Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

Unité Mixte CNRS - Université Montpellier II C 09928

RAPPORT DE RECHERCHE

A filtering algorithm for constraints of difference in CSPs

A paraitre dans les actes de AAAI-94

Jean-Charles RÉGIN

Décembre 1993 R.R.LIRMM 93-068

161, Rue Ada - 34392 Montpellier Cedex 5 - France
Tél: (33) 67 41 85 85 - Fax: (33) 67 41 85 00

A filtering algorithm for constraints of difference in CSPs *

Jean-Charles RÉGIN

GDR 1093 CNRS
LIRMM UMR 9928 Université Montpellier II / CNRS
161, rue Ada – 34392 Montpellier Cédex 5 – France
e-mail : regin@lirmm.fr

Résumé

Many real-life Constraint Satisfaction Problems (CSPs) involve some constraints similar to the alldifferent constraints. These constraints are called constraints of difference. They are defined on a subset of variables by a set of tuples for which the values occuring in the same tuple are all different. In this paper, a new filtering algorithm for these constraints is presented. It achieves the generalized arc-consistency condition for these non-binary constraints. It is based on matching theory and its complexity is low. In fact, for a constraint defined on a subset of p variables having domains of cardinality at most d, its space complexity is $O(pd)$ and its time complexity is $O(p^2d^2)$. This filtering algorithm has been successfully used in the system RESYN [Vismara *et al.*, 1992], to solve the subgraph isomorphism problem.

1 Introduction

The constraint satisfaction problems (CSPs) form a simple formal frame to represent and solve some problems in artificial intelligence. The problem of the existence of solutions in a CSP is NP-complete. Therefore, some methods have been developed to simplify the CSP before or during the search for solutions. The consistency techniques are the most frequently used. Several algorithms achieving arc-consistency have been proposed for binary CSPs [Mackworth, 1977; Mohr and Henderson, 1986; Bessière and Cordier, 1993; Bessière, 1994] and for n-ary CSPs [Mohr and Masini, 1988a]. Only limited works have been carried out on the semantics of constraints : [Mohr and Masini, 1988b] have described

*This work was supported by SANOFI-CHIMIE

an improvement of the algorithm AC-4 for special constraints introduced by a vision problem, [Van Hentenryck *et al.*, 1992] have studied monotonic and functional binary constraints. In this work, we are interested in a special case of n-ary constraints : the constraints of difference, for which we propose a filtering algorithm.

A constraint is called *constraint of difference* if it is defined on a subset of variables by a set of tuples for which the values occuring in the same tuple are all different. They are present in many real-life problems.

These constraints can be represented as n-ary constraints and filtered by the generalized arc-consistency algorithm GAC4 [Mohr and Masini, 1988a]. This filtering efficiently reduces the domains but its complexity can be expensive. In fact, it depends on the length and the number of all admissible tuples. Let us consider a constraint of difference defined on p variables, which take their values in a set of cardinality d. Thus, the number of admissible tuples corresponds to the number of permutations of p elements selected from d elements without repetition : ${}^{d}P_{p} = \frac{d!}{(d-p)!}$. Therefore some constraint resolution systems, like CHIP [Van Hentenryck, 1989], represent these n-ary constraints by sets of binary constraints. In this case, a binary constraint of difference is built for each pair of variables belonging to the same constraint of difference. But the pruning performance of arc-consistency, for these constraints is poor. In fact, for a binary alldifferent constraint between two variables i and j, arc-consistency removes a value from domain of i only when the domain of j is reduced to a single value. Let us suppose we have a CSP with 3 variables x_1, x_2, x_3 and one

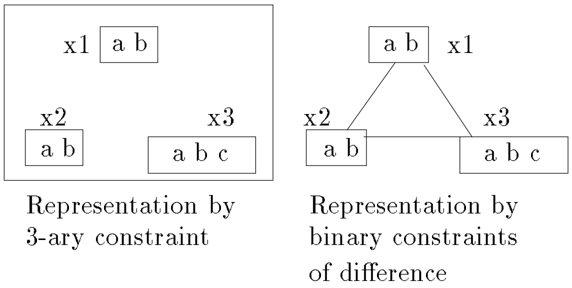

Figure 1:

constraint of difference between these variables (see figure 1). The domains of variables are $D_1 = \{a, b\}$, $D_2 = \{a, b\}$ and $D_3 = \{a, b, c\}$. The GAC4 filtering with the constraint of difference represented by a 3-ary constraint, removes the values a and b from the domain of x_3, while arc-consistency with the constraint of difference represented by binary constraints of difference, does not delete any value.

In this paper we present an efficient way of implementing the generalized arc-consistency condition for the constraints of difference, in order to benefit from its pruning performances. Its space complexity is in $O(pd)$ and its time complexity is in $O(p^2d^2)$.

The rest of the paper is organized as follows. Section 2 gives some preliminaries on constraint satisfaction problems and matching, and proposes a restricted definition of arc-consistency, which concerns only the constraints of difference : the diff-arc-consistency. Section 3 presents a new condition to ensure the diff-arc-consistency in CSPs having constraints of difference. In section 4 we propose an efficient implementation to achieve this condition and analyse its complexity. In section 5, we show its performance and its interest with an example. A conclusion is given in section 6.

2 Preliminaries

A finite CSP (Constraint Satisfaction Problem) $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is defined as a set of n *variables* $X = \{x_1, ..., x_n\}$, a set of finite *domains* $\mathcal{D} = \{D_1, ..., D_n\}$ where D_i is the set of possible *values* for variable i and a set of *constraints* between variables $\mathcal{C} = \{C_1, C_2, ..., C_m\}$. A constraint C_i is defined on a set of variables $(x_{i_1}, ..., x_{i_j})$ by a subset of the cartesian product $D_{i_1} \times ... \times D_{i_j}$. A solution is an assignment of value to all variables which satisfies all the constraints. We will denote by :

- $D(X')$ the union of domains of variables of $X' \subseteq X$ ($i.e\ D(X') = \cup_{i \in X'} D_i$).
- X_C the set of variables on which a constraint C is defined.
- p the arity of a constraint C : $p = |X_C|$.
- d the maximal cardinality of domains.

A value a_i in the domain of a variable x_i is consistent with a given n-ary constraint if there exists values for all the other variables in the constraint such that these values with a_i *together simultaneously* satisfy the constraint. More generally, arc-consistency for n-ary CSPs or the generalized arc-consistency is defined as follows [Mohr and Masini, 1988a]:

Definition 1 *A CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ *is* **arc-consistent** *iff :* $\forall x_i \in X, \forall a_i \in D_i, \forall C \in \mathcal{C}$ *constraining* $x_i, \forall x_j, ..., x_k \in X_C, \exists a_j, ..., a_k$ *such that* $C(a_j, ..., a_i, ..., a_k)$ *holds.*

Definition 2 *Given a CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$*, a constraint C is called* **constraint of difference** *if it is defined on a subset of variables* $X_C = \{x_{i_1}, ..., x_{i_k}\}$ *by a set of tuples, denoted by $tuples(C)$ such that :* $tuples(C) = D_{i_1} \times ... \times D_{i_k} \setminus \{(d_1, ..., d_k) \in D_{i_1} \times ... \times D_{i_k}\ s.t. \exists\, u, v \,|\, d_u = d_v\}$

From the previous definition, we propose a special arc-consistency which concerns only the constraints of difference :

Definition 3 *A CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ *is* **diff-arc-consistent** *iff all of its constraints of difference are arc-consistent.*

Definition 4 *Given a constraint of difference* C*, the bipartite graph* $GV(C) = (X_C, D(X_C), E)$ *where* $(x_i, a) \in E$ *iff* $a \in D_i$ *is called* **value graph** *of* C*.*

Figure 2 gives an example of a constraint of difference and its value graph.

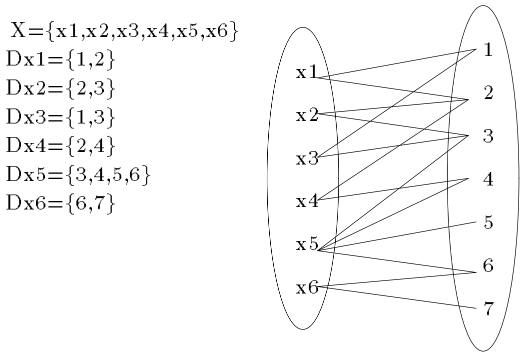

Figure 2: A constraint of difference defined on a set X and its value graph

Definition 5 *A subset of edges in a graph* G *is called* **matching** *if no two edges have a vertex in common. A matching of maximum cardinality is called a* **maximum matching***. A* **matching** M **covers a set** X *if every vertex in* X *is an endpoint of an edge in* M*.*

Note that a matching which covers X in a bipartite graph $G = (X, Y, E)$ is a maximum matching.

From the definition of a matching and the value graph we present, in the next section, a new necessary condition to ensure the diff-arc-consistency in CSPs having constraints of difference.

3 A new condition for CSPs having constraints of difference

The following theorem establishes a link between the diff-arc-consistency and the matching notion in the value graph of the constraints of difference.

Theorem 1 *Given a CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$*.* $\mathcal{P}$ *is diff-arc-consistent iff for each constraint of difference* C *of* $\mathcal{C}$ *every edge in* $GV(C)$ *belongs to a matching which covers* X_C *in* $GV(C)$*.*

proof
$\Rightarrow$: Let us consider a constraint of difference C and $GV(C)$ its value graph. From each admissible tuple of C, a set of pairs can be built. A pair consists of a variable and its assigned value in the tuple. The set of pairs contains a pair for each variable. This set corresponds to a set of edges, denoted by A in $GV(C)$. Since $\mathcal{P}$ is diff-arc-consistent, the values in each tuple are all different. Thus, two edges of A cannot have a vertex in common and A is a matching which covers X_C. Moreover, each value of each variable in the constraint belongs to at least one tuple. So, each edge of $GV(C)$ belongs to a matching which covers X_C.
$\Leftarrow$: Let us consider a variable x_i and a value a of its domain. For each constraint of difference C, the pair (x_i, a) belongs to a matching which covers X_C in $GV(C)$. Since in a matching no two edges have a vertex in common, there exists values for all the other variables in the constraint such that these values together simultaneously satisfy the constraint. So $\mathcal{P}$ is diff-arc-consistent. $\Box$

The use of matching theory is interesting because [Hopcroft and Karp, 1973] have shown how to compute a matching which covers X in a bipartite graph $G = (X, Y, E)$, with m edges, [1] in time $O(\sqrt{|X|}m)$.

This theorem gives us an efficient way to represent the constraint of difference in a CSP. In fact, a constraint of difference can be represent only by its value graph, with a space complexity in $O(pd)$. It also allows us to define a basic algorithm (algorithm 1) to filter the domains of variables of the set on which one constraint of difference is defined. This algorithm builds the value graph of the constraint of difference and computes a matching which covers X_C in order to delete every edge which belongs to no matching covering X_C. Figure 3 gives an application of this filtering.

Algorithm 1: DIFF-INITIALIZATION(C)
% returns false if there is no solution, otherwise true
% the function COMPUTEMAXIMUMMATCHING(G) computes a maximum matching in the graph G
begin
1 Build $G = (X_C, D(X_C), E)$
2 $M(G) \leftarrow$ COMPUTEMAXIMUMMATCHING(G)
if $|M(G)| < |X_C|$ **then return** *false*
3 REMOVEEDGESFROMG(G,$M(G)$)
return *true*
end

The complexity of step 1 is $O(d|X_C| + |X_C| + |D(X_C)|)$. Step 2 costs $O(d|X_C|\sqrt{|X_C|})$. And we now show that it is possible to compute step 3 in linear time. So the complexity for one constraint of difference will be $O(d|X_C|\sqrt{|X_C|})$.

[1] [Alt *et al.*, 1991] give an implementation of Hopcroft and Karp's algorithm which runs in time $O(|X|^{1.5}\sqrt{m \log |X|})$. For dense graph this is an improvement by a factor of $\sqrt{\log |X|}$.

4 Deletion of every edge which belongs to no matching which covers X

In order to simplify the notation, we consider a bipartite graph $G = (X, Y, E)$ rather than the bipartite graph $G = (X_C, D(X_C), E)$, and a matching M which covers X in G. In order to understand how we can delete every edge which

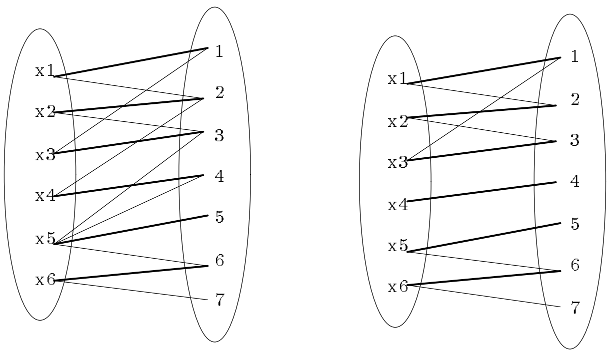

Figure 3: A value graph before and after the filtering.

belongs to no matching, we present a few definitions about matching theory. For more information the reader can consult [Berge, 1970] or [Lovász and Plummer, 1986].

Definition 6 *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. 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. An edge which belongs to every maximum matching is* **vital**.

Figure 3 gives an example of a matching which covers X in a bipartite graph. The bold edges are the matching edges. Vertex 7 is free. The path $(7, x6, 6, x5, 5)$ is an alternating path which begins at a free vertex. The cycle $(1, x3, 3, x2, 2, x1, 1)$ is an alternating cycle. The edge $(x4, 4)$ is vital.

Property 1 (Berge 1970) *An edge belongs to some of but not all maximum matchings, iff, for an arbitrary maximum matching M, it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.*

From this property we can find for an arbitrary matching M which covers X, every edge which belongs to no matching covering X. There are the edges which belong to neither M (there are not vital), nor an even alternating path which begins at a free vertex, nor an even alternating cycle.

Proposition 1 *Given a bipartite graph $G = (X, Y, E)$ with a matching M which covers X and the graph $G_O = (X, Y, Succ)$, obtained from G by orienting edges with the function :*

$\forall x \in X : Succ(x) = \{y \in Y \,/\, (x, y) \in M\}$
$\forall y \in Y : Succ(y) = \{x \in X \,/\, (x, y) \in E - M\}$

we have the two following properties :

1) *Every directed cycle of G_O corresponds to an even alternating cycle of G, and conversely.*

2) *Every directed simple path of G_O, which begins at a free vertex corresponds to an even alternating path of G which begins at a free vertex, and conversely.*

proof
If we ignore the parity, it is obvious that the proposition is true. In the first case, since G is bipartite it does not have any odd cycle. In the second case, we must show every directed simple path of G_O which begins at a free vertex to corresponds to an even alternating path of G which begins at a free vertex. M is a matching which covers X, so there is no free vertex in X. Since G is bipartite and since every path begins at a free vertex, in Y, every odd directed simple path ends with a vertex in X. From this vertex, we can always find a vertex in Y which does not belong to the path, because every vertex in X has one successor and because a vertex in Y has one predecessor. Therefore from an odd directed simple path we can always build an even directed simple path.□

From this proposition we produce a linear algorithm (algorithm 2), that deletes every edge which does not belong to any matching which covers X.

Step 2 finds all edges belonging to the directed simple paths of G_O, which begins at a free vertex. Moreover, it finds some edges belonging to the directed cycles of G_O. Step 3 computes the strongly connected component of G_O, because an edge joining two vertices in the same strongly connected component belongs to a directed cycle and conversely. These edges belong to an even alternating cycle of G (cf point 1 of proposition 1). After this step the set A of all edges belonging to some but not all matchings covering X are known. The set RE of edges to remove from G is: $RE = E - (A \cup M)$. This is done by step 4. The algorithm complexity is the same as the search for strongly connected components[Tarjan, 1972] , *i.e* $O(m+n)$ for a graph with m edges and n vertices.

We have shown how for *one* constraint of difference C every edge which belongs to no matching which covers X_C can be deleted. But a variable can be constrained by several constraints and it is necessary to propagate the deletions. In fact, let us consider x_i a variable of X_C, x_i can be constrained by several constraints. Thus, a value of D_i can be deleted for reasons independent from C. This deletion involves the removal of one edge from $GV(C)$. So, it is necessary to study the consequences of this modification of the $GV(C)$ structure.

Algorithm 2: REMOVEEDGESFROMG(G,$M(G)$)
% RE is the set of edges removed from G.
% $M(G)$ is a matching of G which covers X
% The function returns RE
begin
1 Mark all directed edges in G_O as "unused".
Set RE to $\emptyset$.
2 Perform a breadth-first search starting from
free vertices, and mark all traversed edges as "used".
3 Compute the strongly connected components of G_O.
Mark as "used" any directed edge that joins two
vertices in the same strongly connected component.
4 **for** each directed edge de marked as "unused" **do**
    set e to the corresponding edge of de
    if $e \in M(G)$ **then** mark e as "vital"
    else
        $RE \leftarrow RE \cup \{e\}$
        remove e from G
return RE
end

5 Propagation of deletions

The deletion of values for one constraint of difference can involve some modifications for the other constraints. And for the other constraints of difference we can do better than repeat the first algorithm by using the fact that before the deletion, a matching which covers X is known.

The propagation algorithm we propose has two sets as parameters. The first one represents the set of edges to remove from the bipartite graph, and the second the set of edges that will be deleted by the filtering. The algorithm needs a function, denoted by MATCHINGCOVERINGX(G, M_1, M_2), which computes a matching M_2, which covers X, from a matching M_1 which is not maximum. It returns true if M_2 exists and false otherwise. The new filtering is represented by algorithm 3.

It is divided into three parts. First, it removes edges from the bipartite graph. Second, it eventually computes a new matching which covers X_C. Third, it deletes the edges which does not belongs to any matching covering X_C. The algorithm returns false if ER contains a vital edge or if there does not exist a matching which covers X_C.

Now, let us compute its complexity. Let m be the number of edges of G, and n be the number of vertices. Let us suppose that we must remove k edges from G ($|ER| = k$). The complexity of 1 is in $O(k)$. Step 2 involves, in the worst case, the computation of a matching covering X_C from a matching of cardinality $|M - k|$. This computation has cost $O(\sqrt{k}\, m)$ (see theorem 3 of [Hopcroft and Karp, 1973]). The complexity of step 3 is in $O(m)$.

```
Algorithm 3: DIFF-PROPAGATION(G,M(G),ER,RE)
% the function returns false if there is no solution
% G is a value graph
% M(G) is a matching which covers X_C
% ER is the set of edges to remove from G
% RE is the set of edges that will be deleted by the filtering
begin
    computeMatching ← false
1   for each e ∈ ER do
        if e ∈ M(G) then
            M(G) ← M(G) − {e}
            if e is marked as "vital" then return false
            else computeMatching ← true
        remove e from G
2   if computeMatching then
        if ¬ MATCHINGCOVERINGX(G,M(G),M') then
            return false
        else
            M(G) ← M'
3   RE ← REMOVEEDGESFROMG(G,M(G))
    return true
end
```

In the worst case, the edges of G can be deleted one by one. Then the previous function will be called m times. So the global complexity is in $O(m^2)$. If $p = |X_C|$ and d is the maximum cardinality of domains of variables of X_C, then the complexity is in $O(p^2d^2)$ for one constraint of difference.

6 An example : the zebra problem

1. There are five houses, each of a different color and inhabited by men of different nationalities, with differents pets, drinks and cigarettes.
2. The Englishman lives in the red house.
3. The Spaniard owns a dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old-Gold smoker owns snails.
8. Kools are being smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house on the left.
11. The Chesterfield smoker lives next to the fox owner.
12. Kools are smoked in the house next to the house where the horse is kept.

13. The Lucky-Strike smoker drinks orange juice.
14. The Japanese smokes Parliament.
15. The Norwegian lives next to the blue house.
The *query* is : Who drinks water and who owns the zebra ?

This problem can be represented as a constraint network involving 25 variables, one for each of the five colors, drinks, nationalities, cigarettes and pets :

C_1 red	B_1 coffee	N_1 Englishman	T_1 Old-Gold	A_1 dog
C_2 green	B_2 tea	N_2 Spaniard	T_2 Chesterfield	A_2 snails
C_3 ivory	B_3 milk	N_3 Ukranian	T_3 Kools	A_3 fox
C_4 yellow	B_4 orange	N_4 Norwegian	T_4 Lucky-Strike	A_4 horse
C_5 blue	B_5 water	N_5 Japanese	T_5 Parliament	A_5 zebra

Each of the variables has domain values $\{1, 2, 3, 4, 5\}$, each number corresponding to a house position (e.g. assigning the value 2 to the variable *horse* means that the horse owner lives in the second house) [Dechter, 1990]. The assertions 2 to 15 are translated into unary and binary constraints. In addition, there are three ways of representing the first assertion which means that the variables in the same cluster must take different values :

1. A binary constraint is built between any pair of variables of the same cluster ensuring that they are not assigned the same value. In this case we have a binary CSP.

2. Five 5-ary constraints of difference are built (one for each of the clusters). And the CSP is not binary.

3. The five 5-ary constraints of difference are represented by their value graphs. The space complexity of one constraint is in $O(pd)$.

The first representation is generally used to solve the problem [Dechter, 1990; Bessière and Cordier, 1993]. From these three representations we can study the different results obtained from arc-consistency. They are given in figures 4 and 5. The constraints corresponding to the assertions 2 to 15 are represented in extension. The constraints of difference among the variables of each cluster are omitted for clarity.

For the first representation, the result of the filtering by arc-consistency is given in figure 4.

For the second representation, the filtering algorithm employed is the generalized arc-consistency. Figure 5 shows the new results. It has pruned more values that the previous one.

For the third representation, the filtering algorithm employed is arc-consistency for the binary constraints combined with the new filtering for the constraints of difference. The obtained results are the same as with the second method.

2 (=)	
N_1	C_1
3	3
4	4
5	5

3 (=)	
N_2	A_1
2	2
3	3
4	4
5	5

4 (=)	
B_1	C_2
4	4
5	5

5 (=)	
N_3	B_2
2	2
4	4
5	5

6 (-1)	
C_2	C_3
4	3
5	4

7 (=)	
T_1	A_2
1	1
2	2
3	3
4	4
5	5

8 (=)	
T_3	C_4
1	1
3	3
4	4
5	5

9
B_3
3

10
N_4
1

11 ($\pm$1)	
T_2	A_3
1	2
2	1
2	3
3	2
3	4
4	3
4	5
5	4

12 ($\pm$1)	
A_4	T_3
2	1
2	3
3	4
4	3
4	5
5	4

13 (=)	
T_4	B_4
1	1
2	2
4	4
5	5

14 (=)	
N_5	T_5
2	2
3	3
4	4
5	5

15 ($\pm$1)	
N_4	C_5
1	2

A_5
1
2
3
4
5

B_5
1
2
4
5

Figure 4:

Let us denote by a the number of binary constraints corresponding to the assertions 2 to 15, p the size of a cluster, c the number of clusters, d the number of values in a domain and $O(ed^2)$ the complexity for arc-consistency[2] in binary CSPs. Let us compute the complexity for the three methods :

1. For the first representation, the number of binary constraints of difference added is in $O(cp^2)$. So, the filtering complexity is $O((a + cp^2)d^2)$.

2. In the second case, we can consider that the complexity is the sum of the lengths of all admissible tuples for the five 5-ary constraints. It is in $O(\frac{d!}{(d-p)!}p)$.

3. For the third method arc-consistency is in $O(ad^2)$ and the filtering for the constraints of difference is in $O(cp^2d^2)$. The total complexity is in $O(ad^2) + O(cp^2d^2)$. It is equivalent to the first one.

The second filtering eliminates more values than the first one. But its complexity is higher. The representation and the algorithm proposed in this paper give pruning results equivalent to the second approach with the same complexity as the first one. So we can conclude that the new filtering is good for problems looking like the zebra problem.

[2][Mohr and Masini, 1988b] reduce this complexity to $O(ed)$ for the binary alldifferent constraints

2 (=)	
N_1	C_1
3	3
4	4
5	5

3 (=)	
N_2	A_1
3	3
4	4
5	5

4 (=)	
B_1	C_2
4	4
5	5

5 (=)	
N_3	B_2
2	2
4	4
5	5

6 (-1)	
C_2	C_3
4	3
5	4

7 (=)	
T_1	A_2
3	3
4	4
5	5

8 (=)	
T_3	C_4
1	1

9
B_3
3

10
N_4
1

11 (±1)	
T_2	A_3
2	1
2	3
3	4
4	3
4	5
5	4

12 (±1)	
A_4	T_3
2	1

13 (=)	
T_4	B_4
2	2
4	4
5	5

14 (=)	
N_5	T_5
2	2
3	3
4	4
5	5

15 (±1)	
N_4	C_5
1	2

A_5
1
3
4
5

B_5
1

Figure 5:

7 Conclusion

In this paper we have presented a filtering algorithm for constraints of difference in CSPs. This algorithm can be viewed as an efficient way of implementing the generalized arc-consistency condition for a special type of constraint : the constraints of difference. It allows us to benefit from the pruning performance of the previous condition with a low complexity. In fact, its space complexity is in $O(pd)$ and its time complexity is in $O(p^2d^2)$ for one constraint defined on a subset of p variables having domains of cardinality at most d. It has been shown to be very efficient for the zebra problem. And it has been successfully used to solve the subgraph isomorphism problem in the system RESYN [Vismara *et al.*, 1992], a computer-aided design of complex organic synthesis plan.

8 Acknowledgments

We would like to thank particularly Christian Bessière and also Marie-Catherine Vilarem, Tibor Kökény for their comments which helped improve this paper.

Bibliographie

[Alt *et al.*, 1991] H. Alt, N. Blum, K. Melhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1,5}\sqrt{m/\log n})$. *Information Processing Letters*, 37:237–240, 1991.

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

[Bessière and Cordier, 1993] C. Bessière and M.O. Cordier. Arc-consistency and arc-consistency again. In *Proceedings AAAI*, pages 108–113, Washington, DC, 1993.

[Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, 1994.

[Dechter, 1990] R. Dechter. Enhencement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[Hopcroft and Karp, 1973] J.E. Hopcroft and R.M. Karp. $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 2:225–231, 1973.

[Lovász and Plummer, 1986] L. Lovász and M.D. Plummer. *Matching Theory*. North Holland mathematics studies 121, 1986.

[Mackworth, 1977] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mohr and Henderson, 1986] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Mohr and Masini, 1988a] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI*, pages 651–656, 1988.

[Mohr and Masini, 1988b] R. Mohr and G. Masini. Running efficiently arc consistency. *Syntactic and Structural Pattern Recognition*, F45:217–231, 1988.

[Tarjan, 1972] R.E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.

[Van Hentenryck *et al.*, 1992] P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Van Hentenryck, 1989] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. M.I.T. Press, 1989.

[Vismara *et al.*, 1992] P. Vismara, J-C. Régin, J. Quinqueton, M. Py, C. Laurenço, and L. Lapied. RESYN : Un système d'aide à la conception de plans de synthèse en chimie organique. In *Proceedings 12th International Conference Avignon'92*, volume 1, pages 305–318, Avignon, 1992. EC2.