The `alldifferent` Constraint: A Survey

W.J. van Hoeve

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
`wjvh@cwi.nl`
`http://www.cwi.nl/~wjvh`

Abstract. The constraint of difference is known to the constraint programming community since Lauriere introduced Alice [11] in 1978. Since then, several solving strategies have been designed for this constraint. In this paper we give both a practical overview and an abstract comparison of these different strategies.

1 Introduction

Many problems from combinatorial optimization can be modeled and solved using techniques from Constraint Programming [14, 22]. One of the constraints that arises naturally in these models is the `alldifferent` constraint, which states that all variables in this constraint must be pairwise different. In Example 1, a scheduling problem is modeled using the `alldifferent` constraint.

Example 1 (Scheduling of speeches). Consider the following simple scheduling problem, adapted from Puget [17], where a set of speeches must be scheduled during one day. Each speech lasts exactly one hour (including questions and a coffee break), and only one conference room is available. Furthermore, each speaker has other commitments, and is available only for a limited fraction of the day. A particular instance of this problem is given in Table 1, where the fractions are defined by an earliest and latest possible time slot. This problem

Table 1. Time slots for the speakers

Speaker	Earliest	Latest
Sebastian	3	6
Frédéric	3	4
Jan-Georg	2	5
Krzysztof	2	4
Maarten	3	4
Luca	1	6

can be modeled as follows. We create one variable per speaker, whose value will be the period of his speech. The initial domains of the variables will be the available time intervals as stated in Table 1. Since two speeches cannot be held at the same time in the same conference room, the period for two different speakers must be different. The constraints for this scheduling problem thus become:

$$\begin{gathered} x_1 \in [3,6], x_2 \in [3,4], x_3 \in [2,5], \\ x_4 \in [2,4], x_5 \in [3,4], x_6 \in [1,6], \\ \texttt{alldifferent}(x_1, x_2, x_3, x_4, x_5, x_6). \end{gathered}$$

To find a solution to a model as in the previous example, a constraint solver essentially builds a search tree from all possible variable values. In general, finding a solution for such problems is $\mathcal{NP}$-complete, and this search tree can grow extremely large. Therefore, strategies have been developed to prune parts of the search tree. In Constraint Programming, these strategies mainly consist of the simplification of the problem during the search for a solution. The techniques that are most widely applied are so-called consistency techniques that can reduce the domains of the variables, based on the constraints between them. Therefore, algorithms that achieve some state of consistency are also called filtering algorithms.

This paper deals with consistency techniques or filtering algorithms that can be deduced from the `alldifferent` constraint. It turns out that there exist different degrees of consistency, each degree allowing more or less values in the variable domain. In general it takes more time to obtain a stronger consistency than to obtain a weaker consistency. So with more effort, one could remove more values. Therefore, for each individual problem one has to make a trade-off between the effort (time) and the gain (domain shrinking) when choosing a particular consistency to achieve.

1.1 Overview

The different degrees of consistency will be defined in Section 2, together with some more preliminaries. Then each of the Sections 3 up to 6 will treat one consistency technique. These sections are ordered in increasing strongness of the considered consistency. The treatment consists of a description of the particular consistency with respect to the `alldifferent` constraint, together with an algorithm that achieves this consistency. Finally, a conclusion is given in Section 7.

2 Preliminaries

A constraint satisfaction problem (**CSP**) is defined as a finite set of variables $\mathcal{X} = \{x_1, \dots, x_n\}$, with domains $\mathcal{D} = \{D_1, \dots, D_n\}$ associated with them, together with a finite set of constraints $\mathcal{C}$, each on a subset of $\mathcal{X}$. A CSP P will also be denoted as $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. A **constraint** $C \in \mathcal{C}$ is defined as a subset of the Cartesian product of the domains of the variables that are in C. For instance, $C(x_1, x_3, x_4) \subseteq D_1 \times D_3 \times D_4$. An n-uple $(d_1, \dots, d_n) \in D_1 \times \cdots \times D_n$ is a **solution** to a CSP if for every constraint $C \in \mathcal{C}$ on the variables $x_{i_1}, \dots, x_{i_m}$ we have $(d_{i_1}, \dots, d_{i_m}) \in C$. For finite, linearly ordered domains D_i, we define

$\min D_i$ and $\max D_i$ to be the minimum value and the maximum value of the domain D_i.

We now introduce four notions of local consistency in the order they will be discussed in the text. Note the use of braces ({, }) and brackets ([,]) that indicate a set and an interval of domain values respectively.

Definition 1 (Arc consistency). *A binary constraint $C(x_1, x_2)$ where D_1 and D_2 are non-empty, is called arc consistent iff $\forall d_1 \in D_1$ $\exists d_2 \in D_2$ such that $(d_1, d_2) \in C$, and $\forall d_2 \in D_2$ $\exists d_1 \in D_1$ such that $(d_1, d_2) \in C$.*

Definition 2 (Bound consistency). *An m-ary constraint $C(x_1, \dots, x_m)$ where no domain D_i is empty, is called bound consistent iff for each variable x_i: $\forall d_i \in \{\min D_i, \max D_i\}, \forall j \in \{1, \dots, m\} - \{i\}, \exists d_j \in [\min D_j, \max D_j]$ such that $(d_1, \dots, d_m) \in C$.*

Definition 3 (Range consistency). *An m-ary constraint $C(x_1, \dots, x_m)$ where no domain D_i is empty, is called range consistent iff for each variable x_i: $\forall d_i \in D_i, \forall j \in \{1, \dots, m\} - \{i\}, \exists d_j \in [\min D_j, \max D_j]$ such that $(d_1, \dots, d_m) \in C$.*

Definition 4 (Hyper-arc consistency). *An m-ary constraint $C(x_1, \dots, x_m)$ where no domain D_i is empty, is called hyper-arc consistent iff for each variable x_i: $\forall d_i \in D_i, \forall j \in \{1, \dots, m\} - \{i\}, \exists d_j \in D_j$ such that $(d_1, \dots, d_m) \in C$.*

In other words, both arc consistency and hyper-arc consistency check whether any value in every domain does belong to a feasible instance of the constraint, based on the domains. Range consistency however, does not check the feasibility of the constraint with respect to the domains, but with respect to intervals that include the domains. It can be regarded as a relaxation of hyper-arc consistency. Bound consistency can be regarded as a relaxation of range consistency. It does not even check all values in the domains, but only the minimum and the maximum value, while still verifying the constraint with respect to intervals that include the domains. This is formalized in Proposition 1.

Definition 5 (Consistent CSP). *A CSP is arc consistent if all its binary constraints are. A CSP is range consistent, respectively, bound consistent or hyper-arc consistent if all its constraints are.*

Consider a CSP P. If we apply to P an algorithm that achieves range consistency on P, we will denote the result as $\Phi_R(P)$. Analogously, $\Phi_B(P)$, $\Phi_A(P)$ and $\Phi_{HA}(P)$ denote the achievement of bound consistency, arc consistency and hyper-arc consistency on P respectively. Let $P_\emptyset$ denote a failed CSP, i.e. a CSP with at least one empty domain. We define a CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ smaller than a CSP $P' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ if $\mathcal{D} \subseteq \mathcal{D}'$. This relation is written as $P \preceq P'$. A CSP P is strictly smaller than a CSP P', i.e. $P \prec P'$, when $\mathcal{D} \subseteq \mathcal{D}'$ and $D_i \subset D'_i$ for at least one i. When both $P \preceq P'$ and $P' \preceq P$ we write $P \equiv P'$. By convention, $P_\emptyset$ is the smallest CSP. This notation is adopted from [3].

Proposition 1. *$\Phi_{HA}(P) \preceq \Phi_R(P) \preceq \Phi_B(P)$.*

Proof. Both hyper-arc consistency and range consistency verify all values of all domains. But hyper-arc consistency verifies the constraints with respect to the exact domains D_i, while range consistency verifies the constraints with respect to intervals that include the domains: $[\min D_i, \max D_i]$. A constraint that holds on a domain D_i also holds on the interval $[\min D_i, \max D_i]$ since $D_i \subseteq [\min D_i, \max D_i]$. The converse is not true, see Example 2. Hence $\Phi_R(P) \preceq \Phi_{HA}(P)$.

Both range consistency and bound consistency verify the constraints with respect to intervals that include the domains. But bound consistency only considers $\min D_i$ and $\max D_i$ for a domain D_i, while range consistency considers all values in D_i. Since $\{\min D_i, \max D_i\} \subseteq D_i$, $\Phi_B(P) \preceq \Phi_R(P)$. Example 2 shows that $\Phi_B(P) \prec \Phi_R(P)$ cannot be discarded.

The following examples clarify Proposition 1.

Example 2 (Comparing consistencies). Consider the following CSP:

$$P = \begin{cases} x_1 \in \{1,3\}, x_2 \in \{2\}, x_3 \in \{1,2,3\}, \\ \texttt{alldifferent}(x_1, x_2, x_3). \end{cases}$$

Then $\Phi_B(P) \equiv P$, while

$$\Phi_R(P) = \begin{cases} x_1 \in \{1,3\}, x_2 \in \{2\}, x_3 \in \{1,3\}, \\ \texttt{alldifferent}(x_1, x_2, x_3). \end{cases}$$

and $\Phi_{HA}(P) \equiv \Phi_R(P)$. Next, consider the CSP

$$P' = \begin{cases} x_1 \in \{1,3\}, x_2 \in \{1,3\}, x_3 \in \{1,3\}, \\ \texttt{alldifferent}(x_1, x_2, x_3). \end{cases}$$

This CSP is obviously inconsistent, since there are only two values available, namely 1 and 3, for three variables that must be pairwise different. $\Phi_{HA}(P')$ will detect this inconsistency, while $\Phi_R(P') \equiv P'$.

A useful theorem to derive algorithms that ensure consistency for the `all-different` constraint is Hall's Theorem [9]. The following formulation is stated in terms of the `alldifferent` constraint. The cardinality of a set K is denoted by $|K|$.

Theorem 1 (Hall). *The constraint* $\texttt{alldifferent}(x_1, \dots, x_n)$ *on the variables* $x_1, \dots, x_n$ *with respective domains* $D_1, \dots, D_n$ *has a solution if and only if no subset* $K \subseteq \{x_1, \dots, x_n\}$ *exists such that* $|K| > |\cup_{x_i \in K} D_i|$.

As an application of Theorem 1, let us return to the CSP P' in Example 2. Take as subset $K = \{x_1, x_2, x_3\}$, then $|K| = 3$. Furthermore, $|\cup_{x_i \in K} D_i| = |\{1,3\}| = 2$. For this subset K, Hall's condition does not hold ($3 > 2$), hence this CSP has no solution.

3 Local Consistency of a Decomposed CSP

The standard filtering algorithm for the `alldifferent` constraint is as follows. Whenever the domain of a variable contains only one value, remove this value from the domains of the other variables that occur in the `alldifferent` constraint. This procedure is repeated as long as possible. Although this algorithm might seem rather poor or naive, it has been successfully implemented in many constraint solvers, for instance in the system CHIP [22].

This filtering algorithm can also be described as follows. A common way to rewrite the `alldifferent` constraint is to generate a sequence of disequalities. For instance

$$\texttt{alldifferent}(x_1, x_2, x_3, x_4) \rightarrow \begin{cases} x_1 \neq x_2,\, x_1 \neq x_3,\, x_1 \neq x_4, \\ x_2 \neq x_3,\, x_2 \neq x_4,\, x_3 \neq x_4. \end{cases}$$

If we apply an algorithm that achieves arc consistency on this set of binary constraints, we obtain the same filtering as described above. One of the drawbacks of this method is the quadratic increase of the number of constraints. One needs $\binom{n}{2} = \frac{1}{2}(n^2 - n)$ disequalities to express an n-ary `alldifferent` constraint. But an even more important drawback is the loss of information. When this set of binary constraints is being made arc consistent, only two variables are compared at a time. However, when the `alldifferent` constraint is being made hyper-arc consistent, all variables are considered at the same time, which gives a much stronger consistency. This is shown in Proposition 2. Let P_{dec} denote the decomposed CSP P in which all `alldifferent` constraints have been replaced by a sequence of disequalities.

Proposition 2. $\Phi_{HA}(P) \preceq \Phi_A(P_{dec})$.

Proof. Since the definition of arc consistency and hyper-arc consistency is equivalent for binary constraints, we only need to consider the filtering of the `alldifferent` constraint. Consider the constraint C: `alldifferent`$(x_1, \dots, x_n)$ and the corresponding decomposition in terms of disequalities, denoted by C_{dec}. If a value $d_i \in D_i$ is not arc consistent w.r.t. the set C_{dec}, then it is also not hyper-arc consistent w.r.t. C. Indeed, when d_i is not arc consistent w.r.t. C_{dec}, then we cannot find a $d_j \in D_j$ for some variable x_j such that $x_i \neq x_j$. But then we also cannot find an n-uple $(d_1, \dots, x_n) \in C$, since we cannot find a value $d_j \in D_j$ such that $d_i \neq d_j$. Therefore, $\Phi_{HA}(P) \preceq \Phi_A(P_{dec})$. The converse is not true, as illustrated in Example 3.

Example 3 (Hyper-arc and arc consistency compared). For some integer $n \geq 3$, consider the CSP's

$$P = \begin{cases} x_1 \in \{1, \dots, n-1\}, \dots, x_{n-1} \in \{1, \dots, n-1\}, x_n \in \{1, \dots, n\}, \\ \texttt{alldifferent}(x_1, \dots, x_n) \end{cases}$$

$$P_{dec} = \begin{cases} x_1 \in \{1, \dots, n-1\}, \dots, x_{n-1} \in \{1, \dots, n-1\}, x_n \in \{1, \dots, n\}, \\ x_1 \neq x_2, \dots, x_{n-1} \neq x_n. \end{cases}$$

Now $\Phi_A(P_{dec}) \equiv P_{dec}$, while

$$\Phi_{HA}(P) = \begin{cases} x_1 \in \{1,\ldots,n-1\},\ldots,x_{n-1} \in \{1,\ldots,n-1\}, x_n \in \{n\}, \\ \texttt{alldifferent}(x_1,\ldots,x_n). \end{cases}$$

Our next goal is to find a consistency notion for the set of disequalities that is equivalent to the hyper-arc consistency notion for the `alldifferent` constraint. Relational consistency can be used for this.

Definition 6 (Relational $(1, m)$ consistency, [6]). *A set of constraints $S = \{C_1,\ldots,C_m\}$ is* relationally $(1,m)$-consistent *iff all domain values $d \in D_i$ of variables appearing in S, appear in a solution to the m constraints, evaluated simultaneously. A CSP $P = (\mathcal{X},\mathcal{D},\mathcal{C})$ is relationally $(1,m)$-consistent iff every set of m constraints $S \subseteq \mathcal{C}$ is relationally $(1,m)$-consistent.*

Note that arc consistency is equivalent to $(1,1)$-consistency.

Again, let P be the CSP that consists only of the `alldifferent` constraint and a corresponding set of variables and domains.

Proposition 3. $\Phi_{HA}(P) \equiv \Phi_{R(1,\frac{1}{2}(n^2-n))C}(P_{dec})$.

Proof. By construction we have that the `alldifferent` constraint is equivalent to the simultaneous consideration of the sequence of corresponding disequalities. The number of disequalities is precisely $\frac{1}{2}(n^2-n)$. If we consider only $\frac{1}{2}(n^2-n)-i$ disequalities simultaneously ($1 \leq i \leq \frac{1}{2}(n^2-n)-1$), there are i unconstrained relations between variables, and the corresponding variables could take the same value when a certain instantiation is considered. Therefore, we really need to take all $\frac{1}{2}(n^2-n)$ constraints into consideration, which corresponds to the relational $(1,\frac{1}{2}(n^2-n))$-consistency.

As suggested before, the pruning performance of $\Phi_A(P_{dec})$ is rather poor. Moreover, the complexity is relatively high, namely around $O(n^2)$, whereas the hyper-arc consistency algorithms are around $O(dn^{1.5})$, where d is the maximum cardinality of the domains and n is the number of variables involved [12, 18]. Nevertheless, this filtering algorithm applies quite well to several problems, such as the n-queens problem ($n < 200$) [12, 17].

Other work on the comparison of the `alldifferent` constraints and the corresponding decomposition has for instance been done in [20] and [8].

4 Bound Consistency

The notion of bound consistency for the `alldifferent` constraint was introduced by Puget [17]. We summarize his method in this section. Puget uses Hall's Theorem to construct an algorithm that achieves bound consistency.

Definition 7 (Hall interval). *Given an interval I, let K_I be the set of variables x_i such that $D_i \subseteq I$. We say that I is a Hall interval iff $|I| = |K_I|$.*

Proposition 4 (Puget [17]). *The constraint* `alldifferent`$(x_1, \dots, x_n)$ *where no domain* D_i *is empty, is bound consistent iff*

- *for each interval* I*:* $|K_I| \le |I|$,
- *for each Hall interval* I*:* $\{\min D_i, \max D_i\} \cap I = \emptyset$ *for all* $x_i \notin K_I$*.*

Proposition 4 can be used to construct an algorithm that achieves bound consistency on the `alldifferent` constraint. Indeed, we could check every interval I with bounds ranging from the minimum of all domains to the maximum of all domains. When $|I| \le |K_I|$, we know that the constraint is inconsistent. And for each Hall interval, we remove all $\min D_i$ and $\max D_i$ until $\{\min D_i, \max D_i\} \cap I = \emptyset$. Puget gives an implementation with the time complexity of $O(n \log n)$.

In [15], Mehlhorn and Thiel present an algorithm that achieves bound consistency of the `alldifferent` constraint in time $O(n)$ plus the time required for sorting the interval endpoints. In particular, if the endpoints are from a range of size $O(n^k)$ for some constant k, the algorithm runs in linear time.

Example 4. The following simple problem shows an application of the bound consistency algorithm based on intervals.

$$P = \begin{cases} x_1 \in \{1,2\}, x_2 \in \{1,2\}, x_3 \in \{2,3\}, \\ \texttt{alldifferent}(x_1, x_2, x_3). \end{cases}$$

Intuitively, observe that the variables x_1 and x_2 both have domain $\{1, 2\}$. So these two variables together range over two values, and for a feasible instantiation they must be different. This means that the values 1 and 2 must be assigned to these two variables. Hence, values 1 and 2 cannot be assigned to any other variable and therefore, value 2 will be removed from the domain of x_3.

The algorithm detects this when the interval I is set to $I = \{1, 2\}$. Then the number of variables for which $D_i \subseteq I$ is 2, namely x_1 and x_2. Since $|I| = 2$, I is a Hall interval. The domain of x_3 is not in this interval, and $\{\min D_3, \max D_3\} \cap I = \{\min D_3\}$. In order to obtain the empty set in the right hand side of the last equation, we need to remove $\min D_i$. The resulting CSP is bound consistent.

5 Range Consistency

An algorithm that achieves range consistency was introduced by Leconte [12]. We follow the same procedure as in the previous example. Leconte also uses Hall's Theorem to construct the algorithm.

Definition 8 (Hall set). *Given a set of variables* K*, let* I_K *be the interval* $[\min D_K, \max D_K]$*, where* $D_K = \cup_{x_i \in K} D_i$*. We say that* K *is a Hall set iff* $|K| = |I_K|$.

Note that in the above definition I_K does not necessarily need to be a Hall interval.

Proposition 5 (Leconte [12]). *The constraint* `alldifferent`$(x_1, \ldots, x_n)$ *where no domain* D_i *is empty, is range consistent iff for each Hall set* $K \subseteq \{x_1, \ldots x_n\}$*:* $D_i \cap I_K = \emptyset$ *for all* $x_i \notin K$.

We can deduce an algorithm from Proposition 5 in a similar way as we did for the algorithm for bound consistency. Leconte implemented an algorithm that achieves range consistency with a complexity of $O(n^2 d)$, where d is the average size of the domains.

Observe that this algorithm is similar to the algorithm for bound consistency. Where the algorithm for bound consistency takes the domains as a starting point, the algorithm for range consistency takes the variables. But they both attempt to reach a situation in which the cardinality of a set of variables is equal to the cardinality of the union of the corresponding domains, as was illustrated in Example 4.

6 Hyper-arc Consistency

A filtering algorithm that achieves hyper-arc consistency for constraints of difference was proposed by Régin [18]. A similar result was obtained independently by Costa [5]. Before we can introduce this algorithm, we have to establish a connection with the maximum matching problem in graph theory. The standard reference to matching theory is the book by Lovász and Plummer [13].

6.1 Connections with Matching Theory

Consider again the scheduling problem from Example 1. To illustrate the problem, assume that Krzysztof and Luca decided not to speak. We now want to model this problem graph-theoretically. First we introduce the definition of a bipartite graph.

Definition 9 (Bipartite graph). *A* graph G *consists of a finite non-empty set of elements* V *called* nodes *and a set of pairs of nodes* E *called* edges. *If the node set* V *can be partitioned into two disjoint non-empty sets* X *and* Y *such that all edges in* E *join a node from* X *to a node in* Y*, we call* G bipartite *with* bipartition (X, Y). *We also write* $G = (X, Y, E)$.

The remaining speakers from Example 1 and their available times can be represented by the bipartite graph in Figure 1. Both speakers and time periods are represented by nodes, and these two sets of nodes are connected by edges, giving the bipartition (*Speakers*, *Times*). We call the constructed bipartite graph of an `alldifferent` constraint C the *value graph* of C. Let X_C denote the variables occurring in a constraint C, with corresponding domains D_C.

Definition 10 (Value graph). *Given an* `alldifferent` *constraint* C*, the bipartite graph* $GV(C) = (X_C, D_C, E)$ *where* $(x_i, d) \in E$ *iff* $d \in D_i$ *is called the value graph of* C.

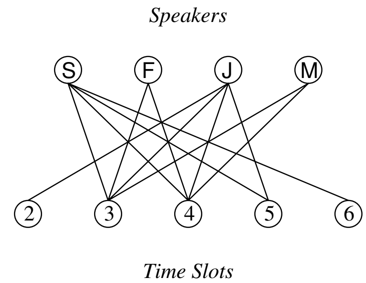

Fig. 1. The value graph for the revised speech scheduling problem

Definition 11 (Maximum matching). *A subset of edges in a graph G is called a* matching *if no two edges have a node in common. A matching of maximum cardinality is called a* maximum matching*. A matching M* covers a set X *if every node in X is an endpoint of an edge in M.*

Note that a matching that covers the set of speakers in Figure 1 is a maximum matching. The following theorem gives the link between a maximum matching in a bipartite graph and hyper-arc consistency of the `alldifferent` constraint.

Proposition 6 (Régin [18]). *The constraint C :* `alldifferent`$(x_1, \ldots, x_n)$ *is hyper-arc consistent iff every edge in its value graph $GV(C)$ belongs to a matching which covers X_C in $GV(C)$.*

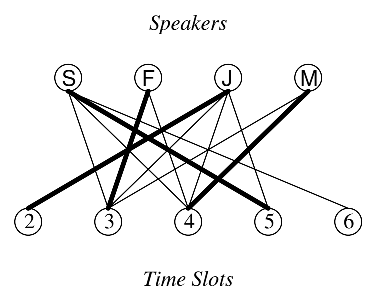

Fig. 2. A maximum matching in the value graph

An illustration of Proposition 6 is given in Figures 2 and 3. The fat lines in the graph of Figure 2 denote a maximum matching that covers all speaker nodes. Not all edges belong to such a matching, and by Proposition 6 they can be removed. When these edges are removed, the resulting `alldifferent` constraint is hyper-arc consistent. This is depicted in Figure 3, which corresponds to Table 2.

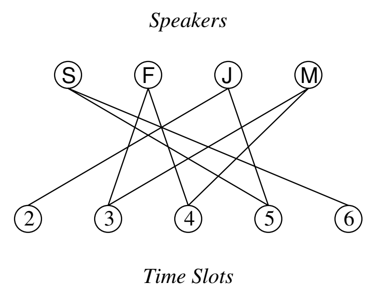

Fig. 3. The value graph after filtering

Table 2. Filtered time slots for the speakers

Speaker	Available
Sebastian	$\{5, 6\}$
Frédéric	$\{3, 4\}$
Jan-Georg	$\{2, 5\}$
Maarten	$\{3, 4\}$

6.2 An Algorithm for Achieving Hyper-arc Consistency

An algorithm that achieves hyper-arc consistency for the `alldifferent` constraint should remove all those edges in the corresponding value graph that do not belong to a maximum matching. Berge has given a property that identifies exactly these edges [2]. But first, we introduce some definitions we need for this property.

Definition 12. *Let M be a matching in a graph $G = (V, E)$. An* alternating path *or* alternating cycle *is a path or a cycle whose edges are alternately in M and in $E - M$. The* length *of a path or a cycle is the number of edges it contains. A node is called* free *w.r.t. M if it is not incident to a matching edge.*

For instance, in Figure 2, $(3, F, 4, M, 3)$ is an even alternating cycle of length 4. Node 6 is a free node.

Proposition 7 (Berge). *An edge belongs to a maximum matching iff for some maximum matching, it belongs to either an even alternating path which begins at a free node, or to an even alternating cycle.*

With this property, we are able to identify and remove edges that are not in any maximum matching. Note that we need to construct a maximum matching before we can apply this property. The algorithm that achieves hyper-arc consistency is represented in Figure 4. To construct the value graph GV, we need $O(d|X_C| + |X_C| + |D_C|)$ steps, where d is the maximum cardinality of a variable domain. The procedure ComputeMaximumMatching(GV) computes a maximum matching in the graph GV. This can be done for instance with a so-called

Input: constraint of difference C, variables $\mathcal{X}$ and domains $\mathcal{D}$
Output: false when no solution, otherwise true and updated domains
begin
1 Build $GV = (X_C, D_C, E)$
2 $M(GV) \leftarrow$ ComputeMaximumMatching(GV)
3 if $|M(GV)| < |X_C|$ then return false
4 RemoveEdgesFromG($GV, M(GV)$)
5 return true
end

Fig. 4. An algorithm for achieving hyper-arc consistency

augmenting path algorithm. Hopcroft and Karp gave an implementation for this that runs in $O(\sqrt{|X_C|}m)$ time, where m is the number of edges of GV [10]. Their algorithm still remains essentially the best known [4].

From Hall's Theorem we already know that whenever we find a subset of nodes the cardinality of which exceeds the cardinality of the corresponding set of domain values, no matching exists that saturates X_C. This is checked in line 3. In the procedure RemoveEdgesFromG($GV, M(GV)$) the actual filtering takes place. Instead of applying Berge's property directly, we can translate the problem in such a way, that we have to search for the so-called strongly connected components of the graph [18]. For this problem we can use an implementation by Tarjan that runs in $O(n + m)$ time on graphs with n nodes and m edges [18, 21]. In the algorithm from Figure 4, the search for a maximum matching remains the dominant factor, hence the total algorithm runs in $O(\sqrt{|X_C|}m)$ time.

The notion of hyper-arc consistency was introduced by Mohr and Masini [16]. They also give a general algorithm to achieve this notion. For an n-ary `alldifferent` constraint, where the domain size of all variables is bounded by d, $D_i \leq d$, the time complexity of the general algorithm is $O(\frac{d!}{(d-n)!})$, whereas the time complexity of the above algorithm is $O(dn\sqrt{n})$.

7 Conclusions and Future Work

In this paper, an overview of several filtering techniques for the `alldifferent` constraint has been given. A comparison of these different techniques has been made by means of corresponding notions of local consistency and algorithms to achieve them.

However, there are other interesting articles related to this subject, that are not considered in this paper. For instance, Focacci et al. [7] use information from the `alldifferent` constraint for a filtering technique based on reduced costs. Furthermore, in [19] Régin introduced the symmetric `alldifferent` constraint, together with filtering algorithms for this constraint. Finally, Barták considers a dynamic version of the `alldifferent` constraint [1].

References

1. R. Barták. Dynamic global constraints: A first view. In *CP-AI-OR 2001, Third International Workshop on Integration of AI and OR Techniques*, 2001.
2. C. Berge. *Graphs and hypergraphs*. North-Holland, 1973.
3. H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, 5:1–16, 1999.
4. W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, 1998.
5. M.-C. Costa. Persistency in maximum cardinality bipartite matchings. *Operations Research Letters*, 15(3):143–149, 1994.
6. R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173:283–308, 1997.
7. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In *PPCP - CP'99*, volume 1713 of *LNCS*, pages 189–203, 1999.
8. I. Gent, K. Stergiou, and T. Walsh. Decomposable Constraints. In *New Trends in Constraints*, volume 1865 of *LNCS*, pages 134–149. Springer, 2000.
9. P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.
10. J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
11. J.-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence. An International Journal*, 10(1):29–127, 1978.
12. M. Leconte. A bounds-based reduction scheme for constraints of difference. In *Constraint-96, Second International Workshop on Constraint-based Reasoning, Key West, Florida*, 1996.
13. L. Lovász and M. D. Plummer. *Matching Theory*. North-Holland, Amsterdam, 1986.
14. K. Marriot and P. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, MA, 1998.
15. K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *PPCP - CP2000*, volume 1894 of *LNCS*, pages 306–319, 2000.
16. R. Mohr and G. Masini. Good Old Discrete Relaxation. In *European Conference on Artificial Intelligence (ECAI-88)*, pages 651–656, Munchen, Germany, 1988.
17. J.-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the AAAI '98*, pages 359–366, 1998.
18. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the AAAI '94*, pages 362–367, 1994.
19. J.-C. Régin. The symmetric alldiff constraint. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 420–425, 1999.
20. K. Stergiou and T. Walsh. The difference all-difference makes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 414–419, 1999.
21. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
22. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. MIT Press, Cambridge, MA, 1989.