cardinality constraints. In CP 2003, volume 2833 of LNCS, pp. 108-122. Springer.

Bessiere C, Hebrard E, Walsh T (2003). Local consistencies in SAT. In SAT 2003, volume 2919 of LNCS, pp. 400-407. Springer.

Bordeaux L, Hamadi Y, Zhang L (2006). Propositional satisfiability and constraint programming: A comparative survey. ACM Computing Survey, 38(4):12.

Crawford B, Soto R, Castro C, Monfroy E, Paredes F (2011). An extensible Autonomous Search framework for Constraint programming. IJPS. In Press.

Cook S (1971). The complexity of theorem-proving procedures. In Third Annual ACM Symposium on Theory of Computing, pp.151-158. ACM.

Davis M, Logemann G, Loveland D (1962). A machine program for theorem proving. Communications of the ACM, 5(7):394-397.

Dimopoulos Y, Stergiou K (2006). Propagation in csp and sat. In CP 2006, volume 4204 of LNCS, pp. 137{-51. Springer.
Elbassioni K, Katriel I, Kutz M, Mahajan M (2005). Simultaneous matchings. In ISAAC 2005, volume 3827 of LNCS, pp. 106-115. Springer.
Gavanelli M (2007). The log-support encoding of csp into sat. In CP 2007, volume 4741 of LNCS, pp. 815-822. Springer.
Gecode (2009). Gecode: generic constraint development environment. http://www.gecode.org/
Gent I (2002). Arc consistency in SAT.Technical Report APES-39A-2002, University of St Andrews.
Gent I, Nightingale P (2004). A new encoding of alldifferent into sat. In Proc. of 3rd Int. Work.onModelling and Reformulating CSP, CP2004, pp. 95-110.
Hirsch E, Le Berre D, Roussel O, Simon L (2010). Sat competitions. http://www.satcompetition.org/.
Hoos H (1999). Sat-encodings, search space structure, and local search performance. In IJCAI 99, pp. 296-303. Morgan Kaufmann.
Kasif S (1990). On the parallel complexity of discrete relaxation in constraint satisfaction networks. AI, 45(3):275-286.
Marques Silva J, Lynce I (2007). Towards robust cnf encodings of cardinality constraints. In CP 2007, volume 4741 of LNCS, p. 483-497. Springer.
Moskewicz M, Madigan C, Zhao Y, Zhang L, Malik S (2001). Chaff: Engineeringan efficient SAT solver. In DAC 2001, pp. 530-535. ACM.
Regin J-C, Gomes C (2004). The cardinality matrix constraint. In CP 2004, pp. 572-587.
Regin J-C (1994). A filtering algorithm for constraint of difference in csps. In Nat. Conf. of AI, pp. 362-367.
Selman B, Kautz H, Cohen B (1994). Noise strategies for improving local search. In Proceedings of the 12th Nat. Conf. on AI, pages 337-343.
Simonis H (2005). Sudoku as a constraint problem. In Proc. of the 4th CP Int. Work. On Modelling and Reformulating Constraint Satisfaction Problems, pp. 17-27.
Sinz C (2005). Towards an optimal cnf encoding of Boolean cardinality constraints.In CP 2005, volume 3709 of LNCS, pp. 827-831. Springer.
Sudopedia. www.sudopedia.org.
van Hoeve W-J, Katriel I (2006). Handbook of Constraint Programming, chapter Global Constraints.Elsevier.
van Hoeve W-J, Pesant G, Rousseau L-M, Sabharwal A (2008). New filtering algorithms for combinations of among constraints.
van Hoeve W-J, Regin J-C (2006). Open constraints in a closed world. In CPAIOR2006, volume 3990 of LNCS, pages 244-257. Springer.
Walsh T (2000). SAT vs CSP. In CP 2000, volume 1894 of LNCS, pp. 441-456. Springer.