SAT Encoding and CSP Reduction for Interconnected Alldiff Constraints

Frédéric Lardeux, Eric Monfroy, Frédéric Saubion, Broderick Crawford, Carlos Castro

SAT Encoding and CSP Reduction for Interconnected Alldiff Constraints

F. Lardeux[3], E. Monfroy[1,2], F. Saubion[3], B. Crawford[1,4], and C. Castro[1]

[1] Universidad Técnica Federico Santa María, Valparaíso, Chile
[2] LINA, Université de Nantes, France
[3] LERIA, Université d'Angers, France
[4] Pontificia Universidad Católica de Valparaíso, PUCV, Chile

Abstract. Constraint satisfaction problems (CSP) or Boolean satisfiability problem (SAT) are two well known paradigm 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.

1 Introduction

During the last decades, two closely related communities have focused on the resolution of combinatorial problems. On the one hand, the SAT community has developed very efficient algorithms to handle the seminal Boolean satisfaction problem which model is very simple: Boolean variables and CNF propositional formulas. The complete resolution techniques (i.e., able to decide if an instance is satisfiable or not) mainly rely on the DPLL procedure [6] whereas incomplete algorithms are mostly based on local search procedures [11]. In addition, very sophisticated techniques (e.g., symmetries detection, learning, hybrid heuristics) were proposed to build very efficient solvers able to handle huge and very difficult benchmarks. On the other hand, the constraint programming community has focus on the resolution of discrete constraint satisfaction problems (CSP). This paradigm provides the user with a more general modeling framework: problems are expressed by a set of decision variables (which values belong to finite integer domains) and constraints model relations 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 explore the search space according to specific or general heuristics (metaheuristics). Again, much work has been achieved to define very efficient propagation algorithms and search heuristics.
Concerning this resolution aspect, the two paradigms share some common principles (see [5] for a comparative survey). Here, we 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).
The identification of typical constraints in CSP (so-called global constraints) has increased the declarativity, and the development of specialized algorithms has significantly improved the resolution performances. The first example is certainly the Alldiff constraint [15] expressing that a set of variables have all different values. This constraint is very useful since it naturally appears in the modeling of numerous problems (timetabling, planning, resource allocation, ...). Moreover, usual propagation techniques are inefficient for this constraint (due to limited domain reduction when decomposing the constraint into $n*(n-1)/2$ disequalities) and specific algorithms were proposed to boost resolution [18]. On the SAT side, no such high level modeling feature is offered. As a consequence, benchmarks (e.g., industrials ones) are often incomprehensible by users and sophisticated preprocessings should be used to simplify their structures. 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 [9, 21, 8] to ensure some consistency properties to the Boolean encodings. Even if some specific relationships 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 [2, 3, 10, 12]. When modeling problems, the user has often to 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 and solvers a recent step consists in handling efficiently combinations of global constraints (e.g., [19, 20, 14]).
We focus on possibly interleaved (i.e., sharing variables) Alldiff constraints, such as it appears in numerous problems (e.g., Latin squares, Sudoku [16]). Our purpose is twofold. 1) We want to provide, on the CSP solving side, a uniform propagation framework to handle possibly interleaved Alldiff constraints. From an operational point of view, we define propagation rules to improve resolution efficiency by tacking into account specific properties. 2) We also want to generalize possible encodings of (multiple) Alldiff constraints in SAT (i.e., 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; if one is interested in better efficiency, the solvers can then be improved (based on the CSP rules structures or on the SAT formulas structures) to take advantage of their own facilities.

2 Encoding CSP vs. 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 $\forall c \in C, d \in c$. We consider C as a set of constraint (equivalent to a conjunction). Usual resolution processes [1, 5] are based on two main components: reduction and search strategies. Search consists in enumerating the possible values of a given variable to progressively build and reach a solution. Reduction techniques are added at each node to reduce the search space (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, e.g., efficient propagation algorithms for global constraints. We recall a basic consistency notion (the seminal arc consistency is the binary subcase of this definition).

Definition 1 (Generalized Arc Consistency (GAC)). *A constraint*[5] c *on variables* $(x_1, \cdots, x_m)$ *is 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. $(d_1, \cdots, d_m) \in c$.

Domain Reduction Rules Inspired by [1], we use a formal system to precisely define reduction rules to reduce domains w.r.t. constraints. We abstract constraint propagation as a transition process over CSPs. A domain reduction rule is of the form:

$$\frac{(X, C, D)|\Sigma}{(X, C, D')|\Sigma'}$$

where $D' \subseteq D$ and Σ and Σ' are first order formulas (i.e., conditions of 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 $\bigcup_{x \in V} D_x$. $\#D$ is the set cardinality.
Given a CSP (X^k, C^k, D^k), a transition can be performed to get a reduced CSP $(X^{k+1}, C^{k+1}, D^{k+1})$ if there is an instance of a rule (i.e., a renaming without variables' conflicts):

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

such that $D^k \models \bigwedge_{x \in X} x \in D^k_x \wedge \Sigma^k$, and D^{k+1} is the greatest subset of D^k such that $D^{k+1} \models \bigwedge_{x \in X} x \in D^{k+1}_x \wedge \Sigma^{k+1}$.

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 variables of V; and $d_1, d_2 \notin D_x$ (resp. D_V) is a shortcut for $d_1 \notin D_x \ \wedge \ d_2 \notin D_x$ (resp. $d_1 \notin D_V \ \wedge \ d_2 \notin D_V$).
Since we only consider here rules that does 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

[5] This definition is classically extended to a set of constraints.

that $< C, D >$ is GAC if C is GAC w.r.t. D. For example, a very basic rule to enforce basic node consistency [1] on equality could be:

$$\frac{< C \wedge x = d, D > | d' \in D_x, d' \neq d}{< C \wedge x = d, D' > | d' \notin D'_x}$$

This rule could be applied on $< X = 2, \{D_X \equiv \{1, 2, 3\}\} >$ with $3 \in D_X, 3 \neq 2$ to obtain $< X = 2, \{D_X \equiv \{1, 2\}\} >$; and so on.
The transition relation using a rule R is denoted $< C, D > \rightarrow_R < C, D' >$. $\rightarrow_R *$ denotes the reflexive transitive closure of $\rightarrow_R$. It is clear that $\rightarrow_R$ terminates due to the decreasing criterion on domains in the definition of the rules (see [1]). This notion can be obviously extended to sets of rules $\mathcal{R}$. Note also that we require that the result of $\rightarrow_{\mathcal{R}} *$ is independent from the order of application of the rules [1] (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 $\Xi = \{\xi_1, ..., \xi_n\}$ 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 [21] : $\forall x \in X, \forall d \in D_x, \exists \xi_x^d \in \Xi$ (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:

$$\bigwedge_{x \in X} \bigvee_{d \in D_x} \xi_x^d \quad \text{and} \quad \bigwedge_{x \in X} \bigwedge_{\substack{d_1, d_2 \in D_x \\ d_1 \neq d_2}} (\neg \xi_x^{d_1} \vee \neg \xi_x^{d_2})$$

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 [2]. 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 [3, 10, 12]. 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 [5]. 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 [21, 2], 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).

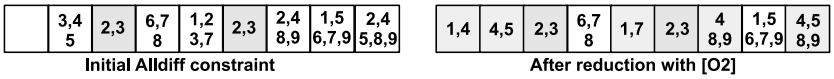

Fig. 1. Application of $[O2]$

3 Alldiff Constraints: Reduction rules and Transformation

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

A Single $Alldiff$ **constraint** We first reformulate a well known consistency property [15, 18] w.r.t. 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.

$$[O1] \quad \frac{< C \wedge Alldiff(V), D > | x \in V \ \wedge \ D_x = \{d_1\}}{< C \wedge Alldiff(V), D' > | d_1 \notin D'_{V \setminus \{x\}}}$$

Property 1. If $< Alldiff(V), D > \rightarrow^*_{[O1]} < Alldiff(V), D' >$, then the corresponding conjunction $\bigwedge_{x_i,x_j \in V} x_i \neq x_j$ is GAC w.r.t. $< D' >$. 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, $1 \leq m \leq (\#V - 1)$:

$$[Om] \quad \frac{< C \wedge Alldiff(V), D > | V' \subset V \ \wedge \ D_{V'} = \{d_1, \ldots, d_m\}}{< C \wedge Alldiff(V), D' > | d_1, \ldots, d_m \notin D'_{V \setminus V'}}$$

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 (see Figure 1).

Property 2. Given $< Alldiff(V), D > \rightarrow^*_{[Om]_{1 \leq m \leq (\#V-1)}} < Alldiff(V), D' >$, then $< Alldiff(V), D' >$ has the GAC property.

The proof can be obtained from [15]. 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:

$$[SAT - O1] \quad \bigwedge_{x \in V} \bigwedge_{y \in V \setminus \{x\}} (\neg \xi^d_x \vee (\bigvee_{f \in D_x \setminus \{d\}} \xi^f_x) \vee \neg \xi^d_y)$$

This representation preserves GAC. Indeed, if $\neg \xi^d_x$ is false (i.e. when the variable x is valued to d) and $\bigvee_{f \in D_{x_1} \setminus \{d\}} \xi^f_{x_1}$ is false (i.e., when the variable x_1 is valued to d) then $\neg \xi^d_{x_2}$ must be true to satisfy the clause (x_2 cannot be valued to d).

Generalized to a subset V' of m variables $\{x_1, ..., x_m\}$ with m possible values $\{d_1, \ldots, d_m\}$, $1 \leq m \leq (\#V - 1)$, the $\#(V \setminus V') * m^{m+1}$ clauses are:

$$[SAT - Om] \bigwedge_{y \in V \setminus V'} \bigwedge_{k=1}^{m} \bigwedge_{p_1=1}^{m} \cdots \bigwedge_{p_m=1}^{m} \big[(\bigvee_{s=1}^{m} \neg \xi_{x_s}^{d_{p_s}}) \vee (\bigvee_{i=1}^{m} \bigvee_{f \in D_{x_i} \setminus \{d_1, \ldots, d_m\}} \xi_{x_i}^{f}) \vee \neg \xi_{y}^{d_k} \big]$$

Property 3. $\bigcup_{1 \leq m \leq \#V-1} [SAT - Om]$ preserves the GAC property.

Proof. As mentioned above, 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). ¡

4 Multiple Overlapping Alldiff Constraints

In 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 $\#V = \#D_V$. This restriction could be weaken but it is generally needed by classical problems (e.g., 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.

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.

$$[OI2] \quad \frac{< C \wedge Alldiff(V_1) \ \wedge \ Alldiff(V_2), D > | d \in D_{V_1 \cap V_2} \ \wedge \ d \notin D_{V_2 \setminus V_1}}{< C \wedge Alldiff(V_1) \ \wedge \ Alldiff(V_2), D' > | d \notin D'_{V_1 \setminus V_2}}$$

$[OI2]$ is coded in SAT as $\#D_{V_1 \cap V_2} * \#(V_1 \cap V_2) * \#(V1 \setminus V_2)$ clauses:

$$[SAT - OI2] \quad \bigwedge_{d \in D_{V_1 \cap V_2}} \bigwedge_{x_1 \in V_1 \cap V_2} \bigwedge_{x_2 \in V_1 \setminus V_2} \bigvee_{x_3 \in V_2 \setminus V_1} (\neg \xi_{x_1}^{d} \vee \xi_{x_2}^{d} \vee \neg \xi_{x_3}^{d})$$

$[OI2]$ can be extended to $[OIm]$ 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: $V = \bigcap_{i=1}^{m} V_i$

$$[OIm] \quad \frac{< C \bigwedge_{i=1}^{m} Alldiff(V_i), D > | d \in D_V \ \wedge \ d \notin D_{V_1 \setminus V}}{< C \bigwedge_{i=1}^{m} Alldiff(V_i), D' > | d \notin \bigcup_{i=2}^{m} D'_{V_i \setminus V}}$$

Note that this rule can be implicitly applied to the different symmetrical possible orderings of the m Alldiff.
$[OIm]$ is translated in SAT as $\#D_V * \#V * \sum_{i=2}^{m} \big(\#(V_i \setminus V)\big)$ clauses:

$$[SAT - OIm] \quad \bigwedge_{d \in D_V} \bigwedge_{x_1 \in V} \bigwedge_{i=2}^{m} \bigwedge_{x_3 \in V_i \setminus V} \bigvee_{x_2 \in V_1 \setminus V} (\neg \xi_{x_1}^{d} \vee \xi_{x_2}^{d} \vee \neg \xi_{x_3}^{d})$$

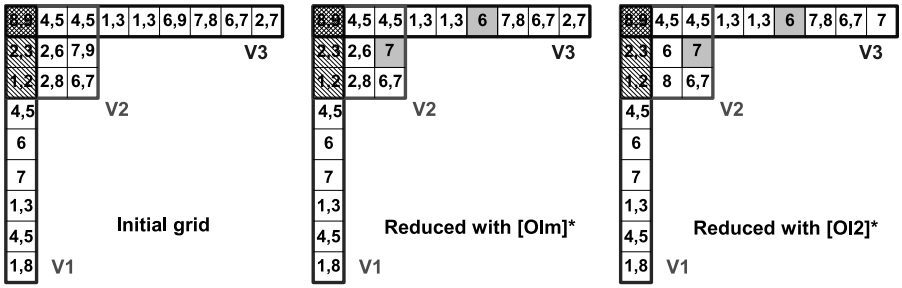

Fig. 2. $[OI2]^*$ reduces more than $[OIm]$

Property 4. Consider $m > 2$ Alldiff with a non empty intersection. Given $< C, D > \rightarrow^*_{[OIm]} < C, D' >$ and $< C, D > \rightarrow^*_{[OI2]} < C, D'' >$, then $D'' \subseteq D'$.

The proof is straightforward (see illustration on Figure 2). Consider the application of $[OIm]$: $9 \in V_1 \cap V_2 \cap V_3$, and 9 is 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); no other application of $[OIm]$ is possible, leading to the second grid. Now, consider the application of $[OI2]$ on the initial grid: first between V_1 and V_2; $9 \in V_1 \cap V_2$ and 9 is not in the rest of V_1; thus, 9 can be removed from V_2; the same for 2; $[OI2]$ 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 3rd grid which is smaller than the second.
Although one could argue that $[OIm]$ is useless (Prop. 4) in terms of reduction, in practice $[OIm]$ can be interesting in terms of the number of rules to be applied. Moreover, $[OIm]$ 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 (see Figure 3). 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: $V = V_{1,2} \cup V_{2,3} \cup V_{3,4} \cup V_{1,4}$.

$$[SI4.4] \quad \frac{< C \bigwedge_{i=1}^{4} Alldiff(V_i), D > | \quad V_{1,2} \neq \emptyset \wedge V_{2,3} \neq \emptyset \wedge V_{3,4} \neq \emptyset \wedge V_{1,4} \neq \emptyset \ \wedge \ d \in D_V \ \wedge \ d \notin D_{V_{13} \setminus V_{24}}}{< C \bigwedge_{i=1}^{4} Alldiff(V_i), D' > | d \notin D'_{V_{24} \setminus V_{13}}}$$

d must at least be an element of 2 opposite intersections (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 $\#D_V * \#V * \#(V_{24} \setminus V_{13})$ clauses with $V_{1,2} \neq \emptyset \wedge V_{2,3} \neq \emptyset \wedge V_{3,4} \neq \emptyset \wedge V_{1,4} \neq \emptyset$:

$$[SAT - SI4.4] \quad \bigwedge_{d \in D_V} \bigwedge_{x_1 \in V} \bigwedge_{x_3 \in V_{24} \setminus V_{13}} \bigvee_{x_2 \in V_{13} \setminus V_{24}} (\neg \xi^d_{x_1} \vee \xi^d_{x_2} \vee \neg \xi^d_{x_3})$$

This rule can be generalized to a **ring** of $2m$ Alldiff with $2m$ **non-empty** intersections. 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 (resp. even): $\bigcup_{i=0}^{m-1} V_{2i+1}$ (resp. $\bigcup_{i=1}^{m} V_{2i}$).

$$[SI2m.2m] \quad \frac{< C \bigwedge_{i=1}^{2.m} Alldiff(V_i), D > | \; \bigwedge_{i=1}^{2.m}(V_i \cap V_{(i \bmod 2m)+1} \neq \emptyset) \; \wedge \; d \in D_V \; \wedge \; d \notin D_{V_{odd} \setminus V}}{< C \bigwedge_{i=1}^{2.m} Alldiff(V_i), D' > | d \notin D'_{V_{even} \setminus V}}$$

These are $\#D_V * \#V * \#(V_{even} \setminus V)$ SAT clauses s.t. $\bigwedge_{i=1}^{2.m}(V_i \cap V_{(i \bmod 2m)+1} \neq \emptyset)$:

$$[SAT - SI2m.2m] \quad \bigwedge_{d \in D_V} \bigwedge_{x_1 \in V} \bigwedge_{x_3 \in V_{even} \setminus V} \bigvee_{x_2 \in V_{odd} \setminus V} (\neg \xi^d_{x_1} \vee \xi^d_{x_2} \vee \neg \xi^d_{x_3})$$

The reduction we obtain by applying rules for a single Alldiff and rules for several Alldiff is stronger than enforcing GAC:

Property 5. Given a conjunction of constraints $C = \bigwedge_{i=1}^{k} Alldiff(V_i)$ and a set of domains D. Given 2 sets of rules $R' = \bigcup_{l=1}^{max_i\{\#V_i\}} \{[Ol]\}$ and $R \subseteq \{[OI2], \ldots, [OIm], [SI4.4], \ldots, [SI2m.2m]\}$. Consider $< C, D > \rightarrow^*_{R'} < C, D' >$ and $< C, D > \rightarrow^*_{R' \cup R} < C, D'' >$ then $< C, D' >$ and $< C, D'' >$ are GAC and moreover $D'' \subseteq D'$.

The proof is based on the fact that $\bigcup_{l=1}^{max_i\{\#V_i\}} \{[Ol]\}$ already enforces GAC and that the $[OIm], [SI2m.2m]$ preserves GAC.

Property 6. $[SAT-OI2]$ (respectively $[SAT-OIm]$, $[SAT-SI4.4]$, and $[SAT-SI2m.2m]$) preserves the consistency property of $[OI2]$ (respectively $[OIm]$, $[SI4.4]$, and $[SI2m.2m]$).

The proof is similar to the proof of [SAT-Om] $\Longleftrightarrow$ [Om] of Property 3.

5 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., [17]) which can be

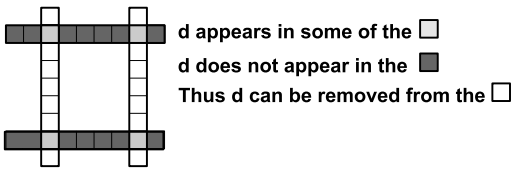

Fig. 3. Example of $[SI4.4]$

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 $[1..n]$:

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

Sudoku puzzle is a special case (i.e., 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:

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

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-O1]) whereas encoding all the rules generates approximately 886×10^9 clauses. This increase of the number of clauses is mainly due to [SAT-Om] (see Figure 4). 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 [13] on 9 Sudoku grids coded with:

- Definition of the variables + [SAT-O1]
- Definition of the variables + [SAT-O1] + [SAT-O2]
- Definition of the variables + [SAT-O1] + [SAT-OI2]
- Definition of the variables + [SAT-O1] + [SAT-O2] + [SAT-OI2]

Figure 5 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.

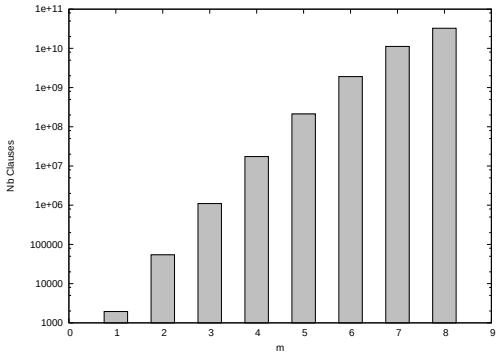

Fig. 4. Number of clauses generated by [SAT-Om] for each value of m with $n = 9$

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, i.e., 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. The preliminary results are promising, 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.

6 Related Work and Conclusion

Global constraints in CSP Recent works deal with the combination of several global constraints. [19] presents some filtering algorithms for the sequence constraint and some combinations of sequence. [20] studied the conjunction of open global cardinality constraints with specific restrictions. [14] 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 $p = q = 1$.

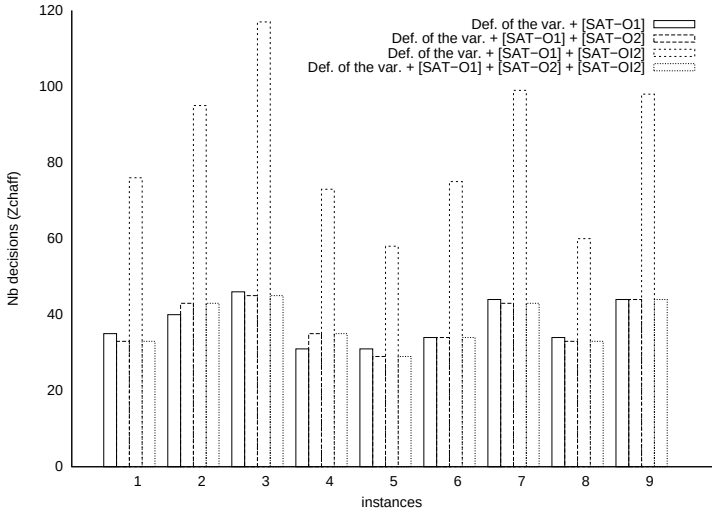

Fig. 5. Encoding impact on the behavior of Zchaff

However, this constraint forces each Alldiff to be of size n or m while with our rules, they can be of different sizes.

Nevertheless, theses 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, [16] evaluate the difficulty of the Sudoku problem. To this end, various modelings 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 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 [2, 4, 9, 21, 8, 7] to preserve consistency properties and induce efficient unit propagation in SAT solvers. The specific encodings of global constraint has been also addressed, e.g., Cardinality [3, 12], Among [2] or Alldiff [10]. Our transformation is based on reduction rules and extended to multiple connected Alldiff. As some of these works we proved it correctness w.r.t. GAC.

Conclusion We have defined a set of consistency rules for general Alldiff constraints that can be easily 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 an

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 solve new combinations of global constraints.

References

1. K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
2. F. Bacchus. Gac via unit propagation. In *CP 2007*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.
3. O. Bailleux and Y. Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *CP 2003*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.
4. C. Bessiere, E. Hebrard, and T. Walsh. Local consistencies in SAT. In *SAT 2003*, volume 2919 of *LNCS*, pages 400–407. Springer, 2003.
5. L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Compututing Survey*, 38(4):12, 2006.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
7. Yannis Dimopoulos and Kostas Stergiou. Propagation in csp and sat. In *CP 2006*, volume 4204 of *LNCS*, pages 137–151. Springer, 2006.
8. Marco Gavanelli. The log-support encoding of csp into sat. In *CP 2007*, volume 4741 of *LNCS*, pages 815–822. Springer, 2007.
9. I. Gent. Arc consistency in SAT. Technical Report APES-39A-2002, University of St Andrews, 2002.
10. I. Gent and P. Nightingale. A new encoding of alldifferent into sat. In *Proc. of 3rd Int. Work. on Modelling and Reformulating CSP, CP2004*, pages 95–110, 2004.
11. Holger H. Hoos. Sat-encodings, search space structure, and local search performance. In *IJCAI 99*, pages 296–303. Morgan Kaufmann, 1999.
12. J. Marques Silva and I. Lynce. Towards robust cnf encodings of cardinality constraints. In *CP 2007*, volume 4741 of *LNCS*, pages 483–497. Springer, 2007.
13. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC 2001*, pages 530–535. ACM, 2001.
14. J.-C. Régin and C. Gomes. The cardinality matrix constraint. In *CP 2004*, pages 572–587, 2004.
15. J.C. Régin. A filtering algorithm for constraint of difference in csps. In *Nat. Conf. of AI*, pages 362–367, 1994.
16. H. Simonis. Sudoku as a constraint problem. In *Proc. of the 4th CP Int. Work. on Modelling and Reformulating Constraint Satisfaction Problems*, pages 17–27, 2005.
17. Sudopedia. www.sudopedia.org.
18. W.-J. van Hoeve and I. Katriel. *Handbook of Constraint Programming*, chapter Global Constraints. Elsevier, 2006.
19. W.-J. van Hoeve, G. Pesant, L.-M. Rousseau, and A. Sabharwal. New filtering algorithms for combinations of among constraints, 2008. Under review.
20. W.-J. van Hoeve and J.-C. Régin. Open constraints in a closed world. In *CPAIOR 2006*, volume 3990 of *LNCS*, pages 244–257. Springer, 2006.
21. T. Walsh. SAT v CSP. In *CP 2000*, volume 1894 of *LNCS*, pages 441–456. Springer, 2000.