An Alldifferent Constraint Solver in SMT

Milan Banković and Filip Marić
Faculty of Mathematics, University of Belgrade
(milan|filip)@matf.bg.ac.rs

May 16, 2010

Abstract

The finite domain `alldifferent` constraint, requiring that all given variables have different values, is one of the fundamental global constraints in constraint programming (CP). Many filtering algorithms for `alldifferent` have been developed and successfully used in CP. Combining these with state-of-the-art SAT solvers is a promising research direction. Support for `alldifferent` within SMT solvers can make them suitable for solving constraint satisfaction and optimization problems. We formulate `alldifferent` as a first order theory and describe a DPLL($\mathcal{T}$) solver for it. The solver relies on Régin's filtering algorithm but introduces a novel algorithm for explaining propagations and conflicts. A prototype implementation has been made and tested on large Sudoku instances with encouraging preliminary results.

1 Introduction

In recent years, integration of techniques used in *constraint programming (CP)* with *satisfiability (SAT) solving* has been proposed as a very promising research direction, since it can employ good sides of both ([5]). On one hand, CP includes specialized algorithms that are well adapted to particular constraints of interest, and on the other hand, modern SAT solvers use sophisticated techniques and strategies during the search process and incorporate efficient implementation techniques and data structures. Detailed survey of CP can be found in [17] and of SAT solving in [4].

One of the most important and best studied global constraints in CP is the `alldifferent` constraint. It is a global constraint defined over some finite set of variables requiring that all its variables take different values from their finite domains. Many combinatorial problems can be expressed in terms of `alldifferent`. Its wide application area includes *puzzle solving*, *scheduling*, *combinatorial design problems*, etc. Various solving techniques have been developed and they usually include different forms of *consistency maintaining* (*domain filtering*), *constraint propagation*, and *search*.

One possible way of integration of CP and SAT is to express a global constraint as a first order theory, and then use *Satisfiability Modulo Theory (SMT)* checking. In this paper we apply this approach to the `alldifferent` constraint. The `alldifferent` constraint is formulated as an SMT theory and a decision procedure for it, based on the DPLL($\mathcal{T}$) framework ([13]), is developed. The procedure relies on a well-known filtering algorithm introduced by Régin ([16]). However, the algorithm is extended with a novel conflict and propagation explaining procedure, which is our main contribution. In this paper we focus on developing a solver for theories involving single `alldifferent` constraints — problems that involve multiple `alldifferent` constraints can be handled by combining several such solvers (which is, for example, the case in our experimental evaluation). We do not discuss algorithm correctness here, but it is covered in the draft longer version of the paper (`http://www.matf.bg.ac.rs/~milan/smt2010/`).

Related work. Different forms of consistency maintaining for `alldifferent` were considered and filtering algorithms based on these have been practically applied in constraint solving ([10]). The first complete algorithm for solving finite-domain `alldifferent` that maintaines generalised arc-consistency was developed by Régin 1994. ([16]). Much effort is currently being invested in implementation efficiency of these algorithms ([9]). There were also some efforts in extending them with explanation generating facilities ([8]).

The `alldifferent` over infinite or very large domains is also considered ([14]). For instance, an extension of SAT that maintains the `alldifferent` constraint over bitvectors is described in [3], but the procedure does not perform constraint propagation.

Combining finite-domain `alldifferent` with SMT was recently listed as one of the challenges in modern SMT solving ([12]), introducing the need for extending procedures used for the `alldifferent` in constraint programming (e.g. Régin's algorithm) with the procedures for generating explanations. As we are aware of, this idea was not yet employed in SMT and no DPLL($\mathcal{T}$) compliant decision procedure has been published yet.

Outline of the paper. In Section 2, the `alldifferent` constraint and SMT solvers based on the DPLL($\mathcal{T}$) architecture are described. In Section 3 the `alldifferent` theories, i.e. first order theories expressing single `alldifferent` constraints are introduced. In Section 4 our DPLL($\mathcal{T}$) solver for an `alldifferent` theory is described (conflict detection is addressed in Section 4.1, theory propagations in Section 4.2, and our novel algorithm for conflict and propagation explaining in Section 4.3). Section 4.4 considers combining multiple `alldifferent` theory solvers into one composite solver for problems involving more than one `alldifferent` constraints. Our implementation and its experimental evaluation are described in Section 5. Conclusions and some possible directions for further work are given in Section 6.

2 Background

The `alldifferent` constraint.

Definition 1. *Let $x_1, \ldots, x_n$ be variables with respective finite domains $D(x_1), \ldots, D(x_n)$. Then,*

$$\texttt{alldifferent}(x_1, \ldots, x_n) = \{(d_1, \ldots, d_n) \mid d_i \in D(x_i),\ d_i \neq d_j \text{ for } i \neq j\}$$

Many CSP problems can be expressed in terms of `alldifferent` constraints. For example, the well-known *Sudoku* problem (for a dimension n) can be represented using `alldifferent` by introducing a variable x_{ij} for each row i and column j of the $n^2 \times n^2$ grid. Each variable takes values from the domain $D = \{1, 2, \ldots, n^2\}$ and the equality $x_{ij} = d$ holds iff the field (i, j) on the grid is filled with a value d. The set of constraints is:

$$\begin{array}{l}
\text{for each } i \in \{1, \ldots, n^2\},\ \texttt{alldifferent}\left\{x_{i,j} \mid j \in \{1, \ldots, n^2\}\right\} \\
\text{for each } j \in \{1, \ldots, n^2\},\ \texttt{alldifferent}\left\{x_{i,j} \mid i \in \{1, \ldots, n^2\}\right\} \\
\text{for each } i, j \in \{1, \ldots, n\}, \\
\qquad\qquad \texttt{alldifferent}\left\{x_{n\cdot(i-1)+k, n\cdot(j-1)+l} \mid k, l \in \{1, \ldots, n\}\right\} \\
\text{for each given } (i_k, j_k, d_k), \quad x_{i_k, j_k} = d_k
\end{array}$$

SMT solvers and the DPLL($\mathcal{T}$) approach. *The Satisfiability Modulo Theory (SMT) problem* is a decision problem for logical formulas with respect to a background theory expressed in classical first-order logic with equality. Due to the lack of space, in this paper we do not define the basic notions of first order logic, but we assume the syntax and semantics used in [4]. The DPLL($\mathcal{T}$) *architecture* ([13]) is the dominant architecture of modern SMT solvers. A DPLL($\mathcal{T}$)-based SMT solver consists of a DPLL-based SAT solver that searches for propositional models satisfying a given formula and a theory solver ($\mathcal{T}$-solver) which is a specific decision procedure that checks for $\mathcal{T}$-satisfiability of the found (partial) propositional models. A theory solver usually should have the following functionality:

asserting and backtracking The solver must be able to assert and backtrack literals in a stack-like fashion (the stack is called the *trail*).

conflict detection The solver must be able to check the $\mathcal{T}$-satisfiability of conjunctions of asserted literals (incrementally, if possible).

conflict explanation If the conjunction of asserted literals is $\mathcal{T}$-unsatisfiable, the solver should be able to find a (preferably small) subset of asserted literals that is also $\mathcal{T}$-unsatisfiable.

theory propagation The solver should be able to find literals (from some fixed set) that are $\mathcal{T}$-consequence of the conjunction of asserted literals.

propagation explanation If a literal l is a $\mathcal{T}$-consequence of the conjunction of asserted literals, the solver should be able to find a (preferably small) subset of asserted literals that also $\mathcal{T}$-entails l.

3 The theory of `alldifferent`

In this section first order theories for `alldifferent` constraints are introduced. Consider the `alldifferent` problem $\mathcal{AD}$: `alldifferent`$(x_1, \ldots, x_n)$, where $x_i \in D(x_i)$, and $D = \bigcup_{i=1}^{n} D(x_i)$. For this problem instance, a first-order theory $\mathcal{T}_{\mathcal{AD}}$ is defined. The language $\Sigma^{\mathcal{AD}}$ of $\mathcal{T}_{\mathcal{AD}}$ consists of constant symbols $\overline{x}_i$ introduced for each variable x_i and $\overline{d}_i$ introduced for each value $d_i \in D$. All $\Sigma^{\mathcal{AD}}$ atomic formulae are ground equalities over constants from $\Sigma^{\mathcal{AD}}$. The theory consists of all models satisfying the following axioms:

- *Axioms of sanity*: $\overline{d}_i \neq \overline{d}_j$, for each two different constant symbols $\overline{d}_i$ and $\overline{d}_j$. These axioms ensure that different constant symbols for values of D represent different values from the variable domains.
- *Domain axioms*: $\bigvee_{d \in D(x)} \overline{x} = \overline{d}$ for each variable $x \in \mathcal{X}$. These axioms define domains of the variables. The meaning of the equality $\overline{x} = \overline{d}$ is that x takes the value d in the corresponding solution of $\mathcal{AD}$.
- *Axioms of difference*: $\overline{x}_i \neq \overline{x}_j$, for each i and j, where $1 \leq i < j \leq n$. These axioms ensure that the `alldifferent` constraint is satisfied.

Models $\mathcal{M} \in \mathcal{T}_{\mathcal{AD}}$ directly correspond to solutions of the problem $\mathcal{AD}$.

4 A solver for `alldifferent`

In this section, a DPLL($\mathcal{T}$) solver for $\mathcal{T}_{\mathcal{AD}}$ is presented. The solver is based on the reduction of `alldifferent` constraint to the matching problem in bipartite graphs. Conflict detection is reduced to constructing optimal matching which is done using the Hopcroft & Karp's algorithm [11]. Theory propagation detection is done using the Régin's algorithm [16]. However, for conflict and propagation explaining we present our novel approach based on the *minimal obstacle set problem*, also introduced in this paper.

4.1 Conflict detection

A *bipartite graph* $B = (U, V, E)$ is a graph consisting of the set of *left* vertices U, the set of *right* vertices V (disjoint with U) and the set of edges $E \subseteq U \times V$. A *matching* in a bipartite graph B is a set of edges $\mathbb{M} \subseteq E$ such that no two edges from $\mathbb{M}$ have a vertex in common. The matching $\mathbb{M}$ is *optimal* if there is no matching of greater cardinality. A matching $\mathbb{M}$ is a *covering matching* for U if every vertex $u \in U$ is matched with some vertex from V. A vertex is *free* if it is not matched with any vertex at the opposite side. The sets of free vertices from U and V will be denoted by $free(U)$ and $free(V)$ respectively.

The constraint `alldifferent`$(x_1, \ldots, x_n)$ can be assigned a bipartite graph $B = (U, V, E)$ (called its *value graph*) such that U contains one vertex

v^x for each variable $x \in \{x_1, \ldots, x_n\}$, and V contains one vertex v^d for each value $d \in \bigcup_{i=1}^{n} D(x_i)$. The edge (v^x, v^d) belongs to E iff $d \in D(x)$ (each variable is connected to the values from its domain).

The following proposition (given without a proof) establishes the correspondence between covering matchings in B and solutions of the `alldifferent` and implies that the `alldifferent` constraint is satisfiable if and only if there is a covering matching in its value graph.

Proposition 1. *A tuple* $(d_1, \ldots, d_n)$ *satisfies* `alldifferent`$(x_1, \ldots, x_n)$ *iff* $\mathbb{M} = \{(v^{x_i}, v^{d_i}) \mid 1 \leq i \leq n\}$ *is a covering matching in its value graph* B.

Therefore, the alldifferent problem is reduced to the optimal matching problem in the corresponding bipartite value graph. Indeed, if the cardinality of a constructed optimal matching is equal to n, then the matching is also a covering matching and it determines a solution satisfying the constraint. Otherwise, there are no covering matchings and, thus, the constraint is unsatisfiable.

An optimal matching in a bipartite graph can be constructed by starting from some existing (non-optimal) matching $\mathbb{M}$ (possibly empty) and incrementally extending it using *augmenting paths* until an optimal matching is obtained. For a bipartite graph B with a matching $\mathbb{M}$ an *alternating path* is a path that consists of edges that alternately belong to $\mathbb{M}$ and $E \setminus \mathbb{M}$. An *augmenting path* is an alternating path that begins at a free vertex on one side of the graph and ends at a free vertex on the other side. Augmenting paths can be detected by some BFS-based procedure applied to the directed graph $\mathit{directed}(B, \mathbb{M})$, obtained from the original graph B by orienting edges of $\mathbb{M}$ from left to right, and edges of $E \setminus \mathbb{M}$ from right to left. Each augmenting path can be used to increase the cardinality of the current matching by one — the current matching is replaced by its symmetric difference with the augmenting path. The matching is optimal if and only if there are no augmenting paths in B (a proof of this can be found in [10]). The runtime of the procedure depends on the difference between the number of edges in the initial matching and in the obtained optimal matching. Therefore, the procedure will run faster if the initial matching is close to optimal. This makes the procedure suitable for incremental applications. An improvement of the described procedure is the well-known Hopcroft & Karp's algorithm ([11]) that finds multiple disjoint augmenting paths in a single graph traversal.

In the context of the $\mathcal{T}_{\mathcal{AD}}$-solver, the algorithm explained above can be used for conflict detection. Assume that M is the set of currently asserted literals. First, the value graph B must be *synchronized* with M, i.e., for each literal $l \in M$, all edges from B that conflict with l should be removed (if $l \equiv \overline{x} \neq \overline{d}$, the edge (v^x, v^d) is removed, if $l \equiv \overline{x} = \overline{d}$, all edges $(v^x, v^{d'})$ where $d' \neq d$ are removed). Next, an optimal matching $\mathbb{M}$ is constructed. The set M is $\mathcal{T}_{\mathcal{AD}}$-satisfiable iff the matching is covering. The conflict check can be done incrementally — whenever some additional literals are asserted, edges

conflicting with those are removed and the optimal matching is only repaired (if necessary) and not constructed from scratch, which is significantly faster due to the nature of the used algorithm.

4.2 Theory propagation

Once a covering matching $\mathbb{M}$ is found, it can be used for detecting edges that belong to all covering matchings (*vital edges*) or edges that do not belong to any covering matching (*inconsistent edges*). These edges are used for the theory propagation. Edges that belong to some, but not all optimal matchings are called *alternating edges*.

Inconsistent edges should be removed from the value graph, since they are not part of any covering matching (and, therefore, their corresponding equalities are not a part of any solution). Removal of such inconsistent edges is called *filtering*. When all inconsistent edges are filtered out, the constraint is *hyper-arc consistent*. Vital edges must not be removed from the graph, since they are a part of every covering matching (and, therefore, their corresponding equalities are a part of every solution).

Detecting vital and inconsistent edges can be reduced to detecting alternating edges. Namely, each edge that is not alternating is vital if it belongs to the already constructed optimal matching (because it then belongs to all optimal matchings) and is inconsistent otherwise. The following Theorem 1 (a proof can be found in [2]) gives a characterization of alternating edges that can be used for their detection.

Theorem 1. *Let B be a bipartite graph with a covering matching $\mathbb{M}$. An edge is* alternating *iff it either belongs to some simple alternating path of an even length that starts at some free right vertex, or it belongs to some simple alternating cycle of an even length.*

The premises about the even length of paths and cycles are not crucial since every alternating cycle in a bipartite graph is of an even length and every simple alternating path starting at a free right vertex can be extended an even length (providing $\mathbb{M}$ is covering). Therefore, it is sufficient to consider arbitrary alternating paths and cycles.

Alternating edges can be found by *Régin's filtering algorithm* [16], which relies directly on the Theorem 1 (and, thus, requires a covering matching in B). Alternating paths in B are found by a BFS traversal of the graph $directed(B, \mathbb{M})$ starting from $free(V)$. All edges beloning to those paths are alternating. Alternating cycles are detected by finding the *strongly connected components* of $directed(B, \mathbb{M})$. A strongly connected component in a directed graph is a maximal subset of vertices such that its every two vertices are mutually reachable. All edges with their both vertices belonging to the same strongly connected belong to some alternating cycle. Strongly connected components can be found by Tarjan's DFS-based algorithm ([18]).

In the context of the $\mathcal{T}_{\mathcal{AD}}$-solver, the explained algorithm can be used for exhaustive theory propagation. Assume that the value graph B is already synchronized with the set of asserted literals M and that no conflict is detected (i.e., the constructed matching $\mathbb{M}$ is covering). After the execution of the Régin's filtering algorithm, each inconsistent edge (v^x, v^d) is removed from the value graph B and the disequality $\overline{x} \neq \overline{d}$ is propagated, since x cannot be equal to d in any solution for the current graph B (thus, $M \vDash_{\mathcal{T}_{\mathcal{AD}}} \overline{x} \neq \overline{d}$). For each vital edge (v^x, v^d) the equality $\overline{x} = \overline{d}$ is propagated, since x is equal to d in every solution for the current graph B (thus, $M \vDash_{\mathcal{T}_{\mathcal{AD}}} \overline{x} = \overline{d}$).

4.3 Conflict and propagation explaining

Our explanation generating procedure is based on reduction to a graph problem that we call the *minimal obstacle set problem* introduced in the following paragraph along with an algorithm which can be used for its solving.

Minimal Obstacle Set (MOS) Problem. Let $G = (V, E)$ be a directed graph, with a set of *final vertices* $F \subseteq V$ and a set of *obstacles* $O \subseteq E$. We say that O *separates* the vertex $v \in V$ from F if every path from v to any vertex $f \in F$ contains at least one edge from O. We also say that the vertex $v \in V$ is *blocked* (or *O-blocked*). If a vertex is not blocked, it is called *unblocked* (or *O-unblocked*). The set of vertices $W \subseteq V$ is *O-blocked* if each vertex w from W is O-blocked (in this case we also say that O *separates* W from F). Otherwise, W is *O-unblocked*. The path that contains no obstacles from O is called an *obstacle-free* (or *O-free*) path.

The *minimal obstacle set (MOS) problem* is defined as follows: given a set of *start vertices* $S \subseteq V$ that is O-blocked, find an obstacle set $O_{min} \subseteq O$ such that S remains O_{min}-blocked and O_{min} is minimal in sense of inclusion. Such obstacle set is called a *minimal obstacle set*.[1] It is clear that such set does not have to be unique.

The algorithm for solving the problem is based on the following theorem.

Theorem 2. *A set of obstacles O is minimal if and only if for each obstacle $e = (v, w) \in O$ the vertex v is reachable from S via some obstacle-free path and the vertex w is unblocked.*

Proof. Suppose that O is a minimal obstacle set. We prove that for each edge $e = (v, w) \in O$ the vertex v is reachable from S via some O-free path, and that the vertex w is O-unblocked. Assume the opposite, i.e., assume that for some edge $e \in O$ there is either no O-free path from vertices of S to v, or there is no O-free path from w to vertices of F. Then the set $O' = O \setminus \{e\}$ also separates S from F, since there is no O'-free path from S to F. This contradicts that O is minimal.

[1] An obstacle set O that separates W from F is actually a *cut* in G such that vertices from W and F are at the opposite sides of the cut. The minimal obstacle set problem can be understood as finding a sub-cut of a given cut that is minimal in the sense of inclusion.

Next, suppose that for each edge $(v, w) \in O$ the vertex v is reachable from S via some obstacle-free path and the vertex w is unblocked. We prove that O is a minimal obstacle set. Assume the opposite, i.e., assume that some $O' \subset O$ also separates S from F. Then, there is an edge $e = (v, w) \in O \setminus O'$ such that v is reachable from some $u \in S$ via an O-free path, and some vertex $f \in F$ is reachable from w via an O-free path. Therefore, since $e \notin O'$, there is an O'-free path from $u \in S$ to $f \in F$. This contradicts the fact that O' separates S from F. □

In the first stage of the algorithm, the the set $O_r \subseteq O$ composed of obstacles $e = (v, w) \in O$ such that v is reachable (via an O-free path) from S is constructed. This can be done by BFS traversal starting from the vertices of S. The set O_r also separates S from F.

In the second stage, the the set O_{min} of all obstacles $e = (v, w) \in O_r$ such that the vertex w is O_r-unblocked is constructed. For this purpose, it suffices to find the set $V_u \subseteq V$ containing all O_r-unblocked vertices. Consider the strongly connected components of G, by using only O_r-free paths (i.e. with the set of edges $E \setminus O_r$). For each component W from G it hold that if one of its vertices is O_r-unblocked, then all its vertices are O_r-unblocked (due to mutual reachability of vertices from the same component). In other words, being O_r-unblocked is a property of a component, not of a vertex itself. Therefore, the set V_u can be found by the DFS-based procedure similar to the Tarjan's algorithm for finding strongly connected components [18]. There are two main differences. First, the connectivity assumes using only O_r-free paths. Second, the algorithm does not return strongly connected components but returns the set V_u consisting of the following vertices:

- vertices from F (since final vertices are trivially O_r-unblocked),
- vertices v such that an edge $(v, w) \notin O_r$ is reached during the traversal and w is already in V_u,
- all vertices of strongly connected components with their *roots*[2] already belonging to V_u.

Finally, an obstacle $e = (v, w) \in O_r$ is added to O_{min} iff $w \in V_u$. According to Theorem 2, such obstacle set O_{min} is minimal.

Using MOS for finding explanations. Propagation explaining can be reduced to the MOS problem in the following way. Assume that M is the set of asserted literals such that $M \vDash_{\mathcal{T}_{\mathcal{AD}}} l$ and $M \nvDash_{\mathcal{T}_{\mathcal{AD}}} \bot$ (M consists of asserted literals at the point when $\mathcal{T}_{\mathcal{AD}}$-entailment of a literal l is detected by the Régin's filtering algorithm). Assume also that the value graph initially assigned to the `alldifferent` constraint is denoted by B^{init} and that the value graph after its synchronization with M is denoted by B^{curr}. The set of

[2]The *root* of a strongly connected component W is the vertex from W first visited during the traversal. See [18] for details.

edges removed during the synchronization with M is denoted by E^{rm}. Since $M \not\models_{\mathcal{T}_{\mathcal{AD}}} \bot$, there is a covering matching $\mathbb{M}$ in B^{curr} (we can assume the same covering matching $\mathbb{M}$ in B^{init}, because it is a superset of B^{curr}). Since $M \models_{\mathcal{T}_{\mathcal{AD}}} l$ and $\emptyset \not\models_{\mathcal{T}_{\mathcal{AD}}} l$, the edge e corresponding to l is alternating in B^{init} and is vital or inconsistent in B^{curr}. According to Theorem 1, each simple alternating path in B^{init} that starts at a free right vertex and that contains e also contains an edge from E^{rm}. The same holds for alternating cycles in B^{init} containing e. Assume that $G = directed(B^{init}, \mathbb{M})$, and that (u, v) is the directed edge in G that corresponds to e in B^{init}. The vertex u is now reachable in G from $free(V)$ or from v only via paths that contain at least one edge from E^{rm}. In terms of the MOS problem, the set of start vertices $S = free(V) \cup \{v\}$ is separated from the set of final vertices $F = \{u\}$ by the set of obstacles $O = E^{rm}$. Assume that the algorithm for the MOS problem returns the set O_{min}. The explanation of l is the subset of literals from M that caused the removal of edges from O_{min} during synchronization. Such explanation is minimal in the sense of inclusion.

When conflict explaining is concerned, assume that M is the set of asserted literals at the point when conflict is detected ($M \models_{\mathcal{T}_{\mathcal{AD}}} \bot$) and that $\mathbb{M}$ is an optimal non-covering matching found in B^{curr}. If the theory is consistent (i.e., $\emptyset \not\models_{\mathcal{T}_{\mathcal{AD}}} \bot$), then the matching $\mathbb{M}$ can be augmented in B^{init}, but this cannot be done in B^{curr}. This means that each augmenting path in B^{init} contains at least one edge from E^{rm}. In terms of the MOS problem, the start vertices $S = free(V)$ are separated from the final vertices $F = free(U)$ in the graph $G = directed(B^{init}, \mathbb{M})$ by the obstacle set $O = E^{rm}$. Assume that the algorithm for the MOS returns the set O_{min}. The explanation for the conflict is the set of literals from M that caused the removal of edges from O_{min} during synchronization.

4.4 Combining multiple `alldifferent` constraints

The described solver considers only the theory defined in Section 3, with only one `alldifferent` constraint involved. The case of multiple `alldifferent` constraints can be handled by combining separate theories (one theory per constraint) with overlapping signatures using the approach similar to *Delayed Theory Combination* [6]. One $\mathcal{T}_{\mathcal{AD}}$ solver is instantiated for each `alldifferent` constraint and they are combined into one composite solver that acts like a proxy — it establishes the communication between $\mathcal{T}_{\mathcal{AD}}$ solvers and the underlying SAT solver by delegating the interface calls to appropriate $\mathcal{T}_{\mathcal{AD}}$ solver(s). $\mathcal{T}_{\mathcal{AD}}$ solvers do not communicate with each other directly, but only via the SAT solver.

5 Implementation and Experimental Evaluation

The `alldifferent` solver described in this paper is implemented in our system `argoalldiff`[3] (implemented in `C++`). The system uses our SAT solver `argosat`. All presented algorithms are fully implemented, with some additional optimizations — for instance, a layered approach is used, detecting some trivial conflicts and propagations by lighter procedures before calling expensive algorithms here presented (their calls are delayed until necessary).

The preliminary experimental evaluation was performed on a set of randomly generated Sudoku instances and Table 1 presents the results for 200 instances of dimension 5 (i.e., 25×25 boards). The boards are randomly generated with around 40% fields filled in — Sudoku exhibits phase transition and these boards tend to be the hardest ones. We have compared our system with several alternative approaches:

1. the constraint solver Minion ([7]);
2. our SAT solver `argosat`, using a naive direct encoding where `alldifferent` is encoded by quadratic number of clauses encoding pairwise disequalities;
3. the SMT solver YICES (`http://yices.csl.sri.com/`), where `alldifferent` is encoded using the `distinct` predicate of SMT-LIB ([15]);
4. the SMT solver YICES, using encoding in the theory of equality with uninterpreted functions (EUF) where `alldifferent`$(x_1, \ldots, x_n)$ is encoded by introducing a fresh uninterpreted function symbol f and the constraint $f(x_1) = 1 \ \wedge \ \ldots \ \wedge \ f(x_n) = n$.
5. the modified version of `argoalldiff` which does not use our MOS explanation procedure, but always uses the full trail for explanations.

Each solver was given 120 seconds for each instance. Obtained results are shown in Table 1 and indicate that `argoalldiff` shows the best overall performance. It is worth mentioning that MINION failed to solve some instances (solved by `argoalldiff` within 120 seconds) even if the cutoff time is increased to 30 minutes.

Next several statistics can be used to indicate effectiveness of our procedures. Average number of conflicts per instance was around 460 with `argoalldiff`, and around 33000 with `argosat`. This shows that, although advanced `alldifferent` algorithms take much more time then efficient SAT conflict detection and propagation procedures (based on so called two-watched literal scheme), the reduction in the search space is so big that it

[3]The system is available under *GNU/GPL* license on `http://argo.matf.bg.ac.rs`.

Solver	Solved instances	Average time on solved instances
`argoalldiff` (with expl.)	194	8.8s
MINION	174	10s
`argosat`	172	18s
`argoalldiff` (without expl.)	169	18s
YICES (using `distinct`)	0	-
YICES (using EUF)	0	-

Table 1: Experimental results for Sudoku instances. All experiments were performed on Intel Pentium Dual Core T3200 2.00GHz, with 2GB RAM.

pays off to employ heavy filtering algorithms. The number of theory propagations was around 69000. Average size of explanation clauses for theory propagations was around 110 and for conflicts around 51. This can be compared to the version of `argoalldiff` that does not use our MOS explanation procedure — the average size of explanation clauses for theory propagations was around 590, and for conflicts was around 410. This shows that our explanation procedure reduces the explanation size to around 27% in the propagation explanation case, and to only 9% in the conflict explanation case. What is also important, profiling shows that most runtime is spent in the Hopcroft & Karp's and Régin's algorithm, and the MOS conflict explanation procedure takes less then 3% of the overall runtime.

6 Conclusions and Further Work

We have formulated the `alldifferent` constraint as an SMT theory and developed a DPLL($\mathcal{T}$)-based solver for it. The decision procedure relies on the Régin's filtering algorithm. However, the main contribution of our work is a novel conflict and propagation explanation procedure. The preliminary experimental results are very encouraging since our prototype implementation is much better of existing ways of encoding the `alldifferent` constraint in SMT (e.g., in the EUF theory) and it also outperforms MINION— one of the leading CSP solvers for `alldifferent`.

Our major future plans are to further improve the implementation of our system by incorporating techniques suggested in [9] and apply it to some real-world optimization problems (e.g., course timetabling).

References

[1] C. Barrett. R. Sebastiani. S. A. Seshia. C. Tinelli. Satisfiability Modulo Theories. *Handbook of Satisfiability*. 2009.

[2] C. Berge. *Graphe et Hypergraphes*. Dunod. 1970.

[3] A. Biere. R. Brummayer. Consistency checking of all different constraints over bit-vectors within a SAT solver. *FMCAD*. 2008.

[4] A. Biere. M. Heule. H. van Maaren. T. Walsh. *Handbook of Satisfiability*. IOS Press. 2009.

[5] L. Bordeaux. Y. Hamadi. L. Zhang. Propositional Satisfiability and Constraint Programming: A Comparative Survey. *ACM Comput. Surv. 38 (4)*. 2006.

[6] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P. van Rossum, R. Sebastiani. Efficient Satisfiability Modulo Theories via Delayed Theory Combination. *CAV*. LNCS 3576. 2005.

[7] I. Gent, C. Jefferson, I. Miguel. MINION: A Fast, Scalable, Constraint Solver. *ECAI 2006*. 2006.

[8] I. Gent. I. Miguel. N. Moore. Lazy Explanations for Constraint Propagators. *PADL*. LNCS 5937. 2010.

[9] I. Gent. I. Miguel. P. Nightingale. Generalised arc consistency for the AllDifferent constraint: An empirical survey. *Artificial Intelligence 172(18)*. 2008.

[10] W. van Hoeve. The Alldifferent Constraint: A Survey. *6th Annual Workshop of the ERCIM Working Group on Constraints*. 2001.

[11] J. E. Hopcroft. R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing 2 (4)*. 1973.

[12] R. Nieuwenhuis. A. Oliveras. E. Rodriguez-Carbonell. A. Rubio. Challenges in Satisfiability Modulo Theories. *Term Rewriting and Applications*. LNCS 4533. 2007.

[13] R. Nieuwenhuis. A. Oliveras. C. Tinelli. Solving SAT and SMT from abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). *Journal of the ACM. 53(6)*. 2006.

[14] C. Quimper. T. Walsh. Beyond Finite Domains: The All Different and Global Cardinality constraints. In *11th CP*. LNCS 3709. 2004.

[15] S. Ranise, C. Tineli. The SMT-LIB Standard. `http://goedel.cs.uiowa.edu/smtlib/`

[16] J. Régin. A filtering algorithm for constraints of difference in CSPs. *12th AAAI*. 1994.

[17] F. Rossi. P. van Beek. T. Walsh. *Handbook of Constraint Programming*. Elsevier. 2006.

[18] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*. 1972.