The Symmetric Alldiff Constraint

Jean-Charles RÉGIN
ILOG
Les Taissounières HB2
06560 Valbonne, FRANCE
e-mail : regin@ilog.fr

Abstract

The *symmetric* alldiff constraint is a particular case of the alldiff constraint, a case in which variables and values are defined from the same set S. That is, every variable represents an element e of S and its values represent the elements of S that are compatible with e. This constraint requires that all the values taken by the variables are different (similar to the classical alldiff constraint) and that if the variable representing the element i is assigned to the value representing the element j, then the variable representing the element j is assigned to the value representing the element i. This constraint is present in many real-world problems, such sports scheduling where it expresses matches between teams. In this paper, we show how to compute the arc consistency of this constraint in $O(nm)$ ($m = \sum_i |D(i)|$), where n is the number of involved variables and $D(i)$ the domain of the variable i. We also propose a filtering algorithm of less complexity ($O(m)$).

1 Introduction

Constraint Satisfaction Problems (CSPs) involve finding values for problem variables subject to constraints on which combinations are acceptable. They are more and more used in real-life applications, such as frequency allocation, crew scheduling, time tabling, car sequencing, etc. [Simonis, 1996].

The general task of finding a solution in a constraint network being NP-hard, many researchers have concentrated on improving the efficiency of solving a CSP.

Currently, it seems that a look-ahead approach is the most promising way. The purpose of this technique is to look at the values of the variables that are not yet instantiated and to remove values that cannot lead to a solution w.r.t. the current partial instantiation. Thus, it anticipates the detection of some failures by using a particular treatment after each modification of domain variables. A filtering algorithm is one such particular treatment. With respect to a partial instantiation, it removes once and for all certain inconsistencies that would have been discovered several times otherwise.

Techniques based on filtering algorithms are thus quite important. Particularly, arc consistency caught the attention of many researchers, who then discovered a large number of algorithms.

Furthermore, it is necessary to deal directly with the arity of the constraints because nonbinary constraints lose much of their semantics when encoded into a set of binary constraints. (See [Régin, 1994].) This encoding leads, for example, to behavior that prunes much less for filtering algorithms handling it.

When the semantics of a nonbinary constraint is not known *a priori*, GAC-Schema [Bessière and Régin, 1997] can be used to achieve arc consistency of the constraint. However, this algorithm does not perform as well with known semantics. In such a situation, it is particularly interesting to develop a specific filtering algorithm, as it was done for the well known alldiff constraint [Régin, 1994]. This approach leads to an important gain in time and in space for solving a CSP, even if the filtering algorithm does not achieve arc consistency. For instance, the diff-n or cumulative constraints are really useful in practice to solve real-world problems, as it has been shown by Simonis, although arc consistency is not achieved for these constraints.

In this paper, we study a new constraint and propose some filtering algorithms for it. Some of those algorithms ensure arc consistency; others are weaker.

The symmetric alldiff constraint

Consider a set of people to be grouped by pairs according to predefined compatibilities such that each person is paired exactly once. This problem can be modeled as a constraint satisfaction problem in which each person is associated with one variable and one value. The domain of a variable associated with a person p is defined by the values which are associated with a person compatible with p. For instance, consider the simple problem defined on a set of three people p_1, p_2, p_3 that are all compatible. The CSP will then involve three variables x_1, x_2, x_3, where x_i is associated with p_i and three values v_1, v_2, v_3, where v_i is associated with

p_i. Moreover, $D(x_1) = \{v_2, v_3\}$, $D(x_2) = \{v_1, v_3\}$ and $D(x_3) = \{v_1, v_2\}$.

Since we want to pair all the variables with different values, we add an alldiff constraint involving all the variables. Constraints stating that "if any variable x is associated with a variable y, then y must be associated with x" can be defined by means of the σ function. This function is defined as follows: $\sigma(x)$ is the value v that is associated with the same person as the variable x, and $\sigma(v)$ is the variable x that is associated with the same person as the value v. Of course, we have $\sigma(x) = v \Leftrightarrow x = \sigma(v)$. Then, for each variable x and for each value v, we define the constraint: $(x = v \Leftrightarrow \sigma(v) = \sigma(x))$.

The CSP we have just defined can be viewed as only one constraint that we will call ***symmetric alldiff constraint***. This constraint requires that all the values taken by the variables are different (similar to the classical alldiff constraint) and that if the variable representing the element i is assigned to the value representing the element j, then the variable representing the element j is assigned to the value representing the element i.

The example problem we consider has no solution. However, the CSP that has just been built is arc consistent[1]. Thus it is important to be able to efficiently handle symmetric alldiff constraints.

These constraints arise in some problems such crew scheduling (two pilots must be in a cockpit at the same time), nurse rostering (two nurses are required for certain operations) or sports scheduling. In the latter problems, one of the main tasks is to compute a set of matches between teams such that each team plays against another team, and each team plays exactly once for each period of time under consideration. There exist compatibility constraints between teams. For instance, during the winter period, travel has to be limited. Therefore, for a given period, the problem we have to solve for each period is exactly a symmetric alldiff constraint.

A symmetric alldiff constraint can be expressed by a graph, in which nodes represent variables and there is an edge between two nodes x_1 and x_2 if and only if $\sigma(x_1) \in D(x_2)$ and $\sigma(x_2) \in D(x_1)$. (See Figure 1).

Initially this graph corresponds to the compatibility graph. During the search for a solution, it is built from the current domain of the variables.

Since this new constraint corresponds to the definition of a particular CSP involving several constraints, there is an equivalence between the consistency of this new constraint and the existence of a solution for the CSP. This equivalence means that an algorithm checking the consistency of a symmetric alldiff constraint is more efficient for the resolution of the problem than is the conjunction of all the algorithms checking the consistency (or achieving arc consistency) of the other constraints involved in the first model.

It is quite important to emphasize this point in order to understand that there is great interest in defining specific global constraints for which efficient algorithms computing the consistency of these constraints are known. For instance, consider the problem given by the left graph in Figure 1 and two models of this problem. First, this problem is represented by a classical alldiff constraint and constraints ensuring the symmetries. Second, the problem is represented by only one symmetric alldiff constraint. The subproblems part I and part II of this graph are odd size cliques. Thus, clearly, if c is not instantiated to neither a nor b then there is no solution[2]. Unfortunately, the CSP defined by the first model is arc consistent, and so no value is removed. The right graph of the figure shows the achievement of arc consistency for the symmetric alldiff constraint.

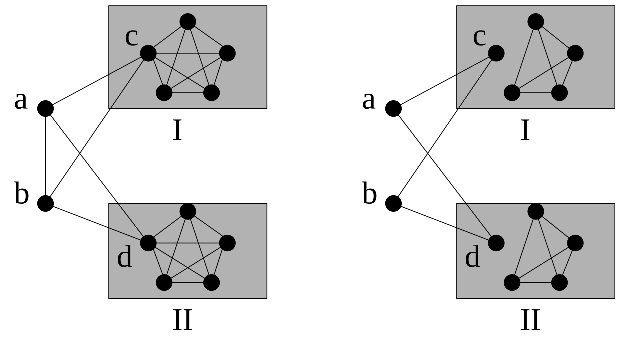

Figure 1: An example of a symmetric alldiff constraint. Nodes represent teams, and edges compatibilities between teams. The left graph is the initial graph, the right graph is the graph obtained after achieving arc consistency.

The consistency of an alldiff constraint can be computed by searching for a maximum matching in a bipartite graph. We will show that the consistency of a symmetric alldiff constraint can also be achieved by searching for a maximum matching in a graph which is not necessarily bipartite. This problem can be easily solved by using, for instance, Edmonds's algorithm [Edmonds, 1965]. We will also present an original algorithm for achieving arc consistency for a symmetric alldiff constraint. Unfortunatly, this algorithm is not incremental. Hence, we will propose a filtering algorithm, which does not necessarily achieve arc consistency, but which has a remarkable complexity.

The paper is organized as follows. First we give some preliminaries about constraint network and matching theory. Then, we formally present the symmetric alldiff constraint, and we explain how to compute the consistency of this constraint. In the next section, an algorithm achieving arc consistency for this constraint is fully detailed. After it, we propose a filtering algorithm which has lower complexity. Then we conclude.

[1]For instance, $\{(x_1, v_2), (x_2, v_3), (x_3, v_1)\}$ satisfies the alldiff constraint.

[2]When there is no ambiguity we will say that node x is instantiated to y instead of saying node x is instantiated to $\sigma(y)$ and node y is instantiated to $\sigma(x)$.

2 Preliminaries

2.1 Constraint network

A finite ***constraint network*** $\mathcal{N}$ is defined as a set of n ***variables*** $X = \{x_1, \ldots, x_n\}$, a set of current finite ***domains*** $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ where $D(x_i)$ is the finite set of possible ***values*** for variable x_i, and a set $\mathcal{C}$ of ***constraints*** between variables. We introduce the particular notation $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ to represent the set of initial domains of $\mathcal{N}$.[3]

Then, a constraint C on the ordered set of variables $X(C) = (x_{i_1}, \ldots, x_{i_k})$ is a subset $T(C)$ of the Cartesian product $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_k})$ that specifies the ***allowed*** combinations of values for the variables $x_{i_1} \times \ldots \times x_{i_k}$. An element τ of $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_k})$ is called a ***tuple on*** $X(C)$ and $\tau[k]$ is the k^{th} value of τ.

$ind(C, x)$ is the position of variable x in $X(C)$; $\#(v, \tau)$ is the number of occurences of the value v in the tuple τ; and $D(X)$ denotes the union of the domain of the variables of X.

A tuple τ on $X(C)$ is ***valid*** if $\forall (x, a) \in \tau, a \in D(x)$. A value $a \in D(x)$ is ***consistent with*** C iff $x \notin X(C)$, or $\exists \tau \in T(C)$, such that $a = \tau[ind(C, x)]$ and τ is valid. C is ***arc consistent*** iff $\forall x \in X(C), D(x) \neq \varnothing$ and $\forall a \in D(x), a$ is consistent with C. We achieve arc consistency of C by removing all values not consistent with C.

The ***value graph*** of a constraint C is the bipartite graph $GV(C) = (X(C), D(X(C)), E)$ where $\{x_i, a\} \in E$ iff $a \in D(x_i)$.

2.2 Matching theory

Most of these definitions are due to [Tarjan, 1983].

If $\{u, v\}$ is an edge of a graph, then we say that u and v are the ***ends*** or the ***extremities*** of the edge. n are the number of nodes and m the number of edges of a graph. $G - \{u, v\}$ denotes the graph G in which the nodes u and v have been removed. $G - \{\{u, v\}\}$ denotes the graph G in which the edge $\{u, v\}$ has been removed. A ***matching*** M on a graph is a set of edges no two of which have a common vertex. The ***size*** $|M|$ of M is the number of edges it contains. The ***maximum matching problem*** is that of finding a matching of maximum size. M ***covers*** X when every vertex of X is an endpoint of some edge in M.

Let M be a matching. An edge in M is a ***matching edge***; every edge not in M is ***free***. A vertex is ***matched*** if it is incident to a matching edge and ***free*** otherwise. For any matched vertex v, ***mate***(v) denotes the vertex w such that $\{v, w\}$ is a matching edge.

A ***path*** in a graph ***from*** v_1 ***to*** v_k is a sequence of vertices $[v_1, v_2, ..., v_k]$ such that $\{v_i, v_{i+1}\}$ is an edge for $i \in [1, ..., k-1]$. The path is ***simple*** if all its vertices are distinct. A path is a ***cycle*** if $k > 1$ and $v_1 = v_k$. An ***alternating path or cycle*** is a simple path or cycle whose edges are alternately matching and free. The ***length*** of an alternating path or cycle is the number of edges it contains.

[3] Indeed, we consider that any constraint network $\mathcal{N}$ can be associated with an initial domain $\mathcal{D}_0$ (containing $\mathcal{D}$), on which constraint definitions were stated.

3 The symmetric alldiff constraint

Definition 1 *Let X be a set of variables and σ be a one-to-one mapping from $X \cup D(X)$ to $X \cup D(X)$ such that*
$\forall x \in X$: $\sigma(x) \in D(X)$; $\forall a \in D(X)$: $\sigma(a) \in X$ and $\sigma(x) = a \Leftrightarrow x = \sigma(a)$.
*A **symmetric alldiff constraint** defined on X is a constraint C associated with σ such that:*

$$T(C) = \{\ \tau \text{ such that } \tau \text{ is a tuple on } X \text{ and } \forall a \in D(X) : \#(a, \tau) = 1 \text{ and } a = \tau[ind(C, x)] \Leftrightarrow \sigma(x) = \tau[ind(C, \sigma(a))]$$

It is denoted by symalldiff(X, σ)

The consistency of the classical alldiff constraint is computed by searching for the existence of a matching in the value graph that covers all the variables, and arc consistency is achieved by identifying all the edges that can never belong to a matching that covers all the variables. Our problem is really close to that one with one major difference: the graph under consideration can be nonbipartite.

The problem we have to solve is called symmetric matching. Symmetric matching in a $2n$-node bipartite graph is, indeed, really no different from matching in an n-node nonbipartite graph. Consider the value graph $GV(C)$ of a symmetric alldiff constraint. This graph is bipartite. Now, we can modify this graph by contracting any variable x with value $a = \sigma(x)$ into a single vertex. The edge between x and a is deleted, and the other edges that have x or a as an endpoint are replaced by edges having the contracting vertex as endpoint and their other extremity unchanged. The graph we get in that way, denoted by $CGV(C)$, is no longer bipartite and is called the ***contracted value graph*** of a symmetric alldiff constraint. More formally: $CGV(C) = (X(C), E)$ where $\{x_1, x_2\} \in E$ iff $\sigma(x_1) \in D(x_2)$ and $\sigma(x_2) \in D(x_1)$.

There is a correspondance between a matching which covers $X(C)$ in $CGV(C)$ and a tuple of $T(C)$.

Proposition 1 *Given C=symalldiff(X, σ), every tuple of $T(C)$ corresponds to a set A of edges in $CGV(C)$ such that for each vertex $x \in X(C)$, x is an end of exactly one edge. And a matching M in $CGV(C)$ which covers $X(C)$ corresponds to an element of $T(C)$*

proof: An element of $T(C)$ corresponds to a matching that covers $X(C)$ in $CGV(C)$, by construction of $CGV(C)$. And from a matching in $CGV(C)$ covering $X(C)$, we can build a tuple that satisfies the constraint, by definition of the symmetric alldiff constraint.

Therefore, we have:

Corollary 1 *A constraint C=symalldiff(X, σ) is consistent iff there exists a matching that covers $X(C)$ in $CGV(C)$.*

Since $CGV(C)$ can be nonbipartite, an algorithm searching for maximum matching in a nonbipartite graph has to be used, like the blossom-shrinking algorithm of Edmonds. An implementation of this algorithm in $O(mn)$ is fully detailed in [Tarjan, 1983]. The advantage of this algorithm is its incrementality. Suppose that we start with a matching of size k, and there exists a matching of size $n/2$; then this matching can be computed in $O((n/2-k)m)$. This point is important if we systematically check the consistency of the constraint during the search for solution.

On the other hand, [Micali and Vazirani, 1980] proposed a complex algorithm in $O(\sqrt{n}m)$.

For computing the consistency of a symmetric alldiff constraint, it is also necessary to update the contracted value graph. Precisely, when a value is removed from the domain of a variable, the corresponding edge must be deleted. All these modifications need at most $O(m)$ operations. Therefore, we can consider that the consistency of a symmetric alldiff constraint can be computed in $O(\sqrt{n}m)$.

3.1 Arc consistency

For the sake of clarity, we will consider that $C =$ symalldiff(X, σ) is a symmetric alldiff constraint. We will also consider that the consistency of C has been checked; thus M a matching which covers $X(C)$ in $CGV(C)$ is known.

First, for every variables x and y of $X(C)$, we have to ensure that if $\sigma(y)$ is removed from $D(x)$ then $\sigma(x)$ is also removed from $D(y)$. This can be easily done in $O(1)$ for each deletion.

From proposition 1 and by definition of arc consistency, we have:

Corollary 2 *A value a of a variable x is consistent with C if and only if the edge $\{x, \sigma(a)\}$ belongs to a matching that covers $X(C)$ in $CGV(C)$.*

Thus, the arc consistency of C is achieved by removing all the values (x, a) such that the edge $\{x, \sigma(a)\}$ does not belong to any matching that covers $X(C)$ in $CGV(C)$. Therefore, there is a simple algorithm achieving arc consistency: For each free edge $\{u, v\}$, we search for a matching in $CGV(C) - \{u, v\}$ that covers $X(C) - \{u, v\}$. If such a matching exists, then the edge $\{u, v\}$ belongs to a matching that covers $X(C)$; otherwise, it does not. To compute the matchings that cover $X(C) - \{u, v\}$ in $CGV(C) - \{u, v\}$, we start from $M - \{\{u, mate(u)\}, \{v, mate(v)\}\}$. So we need only $O([n/2 - (n/2 - 2)]m) = O(m)$ operations. Since there are m edges in $CGV(C)$, the complexity to achieve arc consistency is $O(m^2)$.

We can improve this complexity by using the following proposition which has been used for efficiently computing arc consistency in the classical alldiff constraint. This proposition, indeed, does not depend on whether the graph is bipartite or not.

Proposition 2 ([Berge, 1970]) *An edge belongs to some but not all maximum matchings, iff, for an arbitrary maximum matching, it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.*

ArcConsistency($CGV(C), M$)
  for *each vertex x in $CGV(C)$* **do**
    searchForEvenAlternatingCycle$(x, CGV(C), M)$
    for *each edge $\{u, x\}$ not marked valid* **do**
      remove $\{u, x\}$ from $CGV(C)$
      remove $\sigma(u)$ from $D(x)$ and $\sigma(x)$ from $D(u)$

Algorithm 1: An arc consistency algorithm for a symmetric alldiff constraint.

M is a matching which covers $X(C)$; thus no vertex of $X(C)$ is free. Therefore, a value a of a variable x is consistent with C iff the edge $\{x, \sigma(a)\}$ belongs to an even alternating cycle. If the edge $\{x, \sigma(a)\}$ belongs to M, then the value a of x is consistent with C. Thus, the value a of x is not consistent with C if and only if the edge $\{x, \sigma(a)\}$ is free and if it does not belong to an alternating cycle. Such an alternating cycle is formed by an alternating path $[x, \sigma(a), ..., mate(x)]$ and the matched edge $\{mate(x), x\}$. Therefore the problem of the search for an alternating cycle is equivalent to the problem of the search for an alternating path from x to $mate(x)$ in $CVG - \{\{mate(x), x\}\}$.

We can give the algorithm achieving arc consistency. For each matching edge $\{u, v\}$ in $CGV(C)$, we search for an alternating path from u to v in $CGV(C) - \{\{v, u\}\}$, but we do not stop if we reach an edge with v as its extremity. If such an edge is reached, it is marked as "valid" and the algorithm continues as if the edge does not exist. When there are no more edges to study, the algorithm stops. All edges $\{v, y\}$ different from $\{v, u\}$ that are not marked valid cannot belong to an even alternating cycle. Afterwards, we apply the same reasoning by starting from v in order to identify the valid edges $\{x, u\}$. (See Algorithm 1.)

The problem which remains is the computation of alternating paths.

An alternating path from x to $mate(x)$ in $CGV(C) - \{\{mate(x), x\}\}$ can be found by applying the following procedure due to Edmonds. x is marked even; then we mark ***even*** a vertex reached from a matching edge, and ***odd*** a vertex reached from a free edge. Thus, from any even vertex u, we traverse the free edge having an extremity in u. And from any odd vertex v, we traverse the matching edge linked to v. Note that a vertex is even if it is an even distance from the starting vertex and odd otherwise. This method works fine for a bipartite graph because there is no odd-length cycle, so a vertex marked even can never be reached from a vertex also marked even.

On nonbipartite graphs, there is a subtle difficulty: a vertex can appear on an alternating path in either parity. (See Figure 2.) Such an anomaly can occur only if G contains an alternating path p from a vertex s to an even vertex u and an edge from u to another even vertex w on p. The odd-length cycle formed by $\{u, w\}$ and the part of p from w to u is called a ***blossom***. In Figure 2, $\{c, d, e, f, g\}$ form a blossom.

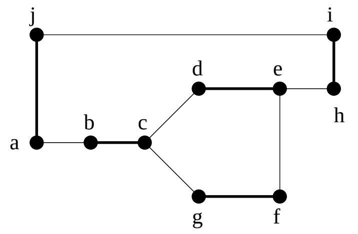

Figure 2: The problem with nonbipartite graphs. Suppose that we search whether $\{a, b\}$ belongs to an alternating cycle. $[a, b, c, d, e]$ marks e even, whereas $[a, b, c, g, f, e]$ marks e odd and h cannot be reached.

Edmonds proposed an algorithm that is able to deal with this difficulty. Since any vertex in a blossom can be reached from an alternating path from the base in either parity, it should be possible to traverse any free edge that has a vertex in the blossom if we do not want to miss an alternating path. As noted by Edmonds, this can be easily obtained by marking all the vertices of a blossom as even.

Algorithm 2 proposes an adaptation to our purpose of Edmonds's algorithm. Since Edmonds's algorithm cannot miss any alternating paths the arc consistency algorithm that we have proposed is exact.

Algorithm 2 traverses each edge at most twice, because the arc (u, w) is introduced in the list of arcs to study when u is marked even and a node is marked even only once. Moreover, when an arc is traversed it is removed from $ArcsToStudy$. Thus, the complexity of this algorithm depends on the functions belongToDifferentBlossom(u, v) and computeNewBlossom(u, v).

The function belongToDifferentBlossom(u, v) determine whether u and v belongs to different blossoms. If this function is true, then a new blossom is detected. The function computeNewBlossom(u, v) determines the nodes involved in the new blossom and updates internal data structures needed by the first function. Tarjan has proposed an efficient and beautiful implementation of them based on a union-find structure. We will not present it here because it is fully detailed in [Tarjan, 1983] p 121–122. This particular implementation leads to an algorithm in $O(m)$. Thus, we will consider that the complexity of Algorithm 2 is $O(m)$.

Therefore, the complexity of the arc consistency algorithm is $O(nm)$ because there are $2n$ calls to Algorithm 2. However, this algorithm is not incremental. In fact, each time arc consistency is achieved, it will be necessary to call a procedure $2n$ times in $O(m)$. For certain problems, this complexity can prevent this algorithm from being systematically used during the search for solutions. Thus, in the next section, we propose a filtering algorithm that does not necessarily ensure arc consistency, but it has a complexity that allows its systematic use during the search for solutions.

```
SEARCHFOREVENALTERNATINGCYCLE(x, CGV(C), M)
    // each edge {v, w} is also represented by
    // two arcs (v, w) and (w, v)
    mark all vertices unreached
    mark all edges not traversed and not valid
    remove the matching edge {x, mate(x)} from CGV(C)
    ArcsToStudy ← ∅
    mark mate(x) even and add all arcs (mate(x), w) in
    ArcsToStudy
    while ArcsToStudy ≠ ∅ do
        pick (u, v) in ArcsToStudy and remove it (u is even)
        mark {u, v} traversed
        if v = x then mark {u, v} valid
        else
            if v is unreached then
                mark v odd, mate(v) even and add all arcs
                (mate(v), w) in ArcsToStudy
            if v is even and belongToDifferentBlossom(u, v)
            then
                // a new blossom is discovered
                blossomNodes ← computeNewBlossom(u, v)
                for each odd node o ∈ blossomNodes do
                    mark o even
                    add all arcs (o, w) in ArcsToStudy
```

Algorithm 2: A modification of the blossom-shrinking algorithm of Edmonds applied to search for even alternating cycles containing a vertex x.

3.2 Another filtering algorithm

Property 1 *Let $\{u, mate(u)\}$ be a matching edge. If $\{u, mate(u)\}$ is traversed by Algorithm 2, then all edges that belong to an even alternating cycle containing $\{u, mate(u)\}$ are also traversed by the algorithm.*

This property holds because Edmonds's algorithm cannot miss any alternating paths.

Proposition 3 *Let M be matching that covers $X(C)$ in $CGV(C)$. Then any free edge $\{u, v\}$ such that at least one of its ends is reached by Algorithm 2 and $\{u, v\}$ is not traversed by Algorithm 2 cannot belong to a maximum matching.*

proof: The ends of $\{u, v\}$ cannot be even; otherwise, this edge would have been traversed by Algorithm 2. Consider that v has been reached, then v is marked odd. The matching covers all the vertices and v is odd, thus the matching edge $\{mate(v), v\}$ has been traversed. v belongs to only one matching edge, so every alternating cycle containing $\{u, v\}$ contains also $\{mate(v), v\}$. Furthermore, by Property 1, all the edges that belong to an even alternating cycle containing $\{v, mate(v)\}$ are traversed by the algorithm. Hence, if $\{u, v\}$ is not traversed by the algorithm then $\{u, v\}$ does not belong to any maximum matching.

From this proposition, we propose a filtering algorithm. We choose any vertex x; then we apply Algorithm 2 to it. Each free edge $\{x, u\}$ which is not marked valid is removed. Then, each edge which satisfies Proposition 3 is also removed. If at least one edge is removed

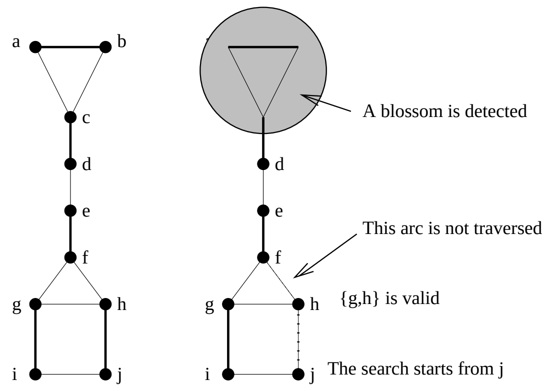

Figure 3: An example of the filtering algorithm for a symmetric alldiff constraint. The bold edges represent the matching edges. The edge $\{f, h\}$ is not traversed, so $\sigma(f)$ is removed from $D(h)$, and $\sigma(h)$ is removed from $D(f)$.

by the previous procedures, then we choose any other vertex that has not already been chosen, and we repeat the previous operation. If no edge is removed, we stop the algorithm. (See Figure 3.) If no deletion occurs, the complexity of the algorithm is $O(m)$, and for each deletion, the complexity of this algorithm is also $O(m)$. The advantage of this approach is that the complexity can be amortized for each deletion.

FILTERINGALGORITHM($CGV(C), M$)
  $S \leftarrow X(C)$; $credit \leftarrow 1$
  do
    pick a vertex x in S and remove it
    $credit \leftarrow credit - 1$
    searchForEvenAlternatingCycle($x, CGV(C), M$)
    for *each edge* $\{u, x\}$ *not marked valid* **do**
      remove $\{u, x\}$ from G
      remove $\sigma(u)$ from $D(x)$ and $\sigma(x)$ from $D(u)$
      $credit \leftarrow credit + 1$
    for *each not traversed free edge* $\{u, v\}$ *s.t.* u *or* v *has been reached* **do**
      remove $\{u, v\}$ from G
      remove $\sigma(u)$ from $D(v)$ and $\sigma(v)$ from $D(u)$
      $credit \leftarrow credit + 1$
  while $credit > 0$ **and** $S \neq \varnothing$

Algorithm 3: A filtering algorithm.

However, we can obtain a better amortization. Suppose that during one pass of the previous algorithm 10 edges are removed. Then, if the next 10 passes delete no edges, the amortized complexity will remain $O(m)$ per deletion. Algorithm 3 is a possible implementation of this idea.

This algorithm does not ensure arc consistency because Algorithm 2 traverses some edges that do not belong to an even alternating path.

In practice, this algorithm can also be improved by using some heuristics. It is important to take care about the possible creation of new connected components when some edges are removed. The previous algorithm can be independently applied to each connected component of the graph. On other hand, note that if $CGV(C)$ contains a connected component with an odd number of nodes, then there is no solution. This observation means that if there is a 2-connected component with an odd number of nodes and containing exactly one cutpoint of the graph[4], then this cutpoint cannot be matched with another node of the component. Similarly, if there is a 2-connected component with an even number of nodes and containing exactly one cutpoint of the graph, then this cutpoint cannot be matched with a node that does not belong to the 2-connected component. Such components can be identified easily in $O(m)$. Moreover, if a node in $CGV(C)$ has only two neighbors, then these two neighbors cannot be matched together. Furthermore, it is also interesting to use the classical alldiff constraint and arc consistency for this constraint.

4 Conclusion

In this paper we have presented the symmetric alldiff constraint. This constraint is present in many real-life applications. We have shown how arc consistency for this constraint can be achieved in $O(nm)$. We have also proposed a filtering algorithm that does not ensure arc consistency but has a complexity that can be used in practice because it can be amortized for each deletion ($O(m)$ per deletion).

References

[Berge, 1970] C. Berge. *Graphe et Hypergraphes.* Dunod, Paris, 1970.

[Bessière and Régin, 1997] C. Bessière and J-C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, Nagoya, 1997.

[Edmonds, 1965] J. Edmonds. Path, trees, and flowers. *Can. J. Math.*, 17:449–467, 1965.

[Micali and Vazirani, 1980] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings 21st FOCS*, pages 17–27, 1980.

[Régin, 1994] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI-94*, pages 362–367, Seattle, Washington, 1994.

[Simonis, 1996] H. Simonis. Problem classification scheme for finite domain constraint solving. In *CP96, Workshop on Constraint Programming Applications: An Inventory and Taxonomy*, pages 1–26, Cambridge, MA, USA, 1996.

[Tarjan, 1983] R.E. Tarjan. *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics, 1983.

[4] A cutpoint is a node whose deletion increases the number of connected components of the graph.