ELSEVIER Artificial Intelligence 93 (1997) 321-335

Artificial Intelligence

Research Note

Constraint satisfaction problem with bilevel constraint: application to interpretation of over-segmented images

A. Deruyver[a,*], Y. Hodé[b]

[a] I.U.T. Strasbourg Sud, Département d'Informatique, 72 route du Rhin, 67400 Illkirch, France
[b] F.O.R.E.N.A.P., Centre Hospitalier de Rouffach, 68250 Rouffach, France

Received March 1996; revised April 1997

Abstract

In classical finite-domain constraint satisfaction problems, the assumption made is that only one value is associated with only one variable. For example, in pattern recognition one variable is associated with only one segmented region. However, in practice, regions are often over-segmented which results in failure of any one to one mapping. This paper proposes a definition of finite-domain constraint satisfaction problems with bilevel constraints in order to take into account a many to one relation between the values and the variables. The additional level of constraint concerns the data assigned to the same complex variable. Then, we give a definition of the arc-consistency problem for bilevel constraint satisfaction checking. A new algorithm for arc consistency to deal with these problems is presented as well. This extension of the arc-consistency algorithm retains its good properties and has a time complexity in $O(en^3d^2)$ in the worst case. This algorithm was tested on medical images. These tests demonstrate its reliability in correctly identifying the segmented regions even when the image is over-segmented. © 1997 Elsevier Science B.V.

Keywords: Semantic graph; Arc consistency; Constraint satisfaction; Image interpretation

1. Introduction

Pattern recognition can be regarded as a matching problem between an abstract description of what is to be recognized and the concrete description of what is observed. Semantic nets are a suitable way to describe many complex entities [1]. This kind of prob-

* Corresponding author. E-mail:deruy@iutsud.u-strasbg.fr.

PII S0004-3702(97)00022-2

lem can be seen as a Finite-Domain Constraint Satisfaction Problem (*FDCSP*) which provides a theoretical framework within which the problem can be solved [5,6,15–18]. The *FDCSP* is defined by two finite sets: a set of variables and a set of constraint relations between these variables. A solution to an *FDCSP* is an assignment of values, taken from finite domains, to variables satisfying all constraints.

In the case of pattern recognition with a semantic net, the semantic links can represent the constraints and the variables are the labels of the different parts of the image.

However, to our knowledge, this approach has seldom been applied to pattern recognition. One reason could be the inconsistencies between the classical definition of *FDCSP* and some particular aspects of image analysis. Indeed, the labeling of parts of an image is rarely a one to one process because the segmentation step often yields over-segmented regions. This is even more true in three-dimensional multi-slice images where a structure can appear on several slices. Each slice where the structure appears introduces a new segmented region. In a multi-slice image a single structure is composed of different regions which by definition means that the structure is over-segmented. To label this kind of data, we might think that it is enough to bring together regions in a unique three-dimensional object. Then, the idea is to find a partition of the set of regions according to an equivalence relation, each class corresponding to a three-dimensional object. In some cases, the transitive closure of the spatial relation "A overlaps B" can fit with this approach. With such a partition, a morphism can be defined to work directly with the equivalence classes instead of the individual regions. The relations between equivalence classes are inherited from the relations between their elements. Unfortunately it is not always so simple. The overlapping of regions from two consecutive slices does not guarantee that these regions belong to the same object. In most practical cases, it is impossible to make a prior grouping before constraint satisfaction checking. However, some properties can be found to decide if the grouping of some regions is possible or not. In spite of this uncertainty, it is worth taking advantage of these properties in the labeling process. But to deal with this uncertainty, we have to manage simultaneously two interdependent criteria: the satisfaction of local constraints and the satisfaction of compatibility to group data.

To adapt the framework of the *FDCSP* for such problems we propose to define the class of *FDCSP* for complex variables with bilevel constraints ($FDCSP_{BC}$), one level corresponding to inter-variable constraints and the other corresponding to the satisfaction of compatibility between data assigned to the same variable. In order to solve *FDCSP*, many approaches have tried to find a local evaluation of constraints [2,11–13,19,20]. Currently, the best-known levels of partial consistency are arc and path consistency. Several arc-consistency algorithms show interesting theoretical and practical optimality properties [2,10,13,14,19]. We propose a new definition of the arc-consistency (*AC*) problem fitted to $FDCSP_{BC}$ and we call it AC_{BC}. Then, we adapt a well-known algorithm called AC_4 [13] to this problem. This new algorithm called AC_{4BC} retains the good properties of time complexity.

This paper is organized as follows: Section 2 describes the notation used in this paper, gives basic definitions and studies the limits of classical arc-consistency problems. It gives the new definitions of $FDCSP_{BC}$ and AC_{BC} as well. Section 3 describes the AC_{4BC} algorithm and its properties. Section 4 describes an application of the AC_{4BC} algorithm to cerebral Nuclear Magnetic Resonance images. Section 5 states the conclusions of this work.

2. Preliminaries

2.1. Constraint satisfaction problem

We use the following conventions:

Variables are represented by the natural numbers $1, \dots, n$. Each variable i has an associated domain D_i.

All constraints are binary and relate two distinct variables. A constraint relating two variables i and j is denoted by C_{ij}.

$C_{ij}(v, w)$ is the Boolean value obtained when variables i and j are replaced by values v and w respectively. $\neg C_{ij}(v, w)$ denotes the negation of the Boolean value $C_{ij}(v, w)$.

Let $\mathcal{R}$ be the set of these constraining relations. We use D to denote the union of all domains and d the size of the largest domain.

A finite-domain constraint satisfaction problem consists of finding all sets of values $\{a_1, \dots, a_n\}$, $a_1 \times \dots \times a_n \in D_1 \times \dots \times D_n$, for $(1, \dots, n)$ satisfying all relations belonging to $\mathcal{R}$.

In this classical definition of *FDCSP*, one variable is associated with one value. This assumption cannot hold for some classes of problems where we need to associate a variable with a set of linked values. We call this new problem the Finite-Domain Constraint Satisfaction Problem with Bilevel Constraints ($FDCSP_{BC}$) and define it as follows:

Definition 1. Let *Cmpi* be a compatibility relation associated with i, such that $(a, b) \in$ *Cmpi* iff a and b are compatible. Clearly, *Cmpi* is reflexive and symmetric.

Let C_{ij} be constraint between i and j. A pair S_i, S_j such that $S_i \subset D_i$ and $S_j \subset D_j$ satisfies C_{ij}, written $S_i, S_j \models C_{ij}$, iff $\forall a_i \in S_i$, $\exists a'_i \in S_i$ and $a_j \in S_j$, such that $(a_i, a'_i) \in$ *Cmpi* and $(a'_i, a_j) \in C_{ij}$ and $\forall a_j \in S_j$, $\exists a'_j \in S_j$ and $a_i \in S_i$, such that $(a_j, a'_j) \in$ *Cmpj* and $(a_i, a'_j) \in C_{ij}$.

Sets $\{S_1, \dots, S_n\}$ satisfy $FDCSP_{BC}$ iff $\forall C_{ij}$ $S_i, S_j \models C_{ij}$.

We associate a graph G to a constraint satisfaction problem in the following way:

- G has a node i for each variable i,
- two directed arcs (i, j) and (j, i) are associated with each constraint C_{ij},
- $arc(G)$ is the set of arcs of G and e is the number of arcs in G,
- $node(G)$ is the set of nodes of G and n is the number of nodes in G.

2.2. Arc-consistency problem

The standard definitions of arc consistency are the following:

Definition 2. Let $(i, j) \in arc(G)$. Arc (i, j) is arc-consistent with respect to D_i and D_j iff $\forall v \in D_i$, $\exists w \in D_j$: $C_{ij}(v, w)$.

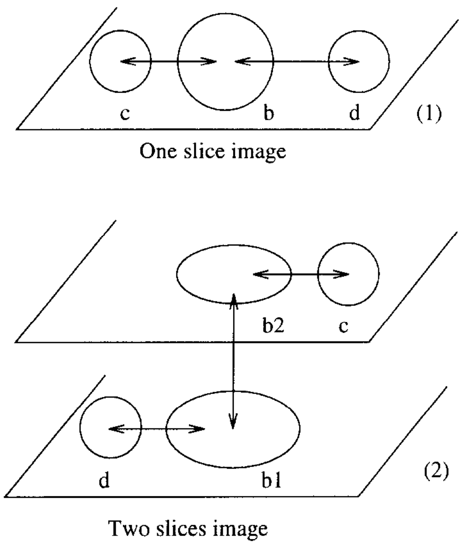

Fig. 1. b is a good candidate for a given node called "center" if there exists a region on the left of b and another one on the right of b. In 2 dimensions b satisfies the two relations (1), but if b is made up of two overlapped regions $b1$ and $b2$ (2), neither $b1$ nor $b2$ satisfies the relations. However if we bring together $b1$ and $b2$ in a unique object b, the relations are satisfied.

Definition 3. Let $P = D_1 \times \cdots \times D_n$. A graph G is arc-consistent with respect to P iff $\forall (i,j) \in arc(G)$: (i,j) is arc-consistent with respect to D_i and D_j.

The purpose of an arc-consistency algorithm is, given a graph G and a set P, to compute P', the largest arc-consistent domain for G in P.

However such an algorithm cannot classify a set of data in a node of the graph as we would like to do in over-segmented image interpretation. Indeed, let $b1$ and $b2$ be two over-segmented regions of the same object associated with the node i. Let c be the only region associated with a node j such that $C_{ij}(b1, c)$ and let d be the only region associated with a node k such that $C_{ik}(b2, d)$. Assuming that no region is in relation with $b1$ by the constraint C_{ik} and no region is in relation with $b2$ by constraint C_{ij}, the arc-consistency algorithm will remove $b1$ from node i because it does not satisfy C_{ik} and $b2$ from node i because it does not satisfy C_{ij} (cf. Fig. 1) instead of keeping both. Of course, if we already knew that $b1$ and $b2$ are parts of the same object, it would be easy to avoid the failure of the arc-consistency principle by making an appropriate data grouping. Unfortunately, it is very unusual to have this previous knowledge because our segmentation is a function of the noise of the image and cannot be predicted.

However, it is often possible to define some relation of compatibility specifying if two regions could belong to a same object. This relation will be denoted by *Cmpi* (cf. Definition 1). The following example illustrates such a *Cmpi* relation.

Example 4. Let R_{i1} be the transitive closure of the symmetrical relation "a overlaps b". Let R_{i2} be the relation "a is in the close neighbourhood of b", where the close

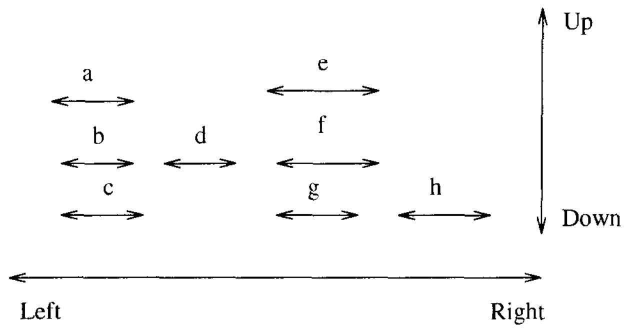

Fig. 2. *a, b, c, d, e, f, g, h* are regions having spatial relations between each other.

neighbourhood is defined by a maximum distance from one object to another in the horizontal plane. We define the relation of compatibility *Cmpi* for a node i as follows:

> For $a \in D_i$ and $b \in D_i$, $Cmpi(a,b)$ iff $R_{i1}(a,b)$ or $R_{i2}(a,b)$ or $\exists c \in D_i$ such that $R_{i1}(a,c)$ and $R_{i2}(c,b)$.

In the situation illustrated by Fig. 2, we have $Cmpi(a,d), Cmpi(d,f)$ and $\neg Cmpi(a,f)$. This relation is not transitive. Consequently *Cmpi* is not an equivalence relation. We can see that the object d is compatible with two different sets: $\{a,b,c,d\}$ and $\{e,d,f,g,h\}$.

This situation is often encountered: for example in a scene with trees, when the branches of the two trees are interlaced. Another case is the brain, where invaginations of superficial cortical grey matter (cf. a,b,c,d in Fig. 2) must be distinguished from deep structures which are adjacent but distinct (e,f,g in Fig. 2). Then, in that case it is not possible to make a previous grouping.

Then, we have to define a new class of problems called arc-consistency problems with bilevel constraints. It is associated with the $FDCSP_{BC}$ and it is defined as follows:

Definition 5. Let $(i,j) \in arc(G)$. Arc (i,j) is arc-consistent with respect to D_i and D_j iff $\forall v \in S_i$, $\exists t \in S_i$, $\exists w \in S_j$: $Cmpi(v,t)$ and $C_{ij}(t,w)$. (v and t could be identical.)

The definition of an arc-consistent graph, given Definition 5, remains unchanged. The purpose of an arc-consistency algorithm with bilevel constraints is, given a graph G and a set P, to compute P', the largest arc-consistent domain with bilevel constraints for G in P.

3. $AC4_{BC}$ algorithm: arc-consistency algorithm with bilevel constraints

3.1. Principle

Considering the previous remarks, we propose a new algorithm working with bilevel constraints whatever they are. For that purpose, we adapt the AC_4 algorithm proposed

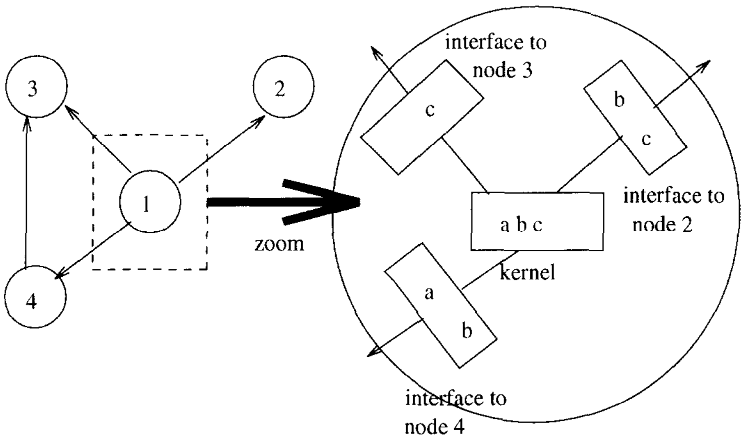

Fig. 3. Structure of a node. *a*, *b* and *c* are the labels which satisfy the unary relations of the node 1.

by Mohr and Henderson in 1986 [2,5,19] to solve the AC_{BC} problem. We call this algorithm AC_{4BC} (cf. Fig. 4).

For AC_{4BC}, we give a new definition of a node i belonging to $node(G)$.

Now a node is made up of a kernel and a set of interfaces associated with each arc which comes from another linked node (cf. Fig. 3). In addition, an intra-node compatibility relation *Cmpi* is associated with each node of the graph. It describes the semantic link between different subparts of an object which could be associated with the node.

Definition 6. Let $i \in node(G)$, then D_i is the domain corresponding to the kernel of i and the set $I_i = \{D_{ij} \mid (i,j) \in arc(G)\}$ is the set of interfaces of i.

As in algorithm AC_4, the domains are initialized with values satisfying unary node constraints and there are two main steps: an initialization step and a pruning step. However, whereas in AC_4 a value was removed from a node i if it had no direct support, in AC_{4BC}, a value is removed if it has no direct support and no indirect support obtained by using the compatibility relation *Cmpi*.

Then, for each $i \in node(G)$, the initialization step initializes the domains D_i and D_{ij} (cf. Fig. 4). This step consists of:

- assigning to the kernel D_i all the values b which satisfy the unary node constraints, as in the AC_4 algorithm (for example, in image analysis we can consider criteria of shape, size or orientation),
- assigning to the interfaces D_{ij}, all the values $b \in D_i$ such that $\exists c \in D_j\ C_{ij}(b,c)$.

Then we have a cleaning step which removes values which do not satisfy local constraint. The cleaning step of a kernel is done by the procedure CleanKernel.

The pruning step updates the nodes as a function of the removals made by the previous step to keep the arc consistency. For each couple (j,w) where $j \in node(G)$ and $w \in D_j$, AC_{4BC} associates a set of couples (i,v) ($i \in node(G)$ and $v \in D_i$) supported by (j,w).

```
begin $AC_{4BC}$
  Step 1: Construction of the data structures.
  InitQueue($Q$);
    for each $i \in node(G)$ do
      for each $b \in D_i$ do
        begin
          $S[i,b]$ := empty set;
        end;
    for each $(i,j) \in arc(G)$ do
      for each $b \in D_{ij}$ do
        begin
          $Total := 0$;
            for each $c \in D_{ji}$ do
              if $C_{ij}(b,c)$ then
                begin
                  $Total := Total + 1$
                  $S[j,c] := S[j,c] \cup (i,b)$;
                end
            if $Total = 0$ then
              begin
                $D_{ij} := D_{ij} - \{b\}$;
              end;
            else $Counter[(i,j),b] := Total$;
        end;
    for each $i \in node(G)$ do
      CleanKernel($D_i, I_i, Q$);

  Step 2: Pruning the inconsistent labels
  While not EmptyQueue($Q$) do
    begin
      DeQueue($i,b,Q$);
        for each $(j,c) \in S[i,b]$ do
          begin
            $Counter[(i,j),c] := Counter[(i,j),c] - 1$;
            if $Counter[(i,j),c] = 0$ then
              begin
                $D_{ij} := D_{ij} - \{c\}$;
                CleanKernel($D_i, I_i, Q$);
              end;
          end;
    end;
  end $AC_{4BC}$;
```

Fig. 4. The AC_{4BC} algorithm.

These sets are noted by $S[j,w]$. It uses a counter, $Counter[(i,j),b]$, which is the number of supports for the value b and the constraint associated with the arc (i,j). When a label is removed from one interface, the procedure CleanKernel is called such that the kernel in question is updated (cf. Fig. 4).

In addition, this algorithm, like all algorithms for arc consistency, works with a queue containing elements removed from the domains and which have to be reconsidered by the algorithm. In AC_{4BC} it contains pairs (i,v), where $i \in node(G)$ and $v \in D_i$. Those elements have to be reconsidered by the algorithm because they could support other couples (j,w). If a removed element was the unique support of (j,w) then (j,w) has to be removed as well. The study of operations on the queue will help us to prove properties of this algorithm. To manage the queue, we need several operations:

- The procedure InitQueue which initializes the queue to an empty set.
- The function EmptyQueue which tests if the queue is empty.
- The procedure EnQueue(i,v,Q) is used whenever the value v is removed from D_i. It introduces elements (i,v) in the queue Q, where i is a node and $v \in D$.
- The procedure DeQueue removes one element from the queue.

All these operations on queues require the same computational time. The procedure CleanKernel uses this notion of queue and is defined as follows (cf. Fig. 5):

procedure CleanKernel(in D_i, I_i, inout Q)
Pre: $i \in node(G)$, $D_i \neq \{\}$, $\forall D_{ij} \in I_i$, $D_{ij} \neq \{\}$
Post: $\Delta_i = \{b \in D_i \mid \exists D_{ij} \in I_i,\ \neg Path_i(b, D_{ij})\} Q = Q_{prev} \cup \Delta_i$.

where $Path_i(b, D_{ij})$ denotes the existence of a path in the node i between $b \in D_i$ and the interface D_{ij} according to the intra-node compatibility relation $Cmpi$. Then $Path_i(b, D_{ij})$ iff $\exists c \in D_{ij}$ such that $Cmpi(b,c)$.

3.2. Properties of AC_{4BC}

3.2.1. Termination of AC_{4BC}

In order to prove the termination of our algorithm, we use a data structure Status also introduced by [19] to prove the termination of AC_5; this data structure is a two-dimensional array, the first dimension being on nodes and the second on values. However, to prove the termination of AC_{4BC} we need an additive possible status called "rejected interface".

Then, the AC_{4BC} algorithm has to preserve the following invariant:

$$Status(i,b) = \begin{cases} \text{present iff } b \in D_i \\ \text{rejected iff } b \notin D_i \text{ and } (i,b) \notin Q \\ \text{suspended iff } b \notin D_i \text{ and } (i,b) \in Q \\ \text{rejected interface iff } \exists j \in node(G),\ b \notin D_{ij} \text{ and } b \in D_i. \end{cases}$$

Then the effect of the procedures manipulating the queue on *Status* is:

procedure InitQueue(out Q)
Post: $\forall i \in node(G)$,
$Status(i,b)$ = present if $b \in D_i$, rejected if $b \notin D_i$

```
Procedure CleanKernel(in $D_i, I_i$, out $Q$)
begin
  for each $b \in D_i$ do
    for each $D_{ij} \in I_i$ do
      if $\neg Path_i(b, D_{ij})$ then
        begin
          EnQueue($i, b, Q$);
          $D_i = D_i - \{b\}$;
            for each $D_{ij} \in I_i$ do
              begin
                $D_{ij} = D_{ij} - \{b\}$;
              end;
        end;
end;
```

Fig. 5. Implementation of CleanKernel.

procedure EnQueue(in i, b, inout Q)
Pre: $i \in node(G)$, $b \in D_i$
$Status(i, b)$ = rejected interface
Post: $Status(i, b)$ = suspended.

function EmptyQueue(in Q)
Post: $\forall i \in node(G)$, $\forall b \in D_i$
$Status(i, b) \neq$ suspended.

procedure DeQueue(inout Q, out i, b)
Post: $Status(i, b)$ = rejected

Then the effect of CleanKernel on *Status* is:

procedure CleanKernel(in D_i, I_i, inout Q)
Pre: $i \in node(G)$, $D_i \neq \{\}$, $\forall D_{ij} \in I_i$, $D_{ij} \neq \{\}$, $\exists b \in D_i$
such that $Status(i, b)$ = rejected interface.
Post: $\Delta_i = \{b \in D_i \mid \exists D_{ij} \in I_i, \neg Path_i(b, D_{ij})\}$ and $\forall b \in \Delta_i$
$Status(i, b)$ = suspended and $Q = Q_{prev} \cup \Delta_i$.

A simple implementation of CleanKernel is shown in Fig. 5.
Then we can prove the following theorem:

Theorem 7. *Algorithm AC_{4BC} (cf. Fig. 4) has the following properties:*
(1) *The invariant on data structure Status holds on line* 2 *and* 25.
(2) *AC_{4BC} enqueues and dequeues at most* $O(nd)$ *elements, and hence the size of the queue is at most* $O(nd)$, *where n is the number of nodes.*
(3) *AC_{4BC} always terminates.*

Proof. To prove this theorem we consider the algorithm of Fig. 4.

Property (1) holds initially. Assuming that it holds in line 2, it is still true after the iteration 7–22. The line 17 makes sure that (i,b) is rejected interface for all b such that $\exists j \in node(G)$ with $Counter[(i,j),b] = 0$. Then the invariant is true in line 25. Indeed the post-condition of CleanKernel makes sure that $\forall b \in D_i$, $\forall i \in node(G)$, (i,b) is suspended if $\exists j \in node(G)$ such that $\neg Path_i(b, D_{ij})$. The execution of lines 26–37 preserves this invariant. Lines 33–34 make sure that (j,c) is rejected interface for all c such that there exists $i \in node(G)$ where $Counter[(j,i),c] = 0$. Line 34 corresponding to the calling of CleanKernel preserves the invariant as seen before. So the invariant is verified at line 2 and 25.

Property (2) holds because each element of *Status* is allowed to make only three transitions:

- one from present to rejected interface through lines 19 and 33.
- one from rejected interface to suspended through the procedure CleanKernel.
- one from suspended to rejected through procedure DeQueue.

Hence there can only be $O(nd)$ dequeues and enqueues.

Property (3) is a direct consequence of Properties (1) and (2) and the preconditions of the procedure Enqueue on the data structure *Status*. □

3.2.2. *Correctness of AC_{4BC}*

Next, we can prove the correctness of AC_{4BC}.

Theorem 8. *G is arc-consistent when AC_{4BC} terminates.*

Proof. Initial hypothesis: Let $i \in node(G)$ such that $\exists(i,j) \in arc(G)$ and $\exists c \in D_i$. Assume that c is not supported directly or indirectly by the node j when AC_{4BC} terminates. In other words, $\forall b \in D_{ji}$, $\forall a \in D_{ij}$, $\neg Cmpi(c,a)$ or $\neg C_{ij}(a,b)$.

If $c \in D_i$ is not supported then either (1) it has never been supported, or (2) it was supported at a previous time.

Case 1: In this case, there is a contradiction with $c \in D_i$ because the initialization step would have not put c in D_i (Line 7 of AC_{4BC}).

Case 2:

- Let $b_1 \ldots b_m$ $(m > 0)$ be the set of elements of D_j supporting c at this previous time. Since at the end c is not supported it means that $b_1 \ldots b_m$ are removed during AC_{4BC} execution from D_{ji}.
- The removing of $b_1 \ldots b_m$ inserts them in the queue Q.
- All these elements $b_1 \ldots b_m$ are necessarily dequeued from Q when AC_{4BC} terminates and the $Counter[(j,i),c]$ becomes necessarily equal to zero when AC_{4BC} terminates (lines 27–30).
- At this time c is removed from the interface D_{ij} (line 33).
- By hypothesis c is not supported indirectly.
  It means that $\forall a \in D_{ij}$, $\neg Cmpi(c,a)$ or $\neg C_{ij}(a,b)$.
  Then we have two cases to study:
  - First, if we have $\forall a \in D_{ij}$, $\neg Cmpi(c,a)$, then we have $\neg Path_i(c, D_{ij})$. In this case CleanKernel removes c from D_i (lines 4–7 of Fig. 5) which is contrary to the initial hypothesis.

```
Procedure CleanKernel(in $D_i, I_i$, out $Q$)
1  begin
2    for each $D_{ij} \in I_i$ do
3      begin
4        $R := D_{ij}$;
5        While (SearchSucc($D_i, R, Cmpi, S$)) do
6          begin
7            $R := R \cup S$;
8          end
9        for each $b \in D_i - R$ do
10         begin
11           EnQueue($i, b, Q$);
12             for each $D_{ij} \in I_i$ do
13               $D_{ij} := D_{ij} - \{b\}$;
14         end;
15       $D_i := R$;
16     end;
17 end;
```

Fig. 6. Optimized implementation of CleanKernel.

- The second case is $\forall b \in D_{ji}$, $\forall a \in D_{ij}$, $Cmpi(c,a)$ and $\neg C_{ij}(a,b)$. This case cannot happen. Indeed from line 17 of AC_{4BC}, $\forall b \in D_j$ $\neg C_{ij}(a,b) \Rightarrow a \notin D_{ij}$. The contra positive statement yields $a \in D_{ij} \Rightarrow \exists b \in D_{ji}$ $C_{ij}(a,b)$. So $\forall a \in D_{ij}$, $Cmpi(c,a) \Rightarrow \exists b \in D_{ji}$ such that $C_{ij}(a,b)$.

In conclusion the initial hypothesis leads to a contradiction. So G is arc-consistent when AC_{4BC} terminates. □

3.2.3. *Complexity of AC_{4BC}*

The implementation of CleanKernel presented in Section 3.1 (Fig. 5) was given for pedagogical reasons but it is not optimal in time. However, we can find another way to implement this procedure (cf. Fig. 6). We introduce the function SearchSucc(in D_i, R, $Cmpi$, out S) which looks for successors of elements of D_i in the set R by using the relation $Cmpi$. Each new successor is marked such that successors already encountered will not be considered again. This function is repeated until no new successor can be found. Once we quit the loop, regions which are not in the set R have to be suspended. Indeed, $\forall b \in D_i$, if $b \notin R$ then $\exists D_{ij} \in I_i$, $\forall c \in D_{ij}$ $\neg Cmpi(b,c)$. The Post and Pre conditions of the function SearchSucc are defined as follows:

Function SearchSucc(in $D_i, R, Cmpi$, out S)
Pre: $i \in node(G)$, $D_i \neq \{\}$
Post: $S = \{b \in D_i \mid b \notin R,\ \exists c \in R,\ Cmpi(b,c)\}$ and SearchSucc $\Leftrightarrow$ ($S \neq \{\}$)

Theorem 9. CleanKernel *is in* $O(n^2 d)$ *in the worst case.*

Proof. The number of Interfaces D_{ij} to check is at most equal to n. The function SearchSucc is such that an element already in R can not be added again to R. Since the size of R is bounded by d, the loop in line 5 is repeated at most d times. As the time complexity of lines 9-14 is in $O(nd)$ the time complexity of CleanKernel is in $O(n^2d)$. □

Theorem 10. *The time complexity of AC_{4BC} is bounded by $O(en^3d^2)$ in the worst case.*

Proof. As in the AC_4 algorithm the time complexity of lines 1-22 is in $O(ed^2)$. In line 23 the procedure CleanKernel is called n times. Then the time complexity of lines 23-24 is in $O(n^3d)$. Then the time complexity of the initialization step is in $O(ed^2 + n^3d)$. As in AC_4 algorithm the line 30 is executed ed^2 times. The test of line 31 is true at most ed times, then CleanKernel is executed at most ed times. The time complexity of line 29-36 is in $O(en^2d^2)$. Then the complexity is in $O(n^3d + ed^2 + en^2d^2)$. This complexity is bounded by $O(en^3d^2)$ in the worst case. Then the time complexity of AC_{4BC} is bounded by $O(en^3d^2)$ in the worst case. □

Theorem 11. *If the graph is totally connected and if there are no more than two relations between two nodes then the time complexity of AC_{4BC} is in $O(e^2d^2)$ in the worst case.*

Proof. If the graph is totally connected then we can say that $e = n^2 - n$. Then $O(e) = O(n^2)$. As in the general case the complexity is in $O(n^3d + ed^2 + en^2d^2)$, we get $O(n^3d + ed^2 + en^2d^2) = O(n^3d + n^2d^2 + n^4d^2) = O(n^4d^2) = O(e^2d^2)$. Then in that case the time complexity of AC_{4BC} is in $O(e^2d^2)$ in the worst case. □

4. Application

The algorithm was applied to a problem of image interpretation [8]. We worked with a set of Nuclear Magnetic Resonance cerebral images. The aim is to detect the main anatomical cerebral regions (cortex, basal nuclei, thalamus, etc.). Anatomical textbooks describe every anatomical part of the brain in terms of unary relations (shape, size, orientation) and binary relations (spatial relations between two parts). We represented this knowledge in a semantic graph corresponding to spatial relations of brain grey matter structures (Fig. 7). To simplify this graph, the interfaces of each node are not drawn, but in fact each node has the structure described in Fig. 3. The algorithm of segmentation described in [7] provides 200 regions, some of which are over-segmented. For each anatomical part (node of the semantic graph), we define unary relations corresponding to shape, size and orientation criteria. These criteria are stored in a file for each segmented region. Only segmented regions satisfying the relations associated with the node in question are assigned to the kernel of this node. It was also necessary to build for each segmented region b_p a set $T_{up}(b_p)$ of regions above b_p and a set $T_{down}(b_p)$ of regions below b_p. In each of these sets, we distinguish regions which may belong to the same object (more then 30% of pixels overlap b_p) as b_p from those that do not belong to

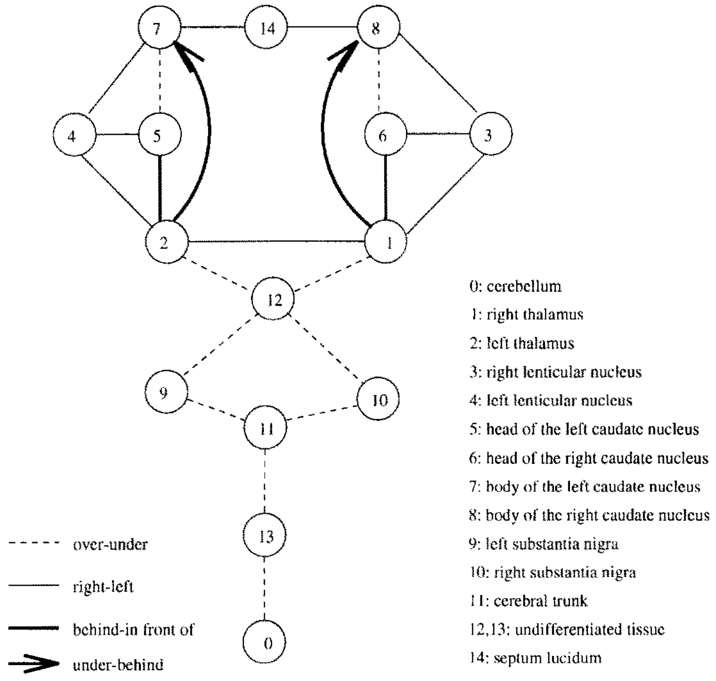
7
14
8
4
5
6
3
2
1
12
9
10
11
13
0
over-under
right-left
behind-in front of
under-behind
0: cerebellum
1: right thalamus
2: left thalamus
3: right lenticular nucleus
4: left lenticular nucleus
5: head of the left caudate nucleus
6: head of the right caudate nucleus
7: body of the left caudate nucleus
8: body of the right caudate nucleus
9: left substantia nigra
10: right substantia nigra
11: cerebral trunk
12,13: undifferentiated tissue
14: septum lucidum

Fig. 7. Semantic graph to label the human brain.

the same object as b_p. Then, let $T(b_p) = T_{up}(b_p) \cup T_{down}(b_p)$ and $I(b_p)$ be the set of pixels of the region b_p. If $c_l \in T(b_p)$ and $I(b_p) \cap I(c_l) > 30\%$ then b_p and c_l could belong to the same object. If i is the node associated with this object then $Cmpi(b_p, c_l)$.

The semantic graph has 14 nodes and 44 arcs. Tests have been made successfully on twenty images and more than 200 regions. After a short time (2 min 30 sec on an HP710, 50 MHz, with 32 MByte RAM), each anatomical part is correctly identified. We can remark that this algorithm is particularly adapted to this problem: the different parts of the brain always have the same spatial relations with one another, even if the distances can change from one brain to another. Moreover, the cerebral structures are all in close relation with one another, with much redundancy in the spatial relations. This redundancy sufficiently constrains the data to avoid undecidability between several solutions. For other images with another semantic graph, the arc consistency might be insufficient for solving the problem and in that case we may need path or global consistency.

5. Conclusion

Until now, few applications of image interpretation have used semantic graph and arc-consistency checking. This is because usually, the classical definition of *FDCSP* that governs *AC* checking does not fit well with the data to analyze. Indeed, perfect image segmentation is very rare and merging regions often requires expert knowledge. This knowledge is necessary for region labeling as well. In fact, region merging and region labeling are interdependent and the classical strategy of arc-consistency checking cannot cope with this difficulty. The extension of *AC* for *FDCSP* with bilevel constraints solves this problem and provides a more general tool. Moreover the proposed extension of AC_4 retains the good properties of AC_4 and has a reasonable time complexity. The notion of intra-node compatibility introduced in $FDCSP_{BC}$ can also be adapted to AC_5 [19] and AC_6 [2] because it does not basically change the way of checking arc consistency. We only change the definition of "node" by introducing for a node the notions of "kernel" and "interfaces". The CleanKernel procedure can easily be adapted for AC_5 and AC_6 in the same way as for AC_4. To avoid a long and tedious formal development far removed from our initial need to label NMR images, we have limited our discussion to arc consistency for binary relations. However, the framework of $FDCSP_{BC}$ can be extended to n-ary relations as defined in [9] and to path-consistency checking as well [13], providing a larger field of application for the constraint satisfaction approach.

Acknowledgements

We thank Dr Paul Bailey and Dr Nicolas Bolo for their assistance with the preparation of the manuscript.

References

[1] D.H. Ballard and C.M. Brown, *Computer Vision* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[2] C. Bessière, Arc consistency and arc consistency again, *Artificial Intelligence* **65** (1994) 179–190.
[3] C. Bessière and J.C. Régin, An arc consistency algorithm optimal in the number of constraint checks, in: *Proceedings 6th IEEE International Conference on Tools for Artificial Intelligence*, New Orleans, LA (1994) 397–403.
[4] A. Belaïd and Y. Belaïd, *Reconnaissance des Formes, Methodes et Applications* (InterEditions, Paris, 1992).
[5] J. Benmouffek, Y. Belaïd, A. Belaïd and L. De Minacelli, RER: un système de reconnaissance d'empreintes de rats, in: *Proceedings 8ième Congrés AFCET: Reconnaissance des Formes et Intelligence Artificielle*, Lyon, France (1991).
[6] P. Charman, A constraint based approach for the generation of floor plans, in: *Proceedings 6th IEEE International Conference on Tools for Artificial Intelligence*, New Orleans, LA (1994) 555–561.
[7] A. Deruyver, Y. Hodé and L. Soufflet, A segmentation technique for cerebral NMR images, in: *Proceedings IEEE Conference on Image Processing 94*, Austin, TX (1994) 716–720.
[8] A. Deruyver and Y. Hodé, Semantic graph and arc consistency in "true" three dimensional image labeling, in: *Proceedings IEEE International Conference on Image Processing 95*, Washington, DC (1995) 619–622.
[9] H. Tolbat, F. Charpillet and J.P. Haton, Representing and propagating constraints in temporal reasoning, in: *Proceedings IEEE International Conference on Tools for Artificial Intelligence*, San Jose, CA (1991) 181–184.
[10] T. Kökény, A new arc consistency algorithm for CSPs with hierarchical domains, in: *Proceedings 6th IEEE International Conference on Tools with Artificial Intelligence*, New Orleans, LA (1994) 439–445.
[11] A.K. Mackworth, Consistency in networks of relations, *Artificial Intelligence* **8** (1977) 99–118.
[12] A.K. Mackworth and E.C. Freuder, The complexity of some polynomial network consistency algorithms for constraint satisfaction problems, *Artificial Intelligence* **25** (1985) 65–74.

[13] R. Mohr and T.C. Henderson, Arc and path consistency revisited, *Artificial Intelligence* **28** (1986) 225-233.

[14] R. Mohr and G. Masini, Good old discrete relaxation, in: *Proceedings ECAI-88*, Munich, Germany (1988) 651-656.

[15] J.A. Mulder, A.K. Mackworth and W.S. Havens, Knowledge structuring and constraint satisfaction: the MAPSEE approach, *IEEE Trans. Pattern Anal. Machine Intelligence* **10** (1988) 866-879.

[16] H. Niemann, G.F. Sagerer, S. Schréder and F. Kummert, ERNEST: a semantic network system for pattern understanding, *IEEE Trans. Pattern Anal. Machine Intelligence* **12** (1990) 883-905.

[17] M. Pelillo and M. Refice, Learning compatibility coefficients for relaxation labeling processes, *IEEE Trans. Pattern Anal. Machine Intelligence* **16** (1994) 933-945.

[18] A. Rosenfeld, R. Hummel and S. Zucker, Scene labeling by relaxation operations, *IEEE Trans. Systems Man Cybernet.* **6** (1976) 420-433.

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

[20] D.L. Waltz, Understanding line drawing of scenes with shadows, in: P.H. Winston, ed., *Psychology of Computer Vision* (McGraw-Hill, New York, 1975) 19-91.