The alldifferent Constraint: A Survey

W.J. van Hoeve (wjvh@cwi.nl)
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract. The constraint of difference is known to the constraint programming community since Lauriere introduced ALICE in 1978. Since then, several strategies have been designed to solve the alldifferent constraint. This paper surveys the most important developments over the years regarding the alldifferent constraint. First we summarize the underlying concepts and results from graph theory and integer programming. Then we give an overview and an abstract comparison of different solution strategies. In addition, the symmetric alldifferent constraint is treated. Finally, we show how to apply cost-based filtering to the alldifferent constraint.

A preliminary version of this paper appeared as [14].

Keywords: constraint programming, constraint satisfaction, alldifferent, filtering algorithms, consistency notions

Table of Contents

Note: Submitted manuscript.

1. Introduction

Many combinatorial optimization problems can be modeled and solved using techniques from constraint programming [22, 33]. 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 this section we first introduce the `alldifferent` constraint by an example, together with some information on how to solve such a constraint. Then a historical overview will be presented. This section concludes with an outline of the paper.

1.1. Introducing the `alldifferent` constraint

In Example 1, we model the famous n-queens problem using the `all-different` constraint.

EXAMPLE 1 (n-Queens problem). *This problem can be described as: "Place n queens on an $n \times n$ chessboard in such a way that no queen attacks another queen."*

One way of modelling this problem is to introduce an integer variable x_i for every row $i \in \{1, \dots, n\}$ that ranges over column 1 to n. This means that in every row i a queen is placed in the x_i-th column. Then we can express the no-attack constraints as

$$\begin{aligned} x_i &\neq x_j \\ x_i - x_j &\neq i - j \\ x_i - x_j &\neq j - i \\ x_i &\in \{1, \dots, n\} \end{aligned}$$

for all i and j such that $0 \leq i < j \leq n$. The first disequality states that no queens are allowed to occur in the same column. The second and the third disequality deal with the diagonal constraints. After rearranging the terms of the last two disequalities, we transform the model to

$$\begin{array}{l} \texttt{alldifferent}(x_1, \dots, x_n) \\ \texttt{alldifferent}(y_1, \dots, y_n) \\ \texttt{alldifferent}(z_1, \dots, z_n) \end{array}$$

where $y_i = x_i - i$ and $z_i = x_i + i$. Here $y_i \in \{-n+1, \dots, n-1\}$ and $z_i \in \{2, \dots, 2n\}$.

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 exponentially 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.

Several consistency techniques or filtering algorithms 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.

Example 1 shows two models for the n-queens problem, the first with disequality constraints, the other with `alldifferent` constraints. From a modelling point of view, one of the advantages of using `alldifferent` constraints is for instance the decrease of the number of constraints. Instead of using $\frac{3}{2}(n^2 - n)$ disequalities we only need 3 `alldifferent` constraints to model the n-queens problem. But maybe even more important is the information that can be extracted from the two models. The three `alldifferent` constraints provide us with so called 'global' information, whereas the disequalities only constrain two variables at a time. Therefore we can apply stronger filtering algorithms to the `alldifferent` constraints than we can apply to the set of disequality constraints.

1.2. Historical overview

In 1978, Lauriere introduced ALICE, "A language and a program for stating and solving combinatorial problems" [19]. Already in this system the importance of the `alldifferent` constraint was recognized. The keyword "`DIS`" applied to a set of variables is used to state that the variables must take different values. It defines a structure that can be exploited for filtering purposes.

After the introduction of constraints in logic programming, for instance in the system CHIP [33], it was also possible to express the constraint of difference as the well known `alldifferent` constraint. The system ECL^{i}PS^e [34] also introduced the constraint of pairwise difference as `alldistinct`. However, in the early constraint (logic) programming systems this constraint was being rewritten as a sequence of disequalities, see for instance [33]. The global information is lost in that way.

Throughout the history of constraint programming, the `alldifferent` constraint has played a special role. Various papers and books make use of this special constraint to show the benefits of constraint programming, either to show its modelling power, or to show that problems can be solved faster using this constraint. A typical example is the integration of operations research techniques and constraint programming, which has gained a lot of attention recently. A common example to introduce this field uses the `alldifferent` constraint, where methods of graph theory and integer programming are used to achieve some kind of consistency [17, 24].

Over the years, the `alldifferent` constraint as well as other global constraints were well-studied in constraint programming (see e.g. [2] for an overview). Special algorithms were being developed that are able to exploit the global information of the constraints. For the `alldifferent` constraint at least five different filtering algorithms exist, each achieving a different kind of consistency, or achieving it faster [28, 20, 26, 23]. Although the constraint is mentioned more or less deeply in a variety of papers and books (e.g. [27, 15, 22]), to our knowledge there is no work that collects all effort that has been put into this constraint. This paper therefore tries to give an overview of the `alldifferent` constraint, which will be outlined in the next section.

1.3. Outline of the paper

This paper is organized as follows. First we define formally the different degrees of consistency that can be applied to the `alldifferent` constraint in Section 2, together with some more preliminaries on constraint programming. Then we introduce results from graph theory and integer programming and make a connection with the `alldifferent` constraint, because several filtering algorithms make use of these results. Each of the Sections 4.1 up to 4.4 will treat one consistency technique. These sections are ordered in increasing strength 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. At the end of Section 4 a practical comparison of the consistencies will be made by applying them to the n-queens problem. In Section 5 a symmetric version of the `alldifferent` constraint is considered, the global constraint `symm_alldifferent`. A different way of filtering domain values is presented in Section 6, namely cost-based domain filtering deduced from the `alldifferent` constraint. It has a close connection with integer programming. Finally, the paper is concluded in Section 7.

2. Relevant Local Consistency Notions

A constraint satisfaction problem (CSP) is defined as a finite set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, with domains $\mathcal{D} = \{D_1, \ldots, D_n\}$ 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 $(d_1, \ldots, d_n) \in D_1 \times \ldots \times D_n$ is a solution to a CSP if for every constraint $C \in \mathcal{C}$ on the variables $x_{i_1}, \ldots, x_{i_m}$ we have $(d_{i_1}, \ldots, d_{i_m}) \in C$. 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 integer domain values respectively. Thus, the set $\{1, 3\}$ contains the vales 1 and 3 whereas the interval $[1, 3]$ contains 1, 2 and 3.

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 (Bounds consistency). *An m-ary constraint $C(x_1, \ldots, x_m)$ where no domain D_i is empty, is called bounds consistent iff for each variable x_i and value $d_i \in \{\min D_i, \max D_i\}$, there exist values $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_m$ in $[\min D_1, \max D_1], \ldots, [\min D_{i-1}, \max D_{i-1}]$, $[\min D_{i+1}, \max D_{i+1}], \ldots, [\min D_m, \max D_m]$, such that $(d_1, \ldots, d_m) \in C$.*

DEFINITION 3 (Range consistency). *An m-ary constraint $C(x_1, \ldots, x_m)$ where no domain D_i is empty, is called range consistent iff for each variable x_i and value $d_i \in D_i$, there exist values $d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_m$ in $[\min D_1, \max D_1], \ldots, [\min D_{i-1}, \max D_{i-1}], [\min D_{i+1}, \max D_{i+1}], \ldots,$ $[\min D_m, \max D_m]$, such that $(d_1, \ldots, d_m) \in C$.*

DEFINITION 4 (Hyper-arc consistency). *An m-ary constraint $C(x_1, \ldots, x_m)$ where no domain D_i is empty, is called hyper-arc consistent iff for each variable x_i and value $d_i \in D_i$, there exist values $d_1, \ldots, d_{i-1}$, $d_{i+1}, \ldots, d_m$ in $D_1, \ldots, D_{i-1}, D_{i+1}, \ldots, D_m$, such that $(d_1, \ldots, d_m) \in C$.*

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. Bounds 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, for which we will first introduce the following.

DEFINITION 5 (Consistent CSP). *A CSP is arc consistent if all its binary constraints are. A CSP is range consistent, respectively, bounds 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 bounds consistency, arc consistency and hyper-arc consistency on P respectively. We define a CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ to be 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. $P \prec P'$, if $\mathcal{D} \subseteq \mathcal{D}'$ and $D_i \subset D_i'$ for at least one i. When both $P \preceq P'$ and $P' \preceq P$ we write $P \equiv P'$. A failed CSP, i.e. a CSP with at least one empty domain, is denoted by $P_\emptyset$. By convention, $P_\emptyset$ is the smallest CSP. This notation is adopted from [5].

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 $\Phi_{HA}(P) \preceq \Phi_R(P)$.

Both range consistency and bounds consistency verify the constraints with respect to intervals that include the domains. But bounds 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 $\{\min D_i, \max D_i\} \subseteq D_i$, $\Phi_R(P) \preceq \Phi_B(P)$. Example 2 shows that $\Phi_R(P) \prec \Phi_B(P)$ 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* $\Phi_R(P') \equiv P'$*.*

3. Connections with Graph Theory and Integer Programming

The filtering algorithms that are applied to the `alldifferent` constraint are often based on results from graph theory or integer programming. This section first presents the `alldifferent` constraint as a maximum matching problem in graph theory. Then an integer programming representation of the `alldifferent` constraint is given, namely the assignment problem. Finally, Hall's theorem is presented as the basis of several filtering algorithms.

3.1. Matching theory

The `alldifferent` constraint has a natural correspondence to the maximum matching problem in graph theory. We will first give an illustrative example from which the `alldifferent` constraint can be easily expressed as a maximum matching problem. Good references to matching theory are [21, 12].

EXAMPLE 3 (Task assignment). *In this small example we want to assign four tasks to five machines. To each machine at most one task can be assigned. However, not every task can be assigned to every machine. Table I presents the possible combinations. For instance, task 2 can be assigned to machines B and C. This problem can be modeled as follows.*

Table I. Machines for tasks

Task	Machines
1	B, C, D, E
2	B, C
3	A, B, C, D
4	B, C

We create one variable per task, whose value will be the machine to which it will be assigned. The initial domains of the variables will be defined by the possible combinations in Table I. Since the tasks have to be assigned to different machines, we introduce an `alldifferent` *constraint. The constraints for this problem thus become:*

$$x_1 \in \{B, C, D, E\}, x_2 \in \{B, C\}, x_3 \in \{A, B, C, D\}, x_4 \in \{B, C\},$$
$$\texttt{alldifferent}(x_1, x_2, x_3, x_4).$$

We now want to model the task assignment problem from Example 3 graph-theoretically. First we introduce the definition of a bipartite graph.

DEFINITION 6 (Bipartite graph). *A* graph G *consists of a finite non-empty set of elements* V *called* nodes *and a set of unordered 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 tasks and machines from Example 3 and the possible combinations can be represented by the bipartite graph in Figure 1. Both tasks and machines are represented by nodes, and these two sets of nodes are connected by edges, giving the bipartition $(\mathit{Tasks}, \mathit{Machines})$. 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 7 (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*.*

DEFINITION 8 (Maximum matching). *A subset of edges in a graph* G *is called a* matching *if no two edges have a node in common. A*

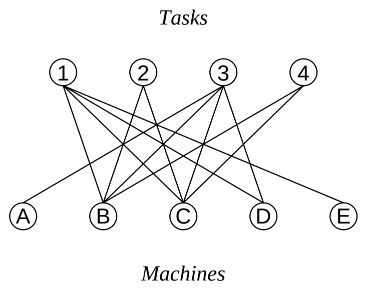


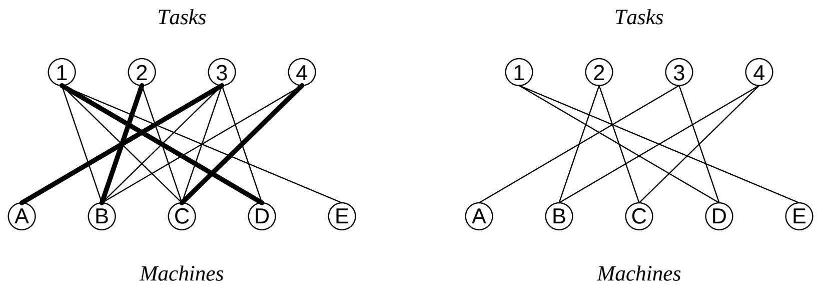
Figure 1. The value graph for the task assignment problem

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 tasks 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 2 (Régin [28]). *The constraint C :* `alldifferent`$(x_1, \dots, x_n)$ *is hyper-arc consistent iff every edge in its value graph $GV(C)$ belongs to a matching that covers X_C in $GV(C)$.*

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



a. A maximum matching b. The value graph after filtering

Figure 2. Filtering the value graph for the task assignment problem

Table II. Filtered machines for tasks

Task	Machines
1	D, E
2	B, C
3	A, D
4	B, C

3.2. The assignment problem

The matching problem from graph theory has a straightforward and equivalent counterpart in integer programming, called the *assignment problem*. Hence the `alldifferent` constraint can also be described as an assignment problem [35, 15]. We will first introduce the assignment problem and then show the equivalence with matching theory and the `alldifferent` constraint.

The assignment problem can be defined as follows: Given a set $I = \{1, \ldots, n\}$ and a set $J = \{1, \ldots, m\}$ with $m > n$, find a unique assignment of members of set I to members of set J. The standard integer programming formulation uses binary variables y_{ij} defined as

$$y_{ij} = \begin{cases} 1 & \text{if } i \in I \text{ is assigned to } j \in J \\ 0 & \text{otherwise} \end{cases}$$

where $i \in I$ and $j \in J$. To enforce the uniqueness of the assignment we define the constraints

$$\sum_{j \in J} y_{ij} = 1 \quad \forall i \in I \quad (\text{all } i \in I \text{ must be assigned}),$$

$$\sum_{i \in I} y_{ij} \leq 1 \quad \forall j \in J \quad (\text{no } j \in J \text{ can be assigned more than once}).$$

A connection with the matching problem from Section 3.1 can be established in the following way. Let the members of the sets I and J correspond to two node sets in a graph. Connect all nodes $i \in I$ to all nodes $j \in J$, thus forming the edge set $E = \{(i, j) | i \in I, j \in J\}$. Then we have constructed a bipartite graph (I, J, E). The assignment problem corresponds to finding a maximum matching in this bipartite graph.

In order to make a direct connection between the `alldifferent` constraint and the assignment problem, we have to make the following observation. First introduce variables $x_i \in J$ such that

$$y_{ij} = 1 \Leftrightarrow x_i = j.$$

Then a unique assignment of members of set I to members of set J can be enforced by

$$\texttt{alldifferent}(x_1, \ldots, x_n).$$

In the following example the task assignment problem of Example 3 is modeled as an integer programming assignment problem.

EXAMPLE 4 (Task assignment problem again). *We want to model the task assignment problem of Example 3 as an integer programming assignment problem as described before. Therefore, we first introduce the following binary variables:*

$$y_{tm} = \begin{cases} 1 & \textit{if } x_t = m \\ 0 & \textit{otherwise} \end{cases}$$

where $t \in T = \{1, 2, 3, 4\}$ and $m \in M = \{A, B, C, D, E\}$. There are several ways to express the allowed combinations from Table I. One way is to add constraints such as $y_{1A} = 0$ to state that task 1 cannot be assigned to machine A. However, we choose to multiply each y_{tm} variable with a binary coefficient a_{tm}. The coefficients form a matrix A that represents the possible combinations from Table I:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

The integer programming model now becomes:

$$\begin{aligned} \sum_{m \in M} a_{tm} y_{tm} = 1, & \qquad \forall t \in T \\ \sum_{t \in T} a_{tm} y_{tm} \leq 1, & \qquad \forall m \in M \\ y_{tm} \in \{0, 1\} & \quad \forall t \in T, \forall m \in M. \end{aligned}$$

3.3. Hall's theorem

A useful theorem to derive algorithms that ensure consistency for the `alldifferent` constraint is Hall's Theorem [13]. 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* $\texttt{alldifferent}(x_1, \ldots, x_n)$ *with respective variable domains $D_1, \ldots, D_n$ has a solution if and only if no subset $K \subseteq \{x_1, \ldots, x_n\}$ exists such that $|K| > |\cup_{x_i \in K} D_i|$.*

The following example shows an application of Theorem 1.

EXAMPLE 5 (Hall's Theorem applied). *Consider the following CSP:*

$$x_1 \in \{2,3\}, x_2 \in \{2,3\}, x_3 \in \{2,3\},$$
$$\texttt{alldifferent}(x_1, x_2, x_3).$$

This CSP is inconsistent, since there are only two values available, namely 2 and 3, for three variables that must be pairwise different. This can also be detected by Hall's Theorem. 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 model has no solution.*

4. Characterizations of Local Consistency Notions

4.1. 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 implemented in many constraint solvers, for instance in the system Chip [33], with a good performance.

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 $\frac{1}{2}(n^2 - n)$ 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 3. Let P_{dec} denote the decomposed CSP P in which all `alldifferent` constraints have been replaced by a sequence of disequalities.

PROPOSITION 3. $\Phi_{HA}(P) \preceq \Phi_A(P_{dec})$.

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`$(x_1, \ldots, x_n)$ and the corresponding decomposition in terms of disequalities, denoted by C_{dec}. If the set C_{dec} is not arc consistent w.r.t. a value $d_i \in D_i$, then the set C is also not hyper-arc consistent w.r.t. d_i. Indeed, when C_{dec} is not arc consistent w.r.t. d_i, 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 $(d_1, \ldots, x_n) \in C$, 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 6.

EXAMPLE 6 (Hyper-arc and arc consistency compared). *For some integer $n \geq 3$, consider the CSP's*

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

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

Then $\Phi_A(P_{dec}) \equiv P_{dec}$, 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 9 (Relational $(1, m)$ consistency, [8]). *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 relational $(1, 1)$-consistency.

Let $\Phi_{R(1,m)C}(P)$ denote the CSP after achieving relational $(1, m)$ consistency on a CSP P. In the following proposition, P is the CSP that consists only of the `alldifferent` constraint and a corresponding set of variables and domains.

PROPOSITION 4. $\Phi_{HA}(P) \equiv \Phi_{R(1,\frac{1}{2}(n^2-n))C}(P_{dec})$.

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 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 [20, 28]. Nevertheless, this filtering algorithm applies quite well to several problems, such as the n-queens problem ($n < 200$) [20, 26].

Other work on the comparison of the `alldifferent` constraints and the corresponding decomposition has for instance been done in [31] and [11].

4.2. Bounds consistency

The notion of bounds consistency for the `alldifferent` constraint was introduced by Puget [26]. We summarize his method in this section. Puget uses Hall's Theorem to construct an algorithm that achieves bounds consistency.

DEFINITION 10 (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 5 (Puget [26]). *The constraint* `alldifferent`$(x_1, \ldots, x_n)$ *where no domain D_i is empty, is bounds consistent iff*

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

Proposition 5 can be used to construct an algorithm that achieves bounds 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 $|I| \leq |K_I|$, 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 [23], 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 endpoints of the intervals. Under certain conditions the sorting can be performed in linear time, which gives a linear total running time. In particular, if the endpoints are from a range of size $O(n^k)$ for some constant k, the algorithm runs in linear time. This is for instance the case when the variables encode a permutation, as the `alldifferent` constraints in Example 1 where the n-queens problem was modeled.

EXAMPLE 7. *The following simple problem shows an application of the bounds 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 bounds consistent.

A comparison of bounds consistency and hyper-arc consistency with respect to the search space has been made by Schulte and Stuckey [30]. In particular, they pay attention to the `alldifferent` constraint.

4.3. Range consistency

Leconte introduced an algorithm that achieves range consistency [20]. To explain this algorithm we follow the same procedure as in the previous subsection. Leconte also uses Hall's Theorem to construct the algorithm.

DEFINITION 11 (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 6 (Leconte [20]). *The constraint* `alldifferent`$(x_1, \dots, x_n)$ *where no domain* D_i *is empty, is range consistent iff for each Hall set* $K \subseteq \{x_1, \dots x_n\}$*:* $D_i \cap I_K = \emptyset$ *for all* $x_i \notin K$.

We can deduce an algorithm from Proposition 6 in a similar way as we did for the algorithm for bounds 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 bounds consistency. Where the algorithm for bounds 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 7.

4.4. Hyper-arc consistency

A filtering algorithm that achieves hyper-arc consistency for constraints of difference was proposed by Régin [28]. A similar result was obtained independently by Costa [7]. This algorithm is based on matching theory, as described in Section 3.1.

An algorithm that achieves hyper-arc consistency for the `alldiff-erent` 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 [3]. 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.a, $(3, F, 4, M, 3)$ is an even alternating cycle of length 4. Node 6 is a free node.

PROPOSITION 7 (Berge [3]). *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.*

```
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
```

Figure 3. An algorithm for achieving hyper-arc consistency

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 3. 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 *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 [16]. Their algorithm still remains essentially the best known [6].

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 [28]. For this problem we can use an algorithm by Tarjan that runs in $O(n + m)$ time on graphs with n nodes and m edges [28, 32]. In the algorithm from Figure 3, 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 [25]. 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, i.e. $D_i \leq d$ for all $i \in \{1, \dots, n\}$, the time complexity of the general algorithm is $O(\frac{d!}{(d-n)!})$, whereas the time complexity of the above algorithm is $O(dn\sqrt{n})$.

During the computation process, other constraints than `alldifferent` might also be used to remove values. In that case, we must update our `alldifferent` constraint. As was presented in [28], we can make use of our current value graph and our current maximum matching to compute a new maximum matching. This is less time consuming than restarting the algorithm from scratch. This same idea has been used by Barták to make the `alldifferent` constraint dynamic with respect to the addition of variables during computation [1].

4.5. Practical comparison of consistencies

In this subsection we consider the practical usage of the four different types of local consistency that were presented. We make a comparison by applying them to the n-queens problem, which was defined in the introduction. These results cannot be generalized in the sense that using local consistency A is always faster than using local consistency B. As was discussed before, different problems behave differently with respect to a certain local consistency, the search space to be explored and the time spent on computing the local consistency. One should therefore always make a trade-off between the time saved by pruning the search space and the time spent on computing the actual pruning. Nevertheless, these results can give some insight in the use of different local consistencies when solving the n-queens problem.

The left part of Table III shows the time spent by each of the four algorithms to solve the n-queens problem, where n is 8 up to 12. The right part shows the number of backtracks during the solution process of the same set of problems. Here AC stands for arc consistency on the decomposed CSP, BC for bounds consistency, RC for range consistency and HAC for hyper-arc consistency, as was defined in Section 2. The table comes from the article by Puget [26].

Note that the simplest algorithm with the least pruning power solves all problems faster than the other algorithms. The pruning power can be derived from the number of backtracks. The less the number of

Table III. Comparing four local consistencies for the n-queens problem

	Time (s)				Number of backtracks			
n	AC	BC	RC	HAC	AC	BC	RC	HAC
8	.17	.23	.21	.26	289	260	260	239
9	.64	.86	.84	.98	1111	947	949	854
10	2.6	3.4	3.3	3.8	5072	4294	4295	3841
11	11.3	15.1	14.8	16.9	22124	18757	18763	16368
12	54.7	73.9	71.8	82.0	103956	87225	87263	74936

backtracks, the higher the pruning power. In this particular example the higher pruning power of the more sophisticated algorithms does not pay off with respect to the running time.

5. The symmetric `alldifferent` constraint

A particular case of the `alldifferent` constraint, the symmetric `all-different` constraint, was introduced by Régin [29]. In this case, the variables and values are defined from the same set S. Every variable represents an element e of S and its domain represents elements of S that are compatible with e. The constraint `symm_alldifferent` states that all variables must take different values, as is the case with the common `alldifferent` constraint. Furthermore, if the variable representing element i is assigned to the value representing element j, then the variable representing the element j must be assigned to the value representing element i. The latter is the symmetry-part of the constraint that can be exploited for filtering purposes. The `symm_alldifferent` constraint is for instance applicable to the scheduling of teams in a sports competition, where variables represent teams and each team has to be matched to another team.

In a CSP, the symmetric `alldifferent` constraint can also be expressed as an `alldifferent` constraint together with one or more constraints that preserve the symmetry. Another representation can be made by using the `cycle` constraint, where each cycle must contain two nodes [2]. However, the symmetric `alldifferent` constraint contains more global information about the CSP than the common `alldiff-erent` constraint together with additional constraints. This additional information can be used for pruning. The following example, taken from [29] shows exactly this.

EXAMPLE 8 (Global information). *Consider a set of three people that have to be grouped in pairs. Each person can only be paired to one other person. This problem can be represented as a CSP by introducing a set of people $S = \{p_1, p_2, p_3\}$ that are pairwise compatible. These people are represented both by a set of variables x_1, x_2 and x_3 and by a set of values v_1, v_2 and v_3, where x_i and v_i represent p_i. Then the CSP*

$$\begin{array}{l} x_1 \in \{v_2, v_3\} \\ x_2 \in \{v_1, v_3\} \\ x_3 \in \{v_1, v_2\} \\ x_i = v_j \Leftrightarrow x_j = v_i, \ \forall i \in \{1,2,3\} \ \forall j \in \{i, \ldots, 3\} \\ \texttt{alldifferent}(x_1, x_2, x_3) \end{array}$$

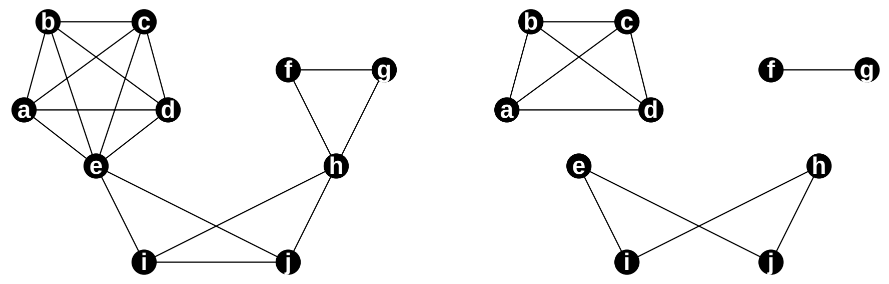


Figure 4. Example of filtering the `symm_alldifferent` constraint

expresses exactly the `symm_alldifferent` *constraint. This CSP is hyper-arc consistent. However, the problem is inconsistent and the global* `symm_alldifferent` *constraint would have recognized this.*

Similarly to the common `alldifferent` constraint, the `symm_alldifferent` constraint can be expressed by a graph. In this case nodes represent the variables inside the constraint and there is an edge between two nodes x_i and x_j whenever both the element that is represented by x_j is represented in the domain of x_i and the element that is represented by x_i is represented in the domain of x_j. This graph is called the *contracted value graph* of the constraint. In Figure 4 an example of such a graph is given. A solution to the `symm_alldifferent` constraint corresponds to a matching in the contracted value graph that covers the set of variables. Property 8 uses this information to make the constraint hyper-arc consistent.

PROPOSITION 8 (Régin [29]). *The constraint C:* `symm_alldifferent` $(x_1, \ldots, x_n)$ *is hyper-arc consistent iff every edge in its contracted value graph belongs to a matching that covers X_C in the contracted value graph.*

Figure 4.b shows the contracted value graph of Figure 4.a that has been made hyper-arc consistent. Consider for instance the subgraph induced by nodes f, g and h. Obviously nodes f and g have to be paired in order to satisfy the `symm_alldifferent` constraint. Hence, node h cannot be paired to f nor g, and the corresponding edges can be removed. The same holds for the subgraph induced by a, b, c, d and e, where the nodes a, b, c and d must form pairs.

The difference with the common `alldifferent` constraint is that the contracted value graph does not need to be bipartite. Therefore we cannot blindly apply the machinery from Section 4.4 to achieve hyper-arc consistency for the `symm_alldifferent` constraint. However, we can apply similar reasoning to this case. There are algorithms for finding a maximum matching in nonbipartite graphs, for instance the one by Edmonds [9]. Régin proposes an algorithm that makes a `symm_alldifferent` constraint hyper-arc consistent that has a running time of $O(nm)$, together with an algorithm that has a time complexity of $O(m)$ but does not ensure hyper-arc consistency [29]. Again, n is the number of nodes and m is the number of edges in the graph.

6. Cost-based Filtering Algorithm

In optimization problems, one tries to optimize an objective function, subject to a set of constraints. For example, the assignment problem in Section 3.2 can be extended to the *minimum* assignment problem, i.e. find an assignment with the lowest objective function value. In the minimum assignment problem, the objective function is defined as follows. Each possible assignment of $i \in I$ to $j \in J$ contributes a cost of c_{ij} to the objective, where the sets I and J are defined as in Section 3.2. Then the objective function to be minimized is defined as

$$\sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} \tag{1}$$

where $y_{ij} = 1$ if i is assigned to j and 0 otherwise, as defined in Section 3.2.

A standard technique to solve optimization problems, both in constraint programming and integer programming, is branch-and-bound. This tree search technique makes use of the best known solution so far and an estimate for the current branch to be explored. This estimate is usually based upon a relaxation of the optimization problem. If we want to solve a minimization problem, the best known solution provides us with an upper bound on the global optimal solution, and the estimate of the current branch with a lower bound. If the lower bound exceeds the upper bound, we can abandon the current branch and the underlying subtree, because this subtree cannot contain a better solution than the currently best one.

Cost-based domain filtering [10] makes use of this property of branch-and-bound, but applies it as a domain filtering technique from constraint programming. Consider a CSP that contains an `alldifferent` constraint, together with an objective function similar to (1). We can relax this CSP by treating the `alldifferent` constraint and the objective function as a minimum assignment problem and ignoring the other constraints. The minimum assignment problem can for instance be solved with the Hungarian Algorithm [18, 4]. An important feature of the Hungarian Algorithm is that it is incremental. The first time it is called it runs in $O(n^3)$ time and all next computations run in $O(n^2)$ time. This algorithm returns, if possible, the optimal feasible assignment, together with a reduced cost matrix. The reduced cost matrix contains information about the increase of the objective function when we change one of the assignments. Consider for instance the following reduced cost matrix $\overline{c}$, where the rows correspond to variables x_1, x_2 and x_3, and the columns to domain values 1, 2 and 3.

$$\overline{c} = \begin{bmatrix} 2 & 0 & 5 \\ 3 & 6 & 0 \\ 0 & 3 & 4 \end{bmatrix}.$$

In this case, the optimal assignment is $x_1 = 2$, $x_2 = 3$ and $x_3 = 1$, say with an objective value of 10. If we now change the assignment $x_1 = 2$ to $x_1 = 3$, the reduced cost matrix tells us that the objective value will increase with at least 5, which makes a total of at least 15.

We can compare the objective value of the relaxed CSP, the lower bound, with the value of best known solution to the CSP, the upper bound. If the lower bound is larger than the upper bound, we know that the current subproblem does not contain the global solution to the CSP. If the lower bound is smaller, we can apply the actual cost-based filtering, by investigating the reduced cost matrix. Namely, in the underlying subtree we try to find the global solution to the CSP by changing the current assignment, without exceeding the upper bound. We can use reduced costs to give us an estimate of the increase of the objective function when we would change a single assignment. If the lower bound plus the value $\overline{c}_{ij}$ is larger than the upper bound, we know that within the current underlying subtree j will never be assigned to i, since otherwise the objective function value would exceed the upper bound. From Section 3.2 we know the mapping between the assignment problem variables y_{ij} and the CSP variables x_i:

$$y_{ij} = 1 \Leftrightarrow x_i = j.$$

For all possible assignments in the current subproblem we can check the validity as described above. If an i,j-assignment is not valid, y_{ij} can never be 1. Hence we can remove the value j from the domain of variable x_i.

EXAMPLE 9 (Traveling salesman problem). *In the traveling salesman problem we are given a finite set of nodes V and a cost c_{ij} of travel between each pair $i,j \in V$. A tour is a circuit through all nodes in V, $|V| = n$, such that no node is visited twice. The traveling salesman problem (TSP) is the task of finding a tour with minimal cost. We can* *model the TSP with the use of an* `alldifferent` *constraint, as shown in the following CSP (see [15]):*

$$\begin{array}{lll} \min & \sum_{i=1}^{n} c_{x_i x_{i+1}} & \\ \text{subject to} & \texttt{alldifferent}(x_1, \ldots, x_n) & \\ & x_i \in \{1, \ldots, n\}, & i \in \{1, \ldots, n\} \\ & x_{n+1} = x_1 & \end{array}$$

The variables x_i represent the order in which the nodes are visited, with a total cost of $\sum_{i=1}^{n} c_{x_i x_{i+1}}$.

We can also describe the TSP as an integer programming model. A classical integer programming formulation for the TSP is as follows:

$$\begin{array}{rlll} \min & \sum_{i \in V} \sum_{j \in V} c_{ij}\, y_{ij} & & (2) \\ \text{subject to} & \sum_{i \in V} y_{ij} = 1, & j \in V & (3) \\ & \sum_{j \in V} y_{ij} = 1, & i \in V & (4) \\ & \sum_{i \in S} \sum_{j \in V \setminus S} y_{ij} \geq 1, & S \subset V, S \neq \emptyset & (5) \\ & y_{ij} \text{ integer}, & i, j \in V & (6) \end{array}$$

where $y_{ij} = 1$ if and only if edge (i, j) is in the optimal solution. Constraints (3) and (4) impose in-degree and out-degree of each vertex equal to one, whereas constraints (5) exclude all possible subtours. A well known relaxation of the TSP is obtained by eliminating the exponential number of constraints (5), yielding the minimum assignment problem.

The constraint programming variables x_i and the integer programming variables y_{ij} are related as follows:

$$x_{k+1} = j \Leftrightarrow y_{x_k j} = 1.$$

As described before, we can apply the Hungarian algorithm to the assignment problem. The resulting reduced cost matrix can be used to identify variables y_{uv} that will never be assigned in the current branch. Consequently, we can remove value v from the domain of the corresponding variable in the CSP model.

7. Summary

In this paper, an overview of the `alldifferent` constraint has been given. First, four notions of local consistency were introduced that were studied for the `alldifferent` constraint. These are arc consistency on the decomposed system of disequalities, range consistency, bounds consistency and hyper-arc consistency. The connection with matching theory from graph theory and the assignment problem from integer programming has been discussed. These fields provide the bases of several filtering algorithms for the `alldifferent` constraint. A comparison of four notions of local consistency has been made and algorithms to achieve them have been presented. Furthermore, the symmetric `alldifferent` constraint has been discussed as a particular kind of `alldifferent` constraint. Finally, the notion of cost-based domain filtering has been applied to the `alldifferent` constraint.

Acknowledgements

Thanks to Krzysztof Apt, Frédéric Goualard and Sebastian Brand for many helpful comments and suggestions on earlier drafts of this paper. Also thanks to Michela Milano for commenting on the cost-based filtering section.

References

1. R. Barták. Dynamic global constraints: A first view. In *Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001. `http://www.arxiv.org/html/cs/0110012`.
2. N. Beldiceanu. Global Constraints as Graph Properties on Structured Network of Elementary Constraints of the Same Type. Technical Report T2000/01, SICS, 2000.
3. C. Berge. *Graphs and hypergraphs.* North-Holland, 1973.
4. G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the assignment problem. *Annals of Operations Research*, 13:193–223, 1988.
5. H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, 5:1–16, 1999.
6. W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* Wiley, 1998.
7. M.-C. Costa. Persistency in maximum cardinality bipartite matchings. *Operations Research Letters*, 15(3):143–149, 1994.
8. R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173:283–308, 1997.
9. J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
10. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Principles and Practice of Constraint Programming - CP'99*, volume LNCS 1713 of *Lecture Notes in Computer Science*, pages 189 – 203. Springer, 1999.
11. I. Gent, K. Stergiou, and T. Walsh. Decomposable Constraints. In *New Trends in Constraints - Proceedings of the joint ERCIM/Compulog Net Workshop 1999*, volume LNCS 1865 of *Lecture Notes in Computer Science*, pages 134–149. Springer, 2000.

12. A.M.H. Gerards. Matching. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 135 – 224. Elsevier Science, 1995.
13. P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935.
14. W.J. van Hoeve. The Alldifferent Constraint: A Survey. In *Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001. `http://www.arxiv.org/html/cs/0110012`.
15. J. Hooker. *Logic-Based Methods for Optimization - Combining Optimization and Constraint Satisfaction.* Wiley-interscience series in discrete mathematics and optimization. John Wiley and Sons, 2000.
16. 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.
17. J.N. Hooker and G. Ottosson and E.S. Thorsteinsson and H.-J. Kim. A Scheme for Unifying Optimization and Constraint Satisfaction Methods. *Knowledge Engineering Review, special issue on AI/OR*, 15(1):11–30, 2000.
18. H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
19. J.-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence. An International Journal*, 10(1):29–127, 1978.
20. 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.
21. L. Lovász and M. D. Plummer. *Matching Theory.* North-Holland, Amsterdam, 1986.
22. K. Marriot and P. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, Cambridge, MA, 1998.
23. K. Mehlhorn and S. Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In R. Dechter, editor, *Principles and Practice of Constraint Programming - CP2000*, volume LNCS 1894 of *Lecture Notes in Computer Science*, pages 306 – 319. Springer, 2000.
24. M. Milano, G. Ottosson, P. Refalo, and E. Thorsteinsson. Global constraints: When constraint programming meets operation research, 2001. Submitted to *INFORMS Journal on Computing, Special Issue on the Merging of Mathematical Programming and Constraint Programming.*
25. R. Mohr and G. Masini. Good Old Discrete Relaxation. In *European Conference on Artificial Intelligence (ECAI-88)*, pages 651–656, Munchen, Germany, 1988.
26. J.-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI '98)*, pages 359 – 366, 1998.
27. J.-F. Puget and M. Leconte. Beyond the glass box: Constraints as objects. In *International Logic Programming Symposium*, pages 513–527, 1995.
28. J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 1, pages 362–367, 1994.
29. J.-C. Régin. The symmetric alldiff constraint. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 420–425, 1999.

30. C. Schulte and P. J. Stuckey. When do bounds and domain propagation lead to the same search space. In H. Søndergaard, editor, *Proceedings of the Third International Conference on Principles and Practice of Declarative Programming*, pages 115–126, Florence, Italy, 2001. ACM Press.
31. 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.
32. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
33. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge, MA, 1989.
34. M. Wallace, S. Novello, and J. Schimpf. ECLiPSe: A platform for constraint logic programming. Technical report, IC-Parc, Imperial College, London, 1997.
35. H.P. Williams and Hong Yan. Representations of the all_different Predicate of Constraint Satisfaction in Integer Programming. *INFORMS Journal on Computing*, 13(2):96–103, 2001.