

Managing Multiple Alldifferent Constraints in CSP and SAT ^{*}

Frédéric Lardeux³, Eric Monfroy^{1,2}, and Frédéric Saubion³

¹ Universidad Técnica Federico Santa María, Valparaíso, Chile

² LINA, Université de Nantes, France

³ LERIA, Université d'Angers, France

Abstract. Combinatorial problems are generally modeled as constraint satisfaction problems (CSP) or Boolean satisfiability problem (SAT). Modeling and resolution of CSP is often strengthened by *global constraints* (e.g., Alldiff constraint) whereas no such high level feature is offered in SAT. In this paper, we propose a uniform propagation framework to handle Alldiff constraints (possibly interleaved) with some reduction rules. We also propose a SAT encoding of these rules that preserves the reduction properties of CSP.

1 Introduction

The resolution of combinatorial problems involves two basic choices: a modeling framework and an appropriate solver. Concerning the model, a suitable solution consists in expressing the problem as a constraint satisfaction problem (CSP), which is classically expressed by a set of decision variables whose values belong to finite integer domains. Constraints are used to model the relationships that exist between these variables. Another possibility is to encode the problem as a Boolean satisfiability problem (SAT), where a set of Boolean variables must satisfy a propositional formula.

Concerning the resolution of such problems, the two paradigms share some common principles (we refer the reader to [4] for a comparative survey). Here we focus one complete methods aims at exploring a tree by enumerating variables and reducing the search space using propagation techniques.

On the CSP side, the identification of typical constraints, so-called global constraints, that arise in several real-world problems and the development of very specialized and efficient algorithms has considerably improve the resolution performances. The first example was certainly the Alldiff constraint [12] 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 (timetabling, planning, resource allocation, ...). 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 $n * (n - 1)/2$ disequalities and specific algorithms have been proposed to

^{*} This article is submitted for publication in AIMS 2008.

boost resolution [15].

From these 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 [6]. For instance, let 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 games [14].

On the SAT side, no such high level modeling feature is offered to the user, who has to translate its problem into propositional logic. Systematic basic transformations from CSP to SAT have been proposed [18, 5, 9] to ensure some consistency properties to the Boolean encodings. Even if some specific relationships between variables such as 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 [3, 10, 13, 2].

Therefore, in this paper, our purpose is twofold:

- 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.
- We also want to generalize possible encodings of Alldiff and 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.

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 enables us to perform more reduction (i.e., to reduce more 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.

2 Encoding CSP vs. SAT

2.1 CSP : Basic Notions

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

Usual resolution processes [1, 4] are based on two main components: 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, e.g., efficient constraint 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 constraint⁴ c on variables $(x_{i_1}, \dots, x_{i_m})$ is generalized arc-consistent iff $\forall k \in 1..m, \forall d \in D_{i_k}, \exists (d_{i_1}, \dots, d_{i_{k-1}}, d_{i_{k+1}}, \dots, d_{i_n}) \in D_{i_1} \times \dots \times D_{i_{k-1}} \times D_{i_{k+1}} \times \dots \times D_{i_n},$ s.t. $(d_{i_1}, \dots, d_{i_m}) \in c$.

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

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 variable 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. $d_1 \notin D_V \wedge d_2 \notin D_V$).

In order to simplify the notation, 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

⁴ This definition is classically extended to a set of constraints.

our notation to $\langle C, D \rangle$. For example, a very basic rule to enforce basic node consistency [1] on equality could be:

$$\frac{\langle C \wedge x = d, D \rangle \mid d' \in D_x, d' \neq d}{\langle C \wedge x = d, D' \rangle \mid d' \notin D'_x}$$

This rule could be applied on $\langle X = 2, \{D_X \equiv \{1, 2, 3\}\} \rangle$ with $3 \in D_X, 3 \neq 2$ to obtain $\langle X = 2, \{D_X \equiv \{1, 2\}\} \rangle$.

The transition relation using a rule R is denoted $\langle C, D \rangle \rightarrow_R \langle C, D' \rangle$. \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 the result of $\rightarrow_{\mathcal{R}}^*$ is independent from the rule application order [1]: from a practical point of view, it is thus generally faster to first sequence less complicated (rules that execute faster) rules.

2.2 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 $\Xi = \{\xi_1, \dots, \xi_n\}$ 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 [18]: $\forall x \in X, \forall d \in D_x, \exists \xi_{x,d} \in \Xi$ ($\xi_{x,d}$ is true when x has the value d , false otherwise).

To enforce exactly one value for each variable, the next clauses must be present:

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

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 mentionned in the introduction, several work have addressed the encodings of usual global constraints into SAT [3, 10, 13]. 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 select variable and unit propagation (UP) which allows to propagate unit clauses in the current formula [4]. As mentioned in the introduction, this principle is very close to the propagation of constraints achieved by reduction rules to enforce consistency. Therefore, we will compare the two encoding CSP and SAT from this consistency point of view. According to [18, 2], we will 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).

3 Alldiff Constraints: Reduction rules and Transformation

In the following, we classically note $Alldiff(V)$ the Alldiff constraint on a subset of variable 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 [12, 15] 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.

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

Property 1. If $\langle Alldiff(V), D \rangle \xrightarrow{*[O1]} \langle Alldiff(V), D' \rangle$, then the corresponding conjunction $\bigwedge_{x_i, x_j \in V} x_i \neq x_j$ is GAC w.r.t. $\langle D' \rangle$. Note that enforcing GAC on the disequalities reduces less the domains than enforcing GAC of the global Alldiff constraint with [O1].

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

$$[Om] \quad \frac{\langle C \wedge Alldiff(V), D \rangle \mid V' \subset V \wedge D_{V'} = \{d_1, \dots, d_m\}}{\langle C \wedge Alldiff(V), D' \rangle \mid d_1, \dots, d_m \notin D_{V \setminus V'}}$$

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.

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

The proof can be obtained from [12].

Now, the Alldiff constraints can be translated in SAT, by encoding [O1] with a set of $\#V^2 * (\#V - 1)$ CNF clauses:

$$[SAT - O1] \quad \bigwedge_{\substack{x_1, x_2 \in V \\ x_1 \neq x_2}} \bigwedge_{d \in D_{x_1}} ((\bigvee_{f \in D_{x_1} \setminus \{d\}} \xi_{x_1, f}) \vee \neg \xi_{x_2, d})$$

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

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

$$[SAT - Om] \quad \bigwedge_{y \in V \setminus V'} \bigwedge_{i=1}^m \left(\bigvee_{j=1}^m \left(\left(\bigvee_{k=1}^m \neg \xi_{x_j, d_k} \right) \left(\bigvee_{f \in D_{x_j} \setminus \{d_1, \dots, d_m\}} \xi_{x_j, f} \right) \right) \vee \neg \xi_{y, d_i} \right)$$

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 process as sketch below:

$$\begin{aligned} [Om] & \frac{\langle C \wedge Alldiff(V), D \rangle | V' \subset V \wedge D_{V'} = \{d_1, \dots, d_m\}}{\langle C \wedge Alldiff(V), D' \rangle | d_1, \dots, d_m \notin D_{V \setminus V'}} \\ \xrightarrow{\text{Encoding}} & \left(\xi_{x_1, d_1} \wedge \dots \wedge \xi_{x_1, d_m} \wedge \left(\bigwedge_{f \in D_{x_1} \setminus \{d_1, \dots, d_m\}} \neg \xi_{x_1, f} \right) \right) \dots \\ & \left(\xi_{x_m, d_1} \wedge \dots \wedge \xi_{x_m, d_m} \wedge \left(\bigwedge_{f \in D_{x_m} \setminus \{d_1, \dots, d_m\}} \neg \xi_{x_m, f} \right) \right) \\ & \rightarrow \bigwedge_{y \in V \setminus V'} (\neg \xi_{y, d_1} \wedge \dots \wedge \neg \xi_{y, d_m}) \\ \iff & \bigwedge_{y \in V \setminus V'} \left[\left(\left(\neg \xi_{x_1, d_1} \vee \dots \vee \neg \xi_{x_1, d_m} \left(\bigvee_{f \in D_{x_1} \setminus \{d_1, \dots, d_m\}} \xi_{x_1, f} \right) \right) \vee \dots \right. \right. \\ & \left. \vee \left(\neg \xi_{x_m, d_1} \vee \dots \vee \neg \xi_{x_m, d_m} \left(\bigvee_{f \in D_{x_m} \setminus \{d_1, \dots, d_m\}} \xi_{x_m, f} \right) \right) \vee \neg \xi_{y, d_1} \right) \wedge \dots \\ & \left. \wedge \left(\left(\neg \xi_{x_1, d_1} \vee \dots \vee \neg \xi_{x_1, d_m} \left(\bigvee_{f \in D_{x_1} \setminus \{d_1, \dots, d_m\}} \xi_{x_1, f} \right) \right) \vee \dots \right. \right. \\ & \left. \left. \vee \left(\neg \xi_{x_m, d_1} \vee \dots \vee \neg \xi_{x_m, d_m} \left(\bigvee_{f \in D_{x_m} \setminus \{d_1, \dots, d_m\}} \xi_{x_m, f} \right) \right) \vee \neg \xi_{y, d_m} \right) \right] \\ \iff & [SAT - Om] \text{ (by grouping } \vee \text{ and } \wedge \text{ conveniently)} \quad \square \end{aligned}$$

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 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 pairwise 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 safely be removed from the other variables of the second Alldiff.

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

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

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

[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: $V = \bigcap_{i=1}^m V_i$

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

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

[OIm] is translated in SAT as $\#V * \#(V_1 \setminus V) * \sum_{i=1}^m (\#(V_i \setminus V)) * \#D_V$ clauses:

$$[SAT - OIm] \quad \bigwedge_{i=1}^m \bigwedge_{x_1 \in V} \bigwedge_{x_2 \in V_1 \setminus V} \bigwedge_{x_3 \in V_i \setminus V} \bigwedge_{d \in D_{x_1}} (\neg \xi_{x_1, d} \vee \xi_{x_2, d} \vee \neg \xi_{x_3, d})$$

Property 4. Consider $m > 2$. Then the application of [OIm] over $\text{Alldiff}(V_1), \dots, \text{Alldiff}(V_m)$ achieves less reduction than the full application of [OI2] over the $n * (n - 1)$ pairwise distinct couples $\text{Alldiff}(V_i), \text{Alldiff}(V_j)$.

The proof is straightforward.

Several *Alldiff* connected by several intersections We first consider 4 *Alldiff* having four intersections two by two. V now denotes the union of the four intersections: $V = (V_1 \cap V_2) \cup (V_2 \cap V_3) \cup (V_3 \cap V_4) \cup (V_1 \cap V_4)$. V_{13} (respectively V_{24}) denotes $V_1 \cup V_3$ (respectively $V_2 \cup V_4$)

$$[SI4.4] \quad \frac{\langle C \bigwedge_{i=1}^4 \text{Alldiff}(V_i), D \rangle |d \in D_V \wedge d \notin D_{V_{13} \setminus V_{24}}}{\langle C \bigwedge_{i=1}^4 \text{Alldiff}(V_i), D' \rangle |d \notin D_{V_{24} \setminus V_{13}}}$$

To explain this rule, we illustrate it with the Sudoku problem modeled as a 9x9 matrix of variables with domains $\{1 \dots 9\}$ and a conjunction of *Alldiff* constraints, one per line, one per column, and 1 per block. Suppose that V_1 and V_3 are lines; V_2 and V_4 are columns; d is a value that appears in one (or several) of the intersections; d does not appear in the 2 lines (except maybe in the intersections with the columns). Then, d can be removed from each cell of the 2 columns (except the cells that intersect lines V_1 and V_3). Obviously, this can also be done with blocks in the Sudoku.

Translated in SAT, we obtain $\#V * \#(V_{13} \setminus V_{24}) * \#(V_{24} \setminus V_{13}) * \#D_V$ clauses:

$$[SAT - SI4.4] \quad \bigwedge_{x_1 \in V} \bigwedge_{x_2 \in V_{13} \setminus V_{24}} \bigwedge_{x_3 \in V_{24} \setminus V_{13}} \bigwedge_{d \in D_{x_1}} (\neg \xi_{x_1, d} \vee \xi_{x_2, d} \vee \neg \xi_{x_3, d})$$

This rule can be generalized to $2m$ Alldiff with $2m$ 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. $\bigcup_{i=1}^m V_{2i}$).

$$[SI2m.2m] \quad \frac{\langle C \bigwedge_{i=1}^{2m} Alldiff(V_i), D \rangle \mid d \in D_V \wedge d \notin D_{V_{odd} \setminus V}}{\langle C \bigwedge_{i=1}^{2m} Alldiff(V_i), D' \rangle \mid d \notin D_{V_{even} \setminus V}}$$

We obtain in SAT $\#V * \#(V_{odd} \setminus V) * \#(V_{even} \setminus V) * \#D_V$ clauses:

$$[SAT-SI2m.2m] \quad \bigwedge_{x_1 \in V} \bigwedge_{x_2 \in V_{odd} \setminus V} \bigwedge_{x_3 \in V_{even} \setminus V} \bigwedge_{d \in D_{x_1}} (\neg \xi_{x_1, d} \vee \xi_{x_2, d} \vee \neg \xi_{x_3, d})$$

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 a subset of rules $R \subseteq \{[OIm], [SI2m.2m]\}$ and $R' = \bigcup_{k=1}^{max_i \{ \#V_i \}} \{[Ok]\}$. Consider $\langle C, D \rangle \xrightarrow{*R'} \langle C, D' \rangle$ and $\langle C, D \rangle \xrightarrow{*R' \cup R} \langle C, D'' \rangle$ then D' and D'' are GAC and moreover $D'' \subseteq D'$.

The proof is based on the fact that $\bigcup_{k=1}^{max_i \{ \#V_i \}} \{[Ok]\}$ already enforces GAC and that $[OIm], [SI2m.2m]$ 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 $[SAT-Om] \iff [Om]$ of Property 3.

Similarly, we have also defined some other rules (e.g., to represent the XYZ-wing of the Sudoku [14]. But we do not present here these rules for lack of space.

We now evaluate the size of the 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	$n + \frac{n^2(n-1)}{2}$	$\mathcal{O}(n^3)$
$[SAT-Om]$ $\forall m \in \{1..n-1\}$	$3n \sum_{m=1}^{n-1} \binom{n}{m}^2 (n-m)m$	$\mathcal{O}(n^{2n})$
$[SAT-OIm] \forall m \in \{2, 3\}$	$2n^3 (6n^2 - 2n^{\frac{3}{2}} - 9n + 5)$	$\mathcal{O}(n^5)$
$[SAT-SI2m.2m]$ with $m = 2$	$16n^3 (n^4 - 3n^3 - 8n^2 + 30n^{\frac{3}{2}} - 26n + 6\sqrt{n})$	$\mathcal{O}(n^7)$

To encode Sudoku problem of size 9, the minimum number of clauses is 17829 (definition of the variables and $[SAT-O1]$) whereas encoding all the rules generates approximately 76×10^6 clauses.

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.

5 Related Work

Global constraints in CSP Recent works deal with the combination of several global constraints. [16] presents some filtering algorithms for the sequence constraint and some combinations of sequence. [17] studied the conjunction of open global cardinality constraints with specific restrictions. [11] 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.

However, 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 attempt at addressing a wider range of possible combinations of Alldiff.

From the modeling point of view, [14] 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 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, but we 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 [18, 9, 5, 2] to preserve consistency properties and induce efficient

unit propagation in SAT solvers. The specific encodings of usual global constraint has been also addressed, for instance cardinality [3, 13], Among [2] or Alldiff [10]. Here, our transformation is based on the reduction rules and extended to multiple connected Alldiff. As some of these works we proved that it is correct w.r.t. GAC.

6 Conclusion

We have defined a set of consistency rules for general Alldiff constraints that can be easily implemented in usual constraint solvers. Of course, more rules have been defined but cannot be presented here due to space limitations. An easy way to implement such rules is to use CHR [7] or to integrate new propagators in systems such as GeCode [8]. Preliminary results show that such rules may improve domain reduction efficiency and could be even more efficient in Gecode by scheduling more finely the application of constraint propagators.

References

1. K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
2. Fahiem Bacchus. Gac via unit propagation. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 2007.
3. Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2003.
4. Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Comput. Surv.*, 38(4):12, 2006.
5. E. Hebrard C. Bessiere and T. Walsh. Local consistencies in sat. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 400–407. Springer, 2003.
6. Khaled M. Elbassioni, Irit Katriel, Martin Kutz, and Meena Mahajan. Simultaneous matchings. In *Algorithms and Computation, 16th International Symposium, ISAAC 2005, Sanya, Hainan, China, December 19-21, 2005, Proceedings*, volume 3827 of *Lecture Notes in Computer Science*, pages 106–115. Springer, 2005.
7. T. Fruewirth and S. Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.
8. SWI-Prolog. <http://www.swi-prolog.org/>.
9. Ian Gent. Arc consistency in sat. Technical Report APES-39A-2002, University of St Andrews, 2002.
10. Ian P. Gent and Peter Nightingale. A new encoding of alldifferent into sat. In A. Frisch and I. Miguel, editors, *Proceedings 3rd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, CP2004, Toronto, Canada*, pages 95–110, 2004.

11. Jean-Charles Régin and Carla P. Gomes. The cardinality matrix constraint. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP 2004)*, pages 572–587, 2004.
12. J.C. Régin. A filtering algorithm for constraint of difference in csp. In *National Conference of Artificial Intelligence*, pages 362–367, 1994.
13. João P. Marques Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 483–497. Springer, 2007.
14. Helmut Simonis. Sudoku as a constraint problem. In *Proceedings of the Fourth CP international Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 17–27, 2005.
15. W.-J. van Hoesve and I. Katriel. *Handbook of Constraint Programming*, chapter Global Constraints. Elsevier, 2006.
16. Willem-Jan van Hoesve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. New filtering algorithms for combinations of among constraints, 2008. Under review, Extended version of the CP 2006 paper.
17. Willem Jan van Hoesve and Jean-Charles Régin. Open constraints in a closed world. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Third International Conference, CPAIOR 2006, Cork, Ireland, May 31 - June 2, 2006, Proceedings*, volume 3990 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2006.
18. Toby Walsh. Sat v csp. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.