Same-Relation Constraints

Christopher Jefferson[1], Serdar Kadioglu[2,⋆], Karen E. Petrie[3], Meinolf Sellmann[2,⋆], and Stanislav Živný[3]

[1] University of St Andrews, School of Computer Science, St Andrews, UK
[2] Brown University, Department of Computer Science, Providence, USA
[3] Oxford University, Computing Laboratory, Oxford, UK

Abstract. The ALLDIFFERENT constraint was one of the first global constraints [17] and it enforces the conjunction of one binary constraint, the not-equal constraint, for every pair of variables. By looking at the set of all pairwise not-equal relations at the same time, AllDifferent offers greater filtering power. The natural question arises whether we can generally leverage the knowledge that sets of pairs of variables all share the same relation. This paper studies exactly this question. We study in particular special constraint graphs like cliques, complete bipartite graphs, and directed acyclic graphs, whereby we always assume that the *same* constraint is enforced on all edges in the graph. In particular, we study whether there exists a tractable GAC propagator for these global Same-Relation constraints and show that AllDifferent is a huge exception: most Same-Relation Constraints pose NP-hard filtering problems. We present algorithms, based on AC-4 and AC-6, for one family of Same-Relation Constraints, which do not achieve GAC propagation but outperform propagating each constraint individually in both theory and practice.

1 Motivation

The ALLDIFFERENT constraint was one of the first global constraints [17] and it enforces the conjunction of one binary constraint, the not-equal constraint, for every pair of variables. By looking at the set of all pairwise not-equal relations at the same time, AllDifferent offers greater filtering power while incurring the same worst-case complexity as filtering and propagating the effects of not-equal constraints for each pair of variables individually. The natural question arises whether we can leverage the knowledge that sets of pairs of variables all share the same relation in other cases as well. We investigate in particular binary constraint satisfaction problems (BCSPs) with special associated constraint graphs like cliques (as in AllDifferent), complete bipartite graphs (important when a relation holds between all variables X in a subset of I and Y in J), and directed acyclic graphs (apart from bounded tree width graphs the simplest generalization of trees), whereby we always assume that the *same* constraint is enforced on all edges in the graph. We refer to the conjunction of the same binary relation over any set of pairs in a BCSP as a *Same-Relation Constraint*.

⋆ This work was supported by the National Science Foundation through the Career: Cornflower Project (award number 0644113).

I.P. Gent (Ed.): CP 2009, LNCS 5732, pp. 470–485, 2009.

2 Theory Background

We study the complexity of achieving GAC on binary CSPs with one same-relation constraint. The classes of structures considered are all constraint graphs as defined in [12]. The ultimate goal is to classify, for a given constraint graph, which same-relation constraints admit a polynomial-time GAC, and which do not. It is well known that the CSP is equivalent to the HOMOMORPHISM problem between relation structures [8]. Most theoretical research has been done on the case where either the domain size or the arities of the constraints are bounded.

We deal with the problem where both the domain size and the arities of the constraints are unbounded (so-called *global constraints*). Since we are interested in the complexity of achieving GAC, we allow all unary constraints. In mathematical terms, we study the LIST HOMOMORPHISM problem. For a CSP instance $\mathcal{P}$, achieving GAC on $\mathcal{P}$ is equivalent to checking solvability of $\mathcal{P}$ with additional unary constraints [3]. Note that showing that a problem does not have a polynomial-time decision algorithm implies that this problem also does not have a polynomial-time algorithm for achieving GAC, which is a more general problem.

Generalizing the result of Freuder [10], Dalmau et al. showed that CSPs with bounded tree-width modulo homomorphic equivalence are solvable in polynomial time [5]. Grohe showed[1] that this is the only tractable class of bounded arity, defined by structure [13]. In other words, a class of structures with unbounded tree-width modulo homomorphic equivalence is not solvable in polynomial time.

In the case of binary CSPs, we are interested in classes of constraint graphs with a same-relation constraint. The first observation is that classes of graphs with bounded tree-width are not only tractable, but also permit achieving GAC in polynomial time (even with separate domains, and even with different constraints: different domains are just unary constraints which do not increase the tree-width). The idea is that such graphs have bounded-size separating sets, and the domains on these separating sets can be explicitly recorded in polynomial space (dynamic programming approach) [9,6,13]. Therefore, we are interested in classes of graphs with unbounded tree-width.

3 Clique Same-Relation

First we look at cliques. The famous ALLDIFFERENT constraint is an example of a same-relation constraint which, if put on a clique, is tractable – in case of AllDifferent, because the microstructure is perfect [18] and also has a polynomial-time GAC [17].

Definition 1. *Given a set of values D and a set of variables $\{X_1, \ldots, X_n\}$, each associated with its own domain $D_i \subseteq D$, and a binary relation $R \subseteq D \times D$, an assignment $\sigma : \{X_1, \ldots, X_n\} \to D$ satisfies the* Clique Same-Relation Constraint *CSR on the relation R if and only if for all i and j such that $1 \leq i, j \leq n, i \neq j$, it holds that $(\sigma(X_i), \sigma(X_j)) \in R$.*

[1] Assuming a standard assumption from parameterized complexity theory FPT $\neq$ W[1], see [7] for more details.

3.1 Complexity of Achieving GAC

Despite the tractability of AllDifferent, in general enforcing the same binary relation over all pairs of variables of a CSP yields a hard filtering problem.

Theorem 1. *Deciding whether CSR is satisfiable for an arbitrary binary relation is NP-hard.*

Proof. We reduce from the CLIQUE problem. Assume we are given an undirected graph $G = (V, E)$ and a value $k \in \mathbb{N}$ and need to decide whether G contains a clique of size k. We construct a CSP with just one CSR constraint in the following way. We introduce k variables $X_1, \ldots, X_k$, each associated with domain V. The relation R is defined as $R \leftarrow \{(a, b) \in V^2 \mid a \neq b, \{a, b\} \in E\}$. We claim that CSR on the relation $(X_1, \ldots, X_k, R)$ is satisfiable if and only if G contains a clique of size k.

"⇒" Assume there is an assignment $\sigma : \{X_1, \ldots, X_k\} \to V$ that satisfies CSR. Then, $C \leftarrow \{\sigma(X_1), \ldots, \sigma(X_k)\} \subseteq V$ is a clique because CSR enforces R for each pair of variables, and thus that there exists an edge between all pairs of nodes in C. Furthermore, $|C| = k$ since R forbids that the same node is assigned to two different variables.

"⇐" Now assume there exists a clique $C = \{v_1, \ldots, v_k\} \subseteq V$ with $|C| = k$. Setting $\sigma(X_i) \leftarrow v_i$ gives a satisfying assignment to CSR because for all $i \neq j$ we have that $(\sigma(X_i), \sigma(X_j)) = (v_i, v_j) \in E$ with $v_i \neq v_j$, and thus $(\sigma(X_i), \sigma(X_j)) \in R$. □

Corollary 1. *Achieving GAC for the CSR is NP-hard.*

In fact, we can show more: Even when we limit ourselves to binary symmetric relations which, for each value, forbid *only one other value*, deciding the satisfiability of the CSR is already intractable. This shows what a great exception AllDifferent really is. Even the slightest generalization already leads to intractable filtering problems.

Theorem 2. *Deciding CSR is NP-hard even for relations where each value appears in at most one forbidden tuple.*

Proof. We reduce from SAT. Given a SAT instance with k clauses over n variables, we consider an instance of CSR with k variables, each corresponding to one clause. Let D be $\{\langle 1, T\rangle, \langle 1, F\rangle, \ldots, \langle n, T\rangle, \langle n, F\rangle\}$. We define $R \subseteq D \times D$ to be the binary symmetric relation which forbids, for every $1 \leq i \leq n$, the set of tuples $\{\langle\langle i, T\rangle, \langle i, F\rangle\rangle, \langle\langle i, F\rangle, \langle i, T\rangle\rangle\}$. Note that R is independent of the clauses in the SAT instance.

Each clause in the SAT instance is encoded into the domain restriction on the corresponding variable. For instance, the clause $(x_1 \vee \neg x_2 \vee x_3)$ encodes as the domain $\{\langle 1, T\rangle, \langle 2, F\rangle, \langle 3, T\rangle\}$.

Any solution to this CSR instance, which can contain at most one of $\langle i, T\rangle$ and $\langle i, F\rangle$ for any $1 \leq i \leq n$, gives a solution to the SAT instance (as each variable must be assigned a literal in its clause). SAT variables which are not assigned a value can be given any value without compromising satisfiability. Analogously, a feasible assignment to the SAT formula maps back to a satisfying assignment to CSR in the same way: in any clause, take any of the literals in the solution which satisfy that clause and assign the variable that value. □

3.2 Restriction on the Size of Domain

Our proof that CSR is intractable required both an increasing number of variables and increasing domain size. The question arises whether the problem becomes tractable when the domain size is limited. The following shows that the CSR is indeed tractable when the domain size is bounded.

Lemma 1. *For a constraint CSR for a symmetric relation R, it is possible to check if an assignment satisfies the constraint given only:*

- *a promise that any domain value d such that $\langle d, d\rangle \notin R$, is used at most once, and*
- *the set S of domain values assigned.*

By ensuring that there are no two distinct values $s_1, s_2 \in S$ such that $\langle s_1, s_2\rangle \notin R$.

Proof. If CSR is violated, there must be two variables which do not satisfy R. This could occur either because two variables are assigned the same domain value d such that the assignment $\langle d, d\rangle$ is forbidden by R, or two variables are assigned different values d_1, d_2 such that the tuple $\langle d_1, d_2\rangle$ which do not satisfy R.

Lemma 1 provides a useful tool for characterizing the satisfying assignments to CSR, which we will use to devise a general filtering algorithm.

Theorem 3. *Achieving GAC for the CSR is tractable for bounded domains.*

Proof. Since the definition of CSR requires that the relation holds in both directions, it is sufficient to consider symmetric relations only. Then, Lemma 1 shows that satisfying assignments can be expressed by the set of allowed values and only using values d such that $\langle d, d\rangle$ is forbidden by R at most once. We shall show how given a CSR constraint, given a set of values S which satisfies Lemma 1 and a list of sub-domains for the variables in the scope of the constraint, we can find if an assignment with values only in S exists in polynomial time.

Given such a set of domain values S, we call values d such that $\langle d, d\rangle \in R$ *sink values*. Note that in any assignment which satisfies the CSR constraint and contains assignments only in S, changing the assignment of any variable to a sink value will produce another satisfying assignment. Therefore, without loss of generality, we can assume every variable which could be assigned any sink value in S is assigned such a value.

This leaves only variables whose domains contain only values which can occur at most once. This is exactly equivalent to an ALLDIFFERENT constraint, and can be solved as such.

Finally, note that for a domain of size d, there are 2^d subsets of the domain, and this places a very weak bound on the subsets of the domain which will satisfy the conditions of Lemma 1. Therefore, for any domain size there is a fixed bound on how many subsets have to be checked.

Theorem 3 shows that achieving GAC for the CSR is tractable for bounded domains, although the algorithm presented here is not practical. There are a number of simple

ways its performance could be improved which we will not consider here as we are merely interested in theoretical tractability.

An interesting implication of our result is the following. Consider a symmetric relation with at most k allowed tuples for each domain value; that is, given $R \subseteq D \times D$, we require that for each $d \in D$, $|\{\langle d, . \rangle \in R\}| \leq k$ for some k.

Corollary 2. *Let k be a bound on the number of allowed tuples involving each domain value in R. If k is bounded, then achieving GAC for the CSR is tractable. If k is unbounded, then solving CSR is NP-hard.*

Proof. If k is bounded, then after assigning a value to an arbitrary variable achieving GAC for the CSR reduces to the bounded domain case (see Theorem 3) . The unbounded case follows from Theorem 1. □

4 Bipartite Same-Relation

After studying complete constraint graphs in the previous section, let us now consider the complete bipartite case. This is relevant for CSPs where a set of variables is partitioned into two sets A and B and the same binary constraint is enforced between all pairs of variables possible between A and B.

Definition 2. *Given a set of values D and two sets of variables $A = \{X_1, \ldots, X_n\}$ and $B = \{X_{n+1} \ldots, X_m\}$, each associated with its own domain $D_i \subseteq D$, and a binary relation $R \subseteq D \times D$, an assignment $\sigma : \{X_1, \ldots, X_n\} \rightarrow D$ satisfies the* Bipartite Same-Relation Constraint *BSR on relation R if and only if $\forall\ X_i \in A, X_i \in B$ it holds that $(\sigma(X_i), \sigma(X_j)) \in R$.*

4.1 Complexity of Achieving GAC

At first, the BSR appears trivially tractable because once an allowed assignment is found between any pair of variables in both parts of the bipartite graph, these values can be assigned to all variables. Indeed, as any bipartite graph (with at least one edge) is homomorphically equivalent to a single edge, such CSPs instance are easy to solve [5,13] (using the fact that CSPs are equivalent to the HOMOMORPHISM problem).

However, we have to take into account that the domains of the variables may be different; in other words, unary constraints are present. This fact causes the problem of achieving GAC for the BSR to become intractable. In mathematical terms, instead of facing a HOMOMORPHISM problem (which is trivial on bipartite graphs), we must deal with the LIST-HOMOMORPHISM problem.

Theorem 4. *Deciding whether BSR is satisfiable is NP-hard.*

Proof. We reduce from the CSR satisfaction problem which we showed previously is NP-hard. Assume we are given the CSR constraint on relation R over variables $\{X_1, \ldots, X_n\}$ with associated domains $D_1, \ldots, D_n$. We introduce variables $Y_1, \ldots, Y_n$, $Z_1, \ldots, Z_n$ and set $A \leftarrow \{Y_1, \ldots, Y_n\}$, $B \leftarrow \{Z_1, \ldots, Z_n\}$. The domain of variables Y_i and Z_i is $\{(i, k) \mid k \in D_i\}$. Finally we define the relation P over the tuples $((i, k), (j, l))$ where $1 \leq i, j \leq n$ and either $i = j \wedge k = l$ or $i \neq j \wedge (k, l) \in R$. We claim that BSR on A, B and P is satisfiable if and only if CSR on R and the X_i is.

"⇒" Let σ denote a solution to BSR on A,B and P. For all i the initial domains and the definition of P imply that $\sigma(Y_i) = \sigma(Z_i) = (i, k_i)$ for some $k_i \in D_i$. Define $\tau : \{X_1, \ldots, X_n\} \rightarrow D$ by setting $\tau(X_i) \leftarrow k_i$. Let $1 \leq i, j \leq n$ with $i \neq j$. Then, since $((i, k_i), (j, k_j)) \in P$, $(\tau(X_i), \tau(X_j)) = (k_i, k_j) \in R$. And therefore, τ satisfies CSR for the relation R.

"⇐" Let τ denote a solution to CSR on R and the X_i. Then, σ with $\sigma(Y_i) \leftarrow \sigma(Z_i) \leftarrow (i, \tau(X_i))$ satisfies BSR on A, B and P. □

Corollary 3. *Achieving GAC for the BSR is NP-hard.*

5 DAG Same-Relation

In the previous sections we showed that achieving GAC for cliques and complete bipartite graphs is hard. Now we go on to show that a simple generalization of trees to directed graphs is intractable. When the binary relation that we consider is not symmetric, each edge in the constraint graph is directed. The generalization of trees (which we know are tractable) to the directed case then results in directed acyclic graphs (DAGs).

Definition 3. *Let D be a set of values, X be a set of variables $X = \{X_1, \ldots, X_n\}$ and G be a directed acyclic graph (DAG) $G = \langle X, A \rangle$. Each variable is associated with its own domain $D_i \subseteq D$. Given a binary relation $R \subseteq D \times D$, an assignment $\sigma : \{X_1, \ldots, X_n\} \rightarrow D$ satisfies the* DAG Same-Relation Constraint *DSR on relation R if and only if $\forall\, 1 \leq i, j \leq n$ such that $(i, j) \in A$, it holds that $(\sigma(X_i), \sigma(X_j)) \in R$.*

5.1 Complexity of Achieving GAC

Somewhat surprisingly, we find that even the simple graph structure of DAGs yields intractable filtering problems: bipartite graphs, with the orientation of all edges from one partition to the other, form a DAG. Therefore, Theorem 4 proves that solving DSR is NP-hard.

The question arises whether DAGs become tractable when we know that the direction on every arc is truly enforced by the constraints. Let us consider anti-symmetric relations. A relation R is anti-symmetric if for all a and b, $(a, b) \in R$ and $(b, a) \in R$ implies $a = b$. First we show that irreflexive antisymmetric relations can be NP-hard on DAGs.

Lemma 2. *Deciding satisfiability if DSR is NP-hard even for irreflexive antisymmetric relations.*

Proof. We use the equivalence between the CSP and the HOMOMORPHISM problem [8]. Solving an instance of DSR on relation R is equivalent to the question of whether there is a homomorphism[2] between the digraph A and digraph R. This problem is known as ORIENTED GRAPH COLORING [19]. The complexity of this problem

[2] A homomorphism between two directed graphs (digraphs) $G = \langle V(G), A(G) \rangle$ and $H = \langle V(H), A(H) \rangle$ is a mapping $f : V(G) \rightarrow V(H)$ which preserves arcs, that is, $(u, v) \in A(G)$ implies $(f(u), f(v)) \in A(H)$.

was studied in [15], and Swart showed that the ORIENTED GRAPH COLORING problem is polynomial-time solvable for R on at most 3 vertices, and NP-complete otherwise, even when restricted to DAGs [20]. Note this proves more that almost all asymmetric relations are NP-hard on DAGs. □

Note that it follows[3] from a recent result of Hell et al. that solving DSR is NP-hard also for reflexive antisymmetric relations [14].

6 Grid Same-Relation

Another interesting class of graphs are *grids*. For $m, n \geq 1$, the $(m \times n)$-grid is the graph with vertex set $\{(i,j) | 1 \leq i \leq m, 1 \leq j \leq n\}$ and an edge between two different vertices (i,j) and (i',j') if $|i-i'| + |j-j'| = 1$.

Definition 4. *Given a set of values D and a set of variables*

$$\{X_{1,1}, \ldots, X_{1,m}, \ldots, X_{m,1}, \ldots, X_{m,n}\},$$

each associated with its own domain $D_i \subseteq D$, and a binary relation $R \subseteq D \times D$, an assignment $\sigma : \{X_{1,1}, \ldots, X_{m,n}\} \to D$ satisfies the Grid Same-Relation Constraint *GSR on relation R if and only if for all distinct pairs of points (i,j) and (i',j') such that $|i-i'| + |j-j'| = 1$, it holds that $\sigma(X_{i,j}, X_{i',j'}) \in R$.*

Once more, we find:

Lemma 3. *Deciding satisfiability of GSR is NP-hard.*

Proof. We reduce from the CLIQUE problem. Let $\langle G, k \rangle$ be an instance of CLIQUE, where $G = \langle V, E \rangle$ is an undirected graph without loops and $V = \{1, \ldots, n\}$. The goal is to determine whether there is a clique of size k in G. We define an instance of GSR with variables $X_{1,1}, \ldots, X_{k,k}$ and a relation R. For every $1 \leq i \leq k$, the domain of $X_{i,i}$ is $\{\langle i, i, u, u \rangle \mid 1 \leq u \leq n\}$. For every $1 \leq i \neq j \leq k$, the domain of $X_{i,j}$ is $\{\langle i, j, u, v \rangle \mid \{u, v\} \in E\}$. We define the relation R as follows: $\{\langle i, j, u, v \rangle, \langle i', j', u', v' \rangle\}$ belongs to R if and only if the following two conditions are satisfied:

1. $i = i' \Rightarrow [(u = u') \,\&\, (v \neq v')]$
2. $j = j' \Rightarrow [(v = v') \,\&\, (u \neq u')]$

We claim that G contains a clique of size k if and only if GSR on the $X_{i,j}$ and R is satisfiable.

"⇒" Assume there exists a clique $C = \{v_1, \ldots, v_k\} \subseteq V$ in G with $|C| = k$. We claim that setting $\sigma(X_{i,i}) \leftarrow \langle i, i, v_i, v_i \rangle$ for all $1 \leq i \leq k$, and $\sigma(X_{i,j}) \leftarrow \langle i, j, v_i, v_j \rangle$ for all $1 \leq i \neq j \leq k$ gives a satisfying assignment to GSR. Let $X_{i,j}$ and $X_{i',j'}$ be two variables such that $|i-i'| + |j-j'| = 1$. Let $\sigma(X_{i,j}) = \langle i, j, u, v \rangle$ and $\sigma(X_{i',j'}) = \langle i', j', u', v' \rangle$. If $i = i'$, then $u = u'$ and $v \neq v'$ from the definition of σ. If $j = j'$, then $v = v'$ and $u \neq u'$ from the definition of σ. Hence in both cases, $\{\langle i, j, u, v \rangle, \langle i', j', u', v' \rangle\} \in R$.

[3] Private communication with A. Rafiey.

"⇐" Assume there is a solution σ to GSR on relation $(X_{1,1}, \ldots, X_{k,k}, R)$. From the definition of R, observe that for every fixed i there exists u_i such that $\sigma(X_{i,j}) = \langle i, j, u_i, . \rangle$ for every j. (In other words, the third argument of every row is the same.) Similarly, for every fixed j there exists v_j such that $\sigma(X_{i,j}) = \langle i, j, ., v_j \rangle$ for every i. (In other words, the fourth argument of every column is the same.) By these two simple observations, for every $1 \leq i \leq k$, there is no v and $j \neq j'$ such that $\sigma(X_{i,j}) = \langle i, j, u_i, v \rangle$ and $\sigma(X_{i,j'}) = \langle i, j', u_i, v \rangle$. (In other words, the fourth arguments of every row are all different.) Assume, for contradiction, that $\sigma(X_{i,j}) = \langle i, j, u_i, v \rangle$ and $\sigma(X_{i,j'}) = \langle i, j', u_i, v \rangle$ for some $v \neq u_i$ and $j' \neq j$; that is, the value v occurs more than once in the i-th row. But then $\sigma(X_{j',j'}) = \langle j', j', v, v \rangle$ as $X_{i,j'}$ and $X_{j',j'}$ are in the same column, and $\sigma(X_{j',j}) = \langle j', j, v, x \rangle$ as $X_{j',j}$ and $X_{j',j'}$ are in the same row.. But as $X_{i,j}$ and $X_{j',j}$ are in the same column, and $\sigma(X_{i,j}) = \langle i, j, u_i, v \rangle$, we get $x = v$. In other words, $\sigma(X_{j',j}) = \langle j', j, v, v \rangle$. But this is a contradiction as σ would not be a solution to GSR. Similarly, we can prove the same results for columns. Moreover, the same argument shows that if $\sigma(X_{i,j}) = \langle i, j, u, v \rangle$, then $\sigma(X_{j,i}) = \langle j, i, v, u \rangle$. Using this kind of reasoning repeatedly shows that there is a set of k different values $C = \{u_1, \ldots, u_k\}$ such that $\delta(X_{i,j}) = \langle i, j, u_i, u_j \rangle$. From the definition of R, C forms a clique in G. □

Corollary 4. *Achieving GAC for the GSR is intractable.*

7 Decomposing Same Relation Constraints

We proved a series of negative results for Same Relation Constraints (SRCs). Even for simple constraint graphs like DAGs and grids, SRCs pose intractable filtering problems. In this section, we investigate whether we can exploit the fact that the same relation is enforced on all edges of a constraint graph to achieve GAC on the corresponding binary CSP, where we consider the *collection* of individual binary constraints. This will achieve the same propagation as propagating each constraint in isolation, unlike for example the AllDifferent constraint [17], which propagates the conjunction constraint. However, by making use of the added structure of SRC, we will show how both theoretical and practical performance gains can be achieved. We begin with the clique same-relation constraint.

7.1 Decomposing CSR

Using AC-4 [16] or any of its successors to achieve GAC, we require time $O(n^2 d^2)$ for a network of n^2 binary constraints over n variables with domain size $|D| = d$. By exploiting the fact that the *same* relation holds for all pairs of variables, we can speed up this computation.

AC-4 Approach. We follow in principle the approach from AC-4 to count the number of supports for each value. The core observation is that a value l has the same number of supports $k \in D_i$ *no matter to which* D_j l *belongs*. Therefore, it is sufficient to introduce counters $supCount[i, l]$ in which we store the number of values in D_i which support

```
1: Init-CSR (X_1, ..., X_n, R)
2: for all l ∈ D do
3:     S_l ← {k ∈ D | (k, l) ∈ R}
4: end for
5: for all l ∈ D do
6:     for all 1 ≤ i ≤ n do
7:         supCount[i, l] ← |D_i ∩ S_l|
8:     end for
9: end for
```

Algorithm 1. Root-Node Initialization for the CSR Constraint

$l \in D_j$ for any $j \neq i$. In Algorithm 1 we show how these counters are initialized at the root by counting the number of values in the domain of each variable that supports any given value l.

In Algorithm 2, we show how to filter the collection of binary constraints represented by the CSR so that we achieve GAC on the collection. The algorithm proceeds in two phases. In the first phase, lines 2-12, we update the counters based on the values that have been removed from variable domains since the last call to this routine. We assume that this incremental data is given in $\Delta_1, \ldots, \Delta_n$. For each value l that has lost all its support in some domain D_i as indicated by the corresponding counter $supCount[i, l]$ becoming 0, we add the tuple (i, l) to the set Q. The tuple means that l has to be removed from all D_j where $j \neq i$. In the second phase, lines 13-26, we iterate through Q to perform these removals and to update the counters accordingly. If new values must be removed as a consequence, they are added to Q.

Lemma 4. *Algorithm 2 achieves GAC on the collection of binary constraints represented by the CSR constraint in time $O(nd^2)$ and space $O(nd)$ where n is the number of variables and d is the size of the value universe D.*

Proof.

- **GAC:** The method is sound and complete as it initially counts all supports for a value and then and only then removes this value when all support for it is lost.
- **Complexity:** The space needed to store the counters is obviously in $\Theta(nd)$, which is linear in the input when the initial domains for all variables are stored explicitly. Regarding time complexity, the dominating steps are step 6 and step 19. Step 6 is carried out $O(nd^2)$ times. The number of times Step 19 is carried out is $O(nd)$ times the number of times that step 13 is carried out. However, step 13 can be called no more than $2d$ times as the same tuple (i, l) cannot enter Q more than twice: after a value $l \in D$ has appeared in two tuples in Q it is removed from all variable domains. □

AC-6 Approach. As it is based on the same idea as AC-4, the previous algorithm is practically inefficient in that it always computes all supports for all values. We can improve the algorithm by basing it on AC-6 [1,2] rather than AC-4. Algorithms 3-5 realize this idea. In Algorithm 3 we show how to modify the initialization part. First,

```
 1: Filter-CSR (X_1, ..., X_n, R, Δ_1, ..., Δ_n)
 2: Q ← ∅
 3: for all i = 1 ... n do
 4:    for all k ∈ Δ_i do
 5:       for all l ∈ S_k do
 6:          supCount[i, l] ← supCount[i, l] − 1
 7:          if supCount[i, l] == 0 then
 8:             Q ← Q ∪ {(i, l)}
 9:          end if
10:       end for
11:    end for
12: end for
13: while Q ≠ ∅ do
14:    (i, l) ∈ Q, Q ← Q \ {(i, l)}
15:    for all j ≠ i do
16:       if l ∈ D_j then
17:          D_j ← D_j \ {l}
18:          for all k ∈ S_l do
19:             supCount[j, k] ← supCount[j, k] − 1
20:             if supCount[j, k] == 0 then
21:                Q ← Q ∪ {(j, k)}
22:             end if
23:          end for
24:       end if
25:    end for
26: end while
```

Algorithm 2. AC-4-based Filtering Algorithm for the CSR Constraint

the set of potential supports S_l for a value $l \in D$ is now an ordered tuple rather than a set. Second, support counters are replaced with support indices. The new variable $supIndex[i, l]$ tells us the index of the support in S_l which is currently supporting l in D_i. Algorithm 4 shows how to find a new support for a given value l from the domain of the variable i. The algorithm iterates through the ordered support list until it reaches to the end of it, line 6, in which case it returns a failure or until it finds a new support value k. Set-variable $T_{v,k}$ is used to store the set of values that are currently being supported by value k that is in the domain of variable v. In case a new support value l is found, $T_{v,k}$ is extended by the value l, line 10. These three new data structures, S_l, $supIndex[i, l]$ and $T_{v,k}$, allow us to quickly find values for which a new support needs to be computed, which can be done incrementally as a replacement support may only be found after the current support in the chosen ordering. This way, the algorithm never needs to traverse more than all potential supports for all values.

Finally, Algorithm 5 provides a general outline to our AC-6 based propagator which again works in two phases, similar to its AC-4 based counter-part. In the first phase, lines 2-12, we scan the values that have been removed from variable domains since the last call to this routine. We assume that this incremental data is given in $\Delta_1, \ldots, \Delta_n$. In line 6, we look for a new support for each value l that was previously supported by a

```
1: initSup (X_1, ..., X_n, R)
2: for all l ∈ D do
3:     S_l ← (k ∈ D | (k, l) ∈ R)
4: end for
5: T ← ∅
6: for all l ∈ D do
7:     for all 1 ≤ i ≤ n do
8:         supIndex[i, l] ← 0
9:         newSup(X_1, ..., X_n, R, i, l)
10:    end for
11: end for
```

Algorithm 3. Support Initialization for the CSR Constraint

value k from the domain of variable i, which is now lost. If the value l is not supported anymore, it is removed from the set $T_{i,k}$ and the tuple (i,l) is added to the queue which means that l has to be removed from all D_j where $j \neq i$. In the second phase, lines 13-26, we iterate through Q to perform these removals and to update the set variables $T_{j,l}$ accordingly. If new values must be removed as a consequence, they are added to Q.

7.2 Decomposing BSR

Analogously to our results on the CSR Constraint, we can exploit again the knowledge that there is the same relation on all edges in the complete bipartite constraint graph. Again, the time that AC-4 or AC-6 would need to achieve GAC on the collection of binary constraints that is represented by the BSR is in $O(n^2d^2)$. Following the same idea as for CSR, we can reduce this time to $O(nd^2)$.

We can devise an AC-6 based propagator for BSR in line with Algorithms 3-5. We can still use the set of potential supports S_l for a value which stores ordered tuples and the support indices $supIndex[i, l]$ which tell us the index of the support in S_l which is currently supporting l in D_i. The only required modification is that the set variable $T_{v,k}$ now has to be replaced with $T^A_{v,k}$ and $T^B_{v,k}$ to distinguish between the partitions of the constraint graph.

```
1: bool newSup (X_1, ..., X_n, R, i, l)
2: supIndex[i, l] ← supIndex[i, l] + 1
3: while supIndex[i, l] ≤ |S_l| and S_l[supIndex[i, l]] ∉ D_i do
4:     supIndex[i, l] ← supIndex[i, l] + 1
5: end while
6: if supIndex[i, l] > |S_l| then
7:     return false
8: else
9:     k ← S_l[supIndex[i, l]]
10:    T_{v,k} ← T_{v,k} ∪ {l}
11:    return true
12: end if
```

Algorithm 4. Support Replacement for the CSR Constraint

```
 1: Filter-CSR (X_1, ..., X_n, R, Δ_1, ..., Δ_n)
 2: Q ← ∅
 3: for all i = 1...n do
 4:    for all k ∈ Δ_i do
 5:       for all l ∈ T_{i,k} do
 6:          if !newSup(X_1, ..., X_n, R, i, l) then
 7:             Q ← Q ∪ {(i, l)}
 8:             T_{i,k} ← T_{i,k} \ {l}
 9:          end if
10:       end for
11:    end for
12: end for
13: while Q ≠ ∅ do
14:    (i, l) ∈ Q, Q ← Q \ {(i, l)}
15:    for all j ≠ i do
16:       if l ∈ D_j then
17:          D_j ← D_j \ {l}
18:          for all k ∈ T_{j,l} do
19:             if !newSup(X_1, ..., X_n, R, j, k) then
20:                Q ← Q ∪ {(j, k)}
21:                T_{j,l} ← T_{j,l} \ {k}
22:             end if
23:          end for
24:       end if
25:    end for
26: end while
```

Algorithm 5. AC-6-based Filtering Algorithm for the CSR Constraint

Lemma 5. *Achieving GAC on the collection of binary constraints represented by the CSR constraint in time $O(nd^2)$ and space $O(nd)$ where n is the number of variables and d is the size of the value universe D.*

8 Experiments

The purpose of our experiments is to demonstrate the hypothesis that our CSR propagator brings substantial practical benefits, and that therefore this area of research has both theoretical as well as practical merit. To this end, we study two problems that can be modeled by the CSR. We show that filtering can be sped up significantly by exploiting the knowledge that all pairs of variables are constrained in the same way. Note that the traditional AC6 and the improved version for CSR achieve the exact same consistency, thus causing identical search trees to be explored. We therefore compare the time needed per choice point, without comparing how the overall model compares with the state-of-the-art for each problem as this is beyond the scope of this paper.

We performed a number of comparisons between our AC-4 and AC-6 based algorithms, and found that the AC-6 algorithm always outperformed the AC-4 algorithm.

Table 1. Average Speed-up for the Stable Marriage Problem

		Couples								
		10	15	20	25	30	35	40	45	50
Probability	0.6	2.3	4.8	6.3	9.1	11.0	11.4	12.4	12.2	12.3
of attraction	0.9	3.8	4.9	6.5	6.9	8.6	9.2	10.1	11.4	12.9

This is not surprising, based both on our theoretical analysis, and previous work showing AC-6 outperforms AC-4 [1]. Because of this and space limitations we only present experiments using our AC-6 based algorithm.

All experiments are implemented using the Minion constraint solver on a Macbook with 4GB RAM and a 2.4GHz processor. These experiments show that CSR is a very robust and efficient propagator, never resulting in a slow-down in any of our experiments and producing over a hundred times speedup for larger instances.

Stable Marriage. The first problem we consider is the Stable Marriage Problem. It consists in pairing n men and n women into couples, so that no husband and wife in different couples prefer each other to their partner. We refer the reader to [11] for a complete definition and discussion of previous work on this problem. We use the hardest variant of this problem, where the men and women are allowed ties in their preference lists, and to give a list of partners they refuse to be married to under any circumstances. These two extensions make the problem NP-hard.

The standard model used in the CP literature keeps a decision variable for each man and woman where each domain consists of the indices of the corresponding preference list. The model then posts a constraint for each man-woman pair consisting of a set of no good pairs of values. We use the following, alternative model. Our model has one decision variable for each couple, whose domain is all possible pairings. We post between every pair of variables that the couples are 'stable', and also do not include any person more than once.

This model is inferior to using a specialized n-ary constraint for the stable marriage problem [21], the intention is not to provide a new efficient model for the stable marriage problem. The reason we consider this model here is to show the benefits of using a CSR constraint in place of a clique of binary constraints.

The instances we consider are generated at random, with a fixed probability that any person will refuse to be married to another. This allows us to vary the number of tuples in the constraints. As the search space is the same size regardless of if we use our specialized CSR propagator or a clique of constraints (they achieve the same level of consistency after all), we show only the speed-up that our algorithm provides per search node. So, 2.0 means that our algorithm solved the problem twice as fast, or searched twice as many search nodes per second. The CSR propagator was never slower in any experiment we ran. For each instance we generated and solved 20 problems and took the average speed-up per choice point.

Table 1 shows the results. We note that CSR always provides a sizable improvement in performance, that only increases as problem instances get larger and harder, increasing up to over 10 times faster for larger instances.

Table 2. Average Speed-up for the Table Planning Problem

		People Per Table							
		30	40	50	60	70	80	90	100
Probability of edge	0.4	15	37	58	76	90	105	117	136
	0.5	11	33	51	66	80	81	83	91
	0.6	13	31	49	63	77	76	77	78
	0.8	7	18	21	27	33	34	36	38
	0.9	5	6	8	14	15	19	20	22

The reason that the gain begins to reach a limit is that the size of the domains of the variable increases as the square of the number of people, meaning the cliques of size 50 have variables of domain 2500. Book-keeping work in the solver and algorithm for such large domains begins to dominate. Nevertheless, our algorithm is still over 10 times faster for these large problems.

Table Planning. The second problem we consider is the *Table Planning Problem*. The Table Planning Problem is the problem of sitting a group of people at tables, so that constraints about who will sit with each other are satisfied. Problems like this one often occur in the planning of events.

In this paper, an instance of the Table Planning Problem (TPP) is a triple $\langle T, S, R\rangle$ where T is the number of tables and S is the size of each table. This implies there are $S \times T$ people to sit. R is a symmetric relation on the set $\{1, \ldots, S \times T\}$, which i is related to j if people i and j are willing to sit on the same table. A solution to the TPP therefore is a partition of people, denoted by the set $\{1, \ldots, S \times T\}$, where each part of the partition represents a table. Therefore in any solution each member of this partition must be of size S and all pairs of numbers within it must satisfy R.

We consider instances of TPP with three tables and where R is generated randomly, with some fixed probability of each edge, and its symmetric image, being added. The model we use is an $S \times T$ matrix, with each variable having domain $\{1, \ldots, S \times T\}$. The constraints are that each row (representing a table) has a clique of the constraint R and a single AllDifferent constraint on all the variables. We consider representing the cliques of the constraint R either as separate constraints, or using our propagator.

As we know that the size of search will be identical, regardless of how the cliques of R are implemented, we show only the speed-up achieved by our improved propagator. We run each of the following experiments for ten different randomly generated relations, and take an average of the speed-ups achieved. We measure speed-up based on the number of nodes per second searched in ten minutes, or how long it takes to find the first solution or prove no solution exists, whichever is fastest.

Our results are presented in Table 2. We observe large, scalable gains for larger problems, with over 20 times speed-up for the densest problems. For sparser constraints, we even achieve over 100 times speed-up using our CSR propagator instead of a clique of constraints. This shows again how well the CSR propagator scales for larger problems, achieving immense practical improvements.

9 Conclusions and Future Work

We have looked at generalizing the famous AllDifferent to cliques of other constraints, and also other standard patterns such as bipartite, directed acyclic, and grid graphs. Unlike with the AllDifferent case, these constraints pose intractable filtering problems. By making use of the structure however, we can still provide substantial improvements in both theoretical and practical performance using new, generic algorithms. We have performed benchmarking across two problems using an AC-6 based decomposition algorithm on the CSR constraint. The experimental results show substantial gains in performance, proving that is worth exploiting the structure of same-relation constraints.

In the future, now we have laid down a theoretical framework, we will consider further benchmarks. In particular, we are interested to study how same-relation constraints interact with other global constraints.

References

1. Bessière, C., Cordier, M.O.: Arc-consistency and arc-consistency again. In: AAAI, pp. 108–113 (1993)
2. Bessière, C., Freuder, E., Régin, J.C.: Using Constraint Metaknowledge to Reduce Arc Consistency Computation. Artificial Intelligence 107(1), 125–148 (1999)
3. Bessière, C., Hebrard, E., Hnich, B., Walsh, T.: The Complexity of Reasoning with Global Constraints. Constraints 12(2), 239–259 (2007)
4. Bulatov, A.: Tractable Conservative Constraint Satisfaction Problems. LICS, 321–330 (2003)
5. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
6. Dechter, R., Pearl, J.: Tree Clustering for Constraint Networks. Artificial Intelligence 38(3), 353–366 (1989)
7. Downey, R., Fellows, M.: Parametrized Complexity. Springer, Heidelberg (1999)
8. Feder, T., Vardi, M.: The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction. SIAM Journal on Computing 28(1), 57–104 (1998)
9. Freuder, E.: A Sufficient Condition for Backtrack-bounded Search. Journal of the ACM 32(4), 755–761 (1985)
10. Freuder, E.: Complexity of K-Tree Structured Constraint Satisfaction Problems. In: AAAI, pp. 4–9 (1990)
11. Gent, I.P., Irving, R.W., Manlove, D.F., Prosser, P., Smith, B.M.: A Constraint Programming Approach to the Stable Marriage Problem. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 225–239. Springer, Heidelberg (2001)
12. Gottlob, G., Scarcello, F.: A comparison of structural CSP decomposition methods. Artificial Intelligence 124(2), 243–282 (1999)
13. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54(1) (2007)
14. Hell, P., Huang, J., Rafiey, A.: List Homomorphism to Reflexive Digraphs: Dichotomy Classification (submitted, 2009)
15. Klostermeyer, W., MacGillivray, G.: Homomorphisms and oriented colorings of equivalence classes of oriented graphs. Discrete Mathematics 274(1-3), 161–172 (2004)
16. Mohr, R., Henderson, T.: Arc and path consistency revisited. Artificial Intelligence 28(2), 225–233 (1986)

17. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. AAAI 1, 362–367 (1994)
18. Salamon, A.Z., Jeavons, P.G.: Perfect Constraints Are Tractable. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 524–528. Springer, Heidelberg (2008)
19. Sopena, E.: Oriented graph coloring. Discrete Mathematics 229(1-3), 359–369 (2001)
20. Swarts, J.S.: The Complexity of Digraph Homomorphisms: Local Tournaments, Injective Homomorphisms and Polymorphisms. PhD thesis, University of Victoria (2008)
21. Unsworth, C., Prosser, P.: An n-ary constraint for the stable marriage problem. In: IJCAI, pp. 32–38 (2005)