Università degli Studi di Bologna
DEIS

Interactive Constraint Satisfaction

Rita Cucchiara Evelina Lamma Paola Mello
Michela Milano

April 15, 1997

DEIS Technical Report no. DEIS-LIA-97-005 *LIA Series no. 22*

Interactive Constraint Satisfaction

Rita Cucchiara [2] Evelina Lamma [1] Paola Mello [2]
Michela Milano [1]

[1] *DEIS, Università di Bologna*
Viale Risorgimento 2, 40136 Bologna, Italy
Tel: +39 51 6443033, Fax: +39 51 6443073
{elamma,mmilano}@deis.unibo.it
[2] *Dipartimento di Ingegneria, Università di Ferrara*
Via Saragat 1, 41100 Ferrara, Italy
{rcucchiara,pmello}@deis.unibo.it

April 15, 1997

Abstract. We present an *Interactive Constraint Satisfaction* model for problems where knowledge is not completely defined at the beginning of the computation, but can be interactively acquired during the computational process. In this case, domains can be partially defined when the constraint satisfaction process starts. Thus, domains are dynamically defined in contrast with classical CSPs where domains are statically defined at the beginning of the constraint satisfaction process. The interactive CSP model copes with incomplete domain definition by propagating constraints on already defined domain values and by adding new constraints on undefined parts of domains. These new constraints can be used to incrementally process new information without restarting a constraint propagation process from scratch each time new information is available. In addition, these constraints can be eventually used in order to guide the knowledge acquisition process. Examples are given in order to show the effectiveness of the approach. In particular, a case study in the field of visual object recognition is considered.

Keywords: *Constraint Satisfaction, Vision*

Contents

1 Introduction

Constraint Satisfaction systems provide a simple but powerful framework for solving a variety of Artificial Intelligence (AI) problems. A survey on these techniques can be found in [9]. Constraint Satisfaction Problems (CSP, for short in the following) are defined on a finite set of variables each ranging on a finite domain (numerical or symbolic domain) and a set of constraints. A solution to a CSP is an assignment of values to variables which satisfies the constraints. A number of consistency techniques (e.g., node, arc, path consistency) have been proposed [10, 11, 12] based on the active use of constraints during the search process. The idea is to remove during the search, by means of constraint propagation, combination of assignments which cannot appear in any consistent solution.

A CSP based inference engine can be used in many applications. In particular, we focus on those application where a low level system (for example a vision system or a sensor based system) provides (constrained) data to be processed. These applications usually require some form of interaction between the low-level module and the CSP module. In classical CSPs variable domains have to be completely defined before the constraint satisfaction process starts. This knowledge acquisition and its processing are sequentially performed thus leading to an inefficient behavior of the whole system especially when the knowledge acquisition process is computationally expensive. A better approach would intertwine the knowledge acquisition with its processing in a sort of pipelined architecture (with feedback) where as soon as some data are available from the low level system, they are processed. For example, in the field of object recognition, we can think of an architecture where a visual system provides low level image features and a constraint-based engine which processes these features in order to recognize a particular object.

For this purpose we present an *Interactive Constraint Satisfaction* model where domains can be partially defined when the constraint satisfaction process starts. Thus domains are dynamically defined during the computation in contrast with classical CSPs where domains are statically defined at the beginning of the constraint satisfaction process.

The interactive CSP model copes with incomplete domain definition by propagating constraints on already defined domain values and by adding new constraints on undefined parts of domains. These new constraints can be used to incrementally process new information without restarting a constraint propagation process from scratch each time new information is available. In addition, these constraints can be eventually used in order to guide the knowledge acquisition process.

The main contributions of the paper are the following:

- define the above mentioned *Interactive Constraint Satisfaction* model;
- provide an arc-consistency algorithm that properly extends AC-4 [11] in order

to cope with partially defined domains;

- apply the model to a case study in the field of visual object recognition.

The paper is organized as follows: in section 2, we describe the interactive CSP model and we explain how to use the model for guiding knowledge acquisition. Section 3 provides an interactive arc-consistency algorithm based on the notion of support as the algorithm AC-4 does. A case study is presented in section 4. A discussion and future works conclude the paper.

2 Interactive CSP

In this section, we provide the interactive CSP model. We first start by giving some preliminaries on CSPs. A CSP can be defined on a set of variables $X_1, X_2, \ldots, X_n$ ranging respectively on finite domains $D_1, D_2, \ldots, D_n$. A constraint $c(X_1, \ldots X_k)$ defines a subset of the cartesian product of $D_1, \ldots, D_k$, i.e., a set of configurations of assignments which can appear in a consistent solution (if it exists). A CSP can be represented by means of a *constraint network* where each node is a variable and arcs are constraints.

A CSP can be solved by means of constraint propagation. Many consistency techniques (node, arc, path consistency) have been proposed [10, 11, 12] based on the active use of constraints during the search process.

We refer in the paper to the *arc-consistency* algorithm. An arc in the CSP representing a binary constraint $c(X_i, X_j)$ is arc-consistent if and only if for each value in the domain D_i for variable X_i there exists at least one value in the domain D_j for variable X_j which is consistent with the constraint $c(X_i, X_j)$. A constraint network is arc-consistent if and only if each arc is arc-consistent. A solution to a CSP is an assignment of values to variables which is consistent with constraints. A solution can be found by intertwining an arc-consistency propagation algorithm with a *labeling* strategy which performs a guess on a variable value thus exploring the solution space.

A standard CSP approach needs all the information and the knowledge on the problem at the beginning of the computation. Then it propagates constraints by removing assignments which cannot appear in any consistent solution. The interaction with a low level system and the consequent propagation requires a partial acquisition of knowledge which lasts during all the computation process. Therefore, we have to change the CSP model we use and allow the propagation algorithm to work on partially defined domains.

To this purpose, we define an *Interactive CSP* (*ICSP*) model which has to:

- cope with incomplete domain definition. Therefore, domains can be partially defined in the sense that some domain elements can be already at disposal for propagation, while other domain elements have to be acquired from a low level system in the future;

- guide knowledge acquisition by means of constraints. Therefore, the ICSP system should generate at each computational step new constraints on the basis of already acquired knowledge involving the undefined part of the domains;

- incrementally process new information without restarting a constraint propagation process from scratch each time new information is available. Therefore, the constraint satisfaction model should maintain a representation of constraints on undefined values and check them against newly available information.

On the basis of these requirements, we define the ICSP model as follows:

Definition 2.1 *An interactive CSP (ICSP) is defined on a finite set of variables* $\{X_1, X_2, \ldots, X_n\}$ *each ranging on a partially defined domain* $\{D_1, D_2, \ldots, D_n\}$ *where each* $D_i = [Def_i \cup UnDef_i]$. Def_i *represents the defined part, while* $UnDef_i$ *is a domain variable itself representing information which is not yet available. Both* Def_i *and* $Undef_i$ *can be possibly empty*[1]. *Also, for each* i, $Def_i \cap Undef_i = \emptyset$. *A constraint among variables defines a possibly partially defined subset of the cartesian product of variable domains.*

A solution to the ICSP is, as in the case of the standard CSP, an assignment of values to variables which is consistent with constraints.

On the contrary, the constraint propagation is quite different from the standard case. Consider for the sake of clarity only binary constraints $c(X_i, X_j)$. In the more general case, in order to propagate the constraint $c(X_i, X_j)$ we have to propagate four kinds of constraints and collect the propagation results[2]:

$$c(Def_i, Def_j) \cup c(Def_i, Undef_j) \cup c(UnDef_i, Def_j) \cup c(UnDef_i, UnDef_j)$$

The propagation between definite domains is the usual constraint propagation, while propagation on undefined domains produces new constraints. These new constraints can be used in order to guide the search and to incrementally process newly acquired knowledge from the low-level system. Note that constraints on at least one undefined part intensionally represent potential solutions on not yet acquired knowledge.

Let us consider an example on integer numbers in order to clarify the behavior of the model. Then in section 4, we will apply the model for the recognition of an object in a vision system. Consider the constraint $X < Y$ where X ranges on the finite domain[3] $[1, X1]$, and Y ranges on $[-3, 2, Y1]$ where $X1$ and $Y1$ are domain

[1] When both are empty an inconsistency arises.

[2] We refer, with an abuse of notation, to domains instead of variables. However, we meaning is straightforward.

[3] The notation $[1, X1]$ is a shortcut of $[1] \cup [X1]$.

variables representing the undefined part of the domains. Suppose we are using an arc-consistency propagation algorithm. The propagation of constraints on definite parts is the following. Value $1 \in Def_X$ is supported from the value 2 in the domain of Y. Viceversa, while the value $2 \in Def_Y$ is supported by the value 1 in the domain of X, the value -3 is not supported and can be deleted from the domain of Y. The resulting domains from this first propagation are $X :: [1]$, $Y :: [2]$. It is worth noting that this first result of the propagation should be considered together with other propagations, i.e., $c(Def_i, Undef_j) \cup c(UnDef_i, Def_j) \cup c(UnDef_i, UnDef_j)$. In particular, the propagation between the defined part of X and the undefined part of Y leads to the generation of the following constraints: $Y1 > 1, Y1 \neq 2$ and $Y1 \neq -3$ (these last constraints derive from the definition of ICSP: $Def_i \cap UnDef_i = \emptyset$). The propagation between the defined part of Y and the undefined part of X leads to the generation of the following constraints: $X1 < 2$, $X1 \neq 1$ and a conditional constraint $Y \neq -3\ if\ X1 < -3$. This last constraint represents in some sense the support for the value -3 for Y. This constraint together with the the first propagation between defined domains results in the final domain for Y which contains also the value -3 since we are not yet sure that only values greater than -3 are provided for $X1$. Finally, the last propagation gives the constraint between the undefined parts which is exactly the same as the constraint between the original variables, i.e., $X1 < Y1$.

Therefore, the result of the propagation will be:
$X :: [1, X1], Y :: [-3, 2, Y1]$ such that
$X \neq 1,\ Y1 \neq 2\ Y1 \neq -3$
$((Y1 > 1, X1 < 2) \vee X1 < Y1), Y \neq -3\ if\ X1 < -3$

Let us now consider two possible knowledge acquisitions from the underlying system.

- Suppose that a new value, -7, for X is now available. This value can be checked only against newly introduced constraints. In fact, the value -7 becomes part of variable $X1$'s domain which is now $X1 :: [-7, X2]$. Therefore, the constraint $X1 < 2$ is satisfied provided that $X2 < 2$. The conditional constraint $Y \neq -3\ if\ X1 < -3$ is now satisfied and can be removed. The constraint $X1 < Y1$ becomes $X2 < Y1$.

- Suppose that a new value 7 for X is now available. The value 7 becomes part of the variable $X1$ domain which is $X1 :: [7, X2]$. The constraint $X1 < 2$ is not satisfied but the other constraint in the disjunction $X1 < Y1$ can be satisfied if $7 < Y1$. This constraint is a conditional constraint for the support of value 7: $X \neq 7\ if\ Y1 > 7$.

When the knowledge acquisition ends, undefined domains become empty, constraints can be solved and the constraint propagation is the same as in usual CSP models.

Another requirement of the interactive CSP concerns the possibility of guiding the search by means of constraints. In fact, the constraints

$X :: [1, X1], Y :: [-3, 2, Y1]$ such that

$X \neq 1,\ Y1 \neq 2\ Y1 \neq -3$

$((Y1 > 1, X1 < 2) \vee X1 < Y1), Y \neq -3\ if\ X1 < -3$

can be used in order to guide the acquisition of variable values. This can be useful when the knowledge acquisition process is very expensive. Suppose we are looking for a value for Y. We can collect all the constraints involving $Y1$ and guide the search in order to find consistent values with the constraints. For example we can use the constraint $Y1 > 1$, $Y1 \neq 2$, and $Y1 \neq -3$ in order to collect values which are already consistent with the definite parts of X and Y. The constraints between undefined parts cannot be used in general except for the case that the underlying system is able to provide couples of values already consistent with this constraint.

3 Interactive Arc-Consistency

In this section, we present an arc-consistency algorithm for coping with partially defined domains and with interactive knowledge acquisition. The algorithm is based on the notion of *support*, as the algorithm AC-4 [11]: a value is supported if there exists a compatible value in the domain of every other variable. For this purpose the algorithm AC-4 builds during the computation the *support sets*. Each support set can be represented as follows:

$S_{\langle j,c \rangle} = \{(i, b) | c_{i,j}(b, c)\ holds\}$.

representing the fact that value c for variable X_j supports all values b for variable X_i contained in the set. Besides the support sets, AC-4 works on the following data structures: a set of counters which count how many supports a variable value has and a boolean matrix which marks the labels that have been rejected. The difference of our algorithm with AC-4 consists in the fact that the support of a value can be also represented by a domain variable. In fact as far as the knowledge acquisition is not yet ended, we cannot delete any variable value since it can be possibly supported by other values which have not been already acquired. Therefore, we can commit to a deletion of a value from a variable domain when the knowledge acquisition process ends, i.e., when the low-level system provides a *null* value for a variable.

The interactive arc-consistency algorithm is structured as follows:

- Building of initial support sets (procedure support_init);
- acquisition of a new value v for a variable X (procedure acquire_new_value which is not here described since it depends from the lower level system);
- if the value is a termination value ($v = null$), on the basis of counters we can possibly delete some values from other variable domains which are not

```
procedure IAC(C, D)
begin
support_init(D,C,S,Count);
while not end_acquisition do
      begin
      acquire_new_value(Val,Var);
      process(Val,Var,D,S,Count);
      end;
end.
```

Figure 1: The incremental consistency algorithm

supported by values in the domain of X;

- otherwise, split all couples containing the undefined part of X in two parts: one containing the acquired value and the other still undefined part of the domain of X.

The algorithm is depicted in figure 1. In figure 2, we report the code for the initialization procedure of data structures, i.e., support sets and counters. Parameters of the procedure are domains D, constraints C, support sets which are here initialized S and counters $Count$. Note that at the end of the procedure no counter can be equal to zero as in the usual arc-consistency algorithm since each value is supported by at least the undefined part of other variables. If we suppose that some variable domain can be completely defined a control is needed as in the original AC-4 algorithm.

It is worth noting that the procedure satisfies$((b,c), C_{ij})$ in figure 2 is true in the following cases:

- b and c are both ground and the constraint C_{ij} is satisfied with this values;
- b or c are undefined.

We now explain the behavior of the procedure for processing newly acquired knowledge presented in figure 3. Two cases are considered: the end of acquisition for variable j, i.e., v is a *null* value, and the case where v is a new *non null* value which should be processed. Let us start considering the first case. The undefined part of j, $Undef_j$ becomes empty. Therefore, the counters for all values supported by $Undef_j$ should be decremented by 1. If one of those counters reaches the value 0, i.e., the value b for variable i was supported only by the undefined part of j, b should be deleted form the domain of i and added to the list *List* of unsupported labels.

```
procedure support_init(D, C, S, Count)
begin
/*Construction of the data structures */
M :=0; S:=∅;
for each C_ij in C do
      for each b in D_i do
            begin
            Total:=0;
            for each c in D_j do
                  if satisfies((b,c), C_ij) then
                        begin
                        Total := Total +1;
                        S_<j,c> := S_<j,c> + <i,b>;
                        end;
                  Counter[(i,j),b] := Total;
            end ;
end.
```

Figure 2: Procedure for the creation of initial support sets and counters

The processing of unsupported labels (the while loop in Step 2) is the same as for AC-4. Note, however, that only completely known domain variables can cause the deletion of other variables domains. In the case where value v is null the support set $S_{\langle j, Undef_j\rangle}$ should be removed from the list S of support sets and all couples $\langle j, Undef_j\rangle$ deleted from other value support sets.

On the contrary, if the acquisition for variable j provides a non null value v a new support set $S_{\langle j,v\rangle}$ should be initialized with all values that are supported by v previously posted in $S_{\langle j, Undef_j\rangle}$. Finally, the couple $\langle j, Undef_j\rangle$ in any other support set $S_{\langle i,b\rangle}$ is splitted in two parts one concerning the newly acquired value (if it supports value b for variable i) and the other regarding the still undefined part.

Now consider again the example presented in section 2 where variables $X :: [1, X1]$ and $Y :: [-3, 2, Y1]$ were linked by the constraint $X < Y$. We explain now how support sets are initialized and updated during the knowledge acquisition. Initial support sets built by the procedure support_init are the following:

$S_{\langle X,1\rangle} = \{\langle Y, 2\rangle, \langle Y, Y1\rangle\}$
$S_{\langle X,X1\rangle} = \{\langle Y, -3\rangle, \langle Y, 2\rangle, \langle Y, Y1\rangle\}$
$S_{\langle Y,-3\rangle} = \{\langle X, X1\rangle\}$
$S_{\langle Y,2\rangle} = \{\langle X, 1\rangle, \langle X, X1\rangle\}$
$S_{\langle Y,Y1\rangle} = \{\langle X, 1\rangle, \langle X, X1\rangle\}$

The counters are initialized as following:

```
procedure process(v,j,D,S,Count)
begin
if v = null /* end of the acquisition for variable j*/
      then begin
            for each ⟨i, b⟩ in S⟨j,Undef_j⟩ do begin
                  Counter[(i, j), b] := Counter[(i, j), b] -1;
                  if Counter[(i, j), b] = 0
                        then begin
                              D_i := D_i - {b}; M[i,b] :=1;
                        end;
                  end;
                        S:= S- S⟨j,Undef_j⟩;
                  for each ⟨j, Undef_j⟩ in S⟨i,b⟩ do
                        S⟨i,b⟩ := S⟨i,b⟩ − ⟨j, Undef_j⟩
      List := {⟨i, b⟩|M[i, b] = 1}; /*set of rejected labels*/
      while List ≠ ∅ do /* Step 2: remove unsupported labels */
            begin choose ⟨j, c⟩ from List;
            List := List - {⟨j, c⟩};
            for each ⟨i, b⟩ in S⟨j,c⟩ do
                  begin Counter[(i, j), b] := Counter[(i, j), b] - 1;
                  if (Counter[(i, j), b]=0) and (M[i, b] = 0) then
                        begin List := List + ⟨i, b⟩;
                        M[i, b]:=1; D_i := D_i - {b}; end; end
            S := S - S⟨j,c⟩; end; end
/* the acquisition for variable j provides non null values */
      else beginfor each ⟨i, b⟩ in S⟨j,Undef_j⟩ do
            if satisfies((b,v), C_ij) then
                  begin
                  Counter[(i, j), b] := Counter[(i, j), b] + 1;
                  S⟨j,v⟩ := S⟨j,v⟩ + ⟨i, b⟩;
                  end;
for each ⟨j, Undef_j⟩ in S⟨i,b⟩ do
      if satisfies((b,v), C_ij) then
            begin
            Total := Total +1;
            S⟨i,b⟩ = S⟨i,b⟩ + ⟨j, v⟩;
            end;
      end;
      Counter[(j, i), v] := Total;
end;
```

Figure 3: Procedure for processing new acquired knowledge

$Counter[(X,Y),1] = 2, Counter[(X,Y),X1] = 3, Counter[(Y,X),-3] = 1,$
$Counter[(Y,X),2] = 2, Counter[(Y,X),Y1] = 2$

Note that value -3 for variable Y has only one support represented by the undefined part of X. Therefore, if the next value acquired for variable X is $null$, counters are changed in the following way:

$Counter[(X,Y),1] = 2, Counter[(X,Y),X1] = 3, Counter[(Y,X),-3] = 0,$
$Counter[(Y,X),2] = 1, Counter[(Y,X),Y1] = 1$

Value -3 is no longer supported and can be deleted from the domain of Y and added to a list to be processed. However, in this case the deletion of value -3 does not influence other variable domains.

On the contrary, if in the initial condition we acquire for variable X the value -7, support sets and counters change as follows:

$S_{\langle X,1\rangle} = \{\langle Y,2\rangle, \langle Y,Y1\rangle\}$
$S_{\langle X,-7\rangle} = \{\langle Y,-3\rangle, \langle Y,2\rangle, \langle Y,Y1\rangle\}$
$S_{\langle X,X2\rangle} = \{\langle Y,-3\rangle, \langle Y,2\rangle, \langle Y,Y1\rangle\}$
$S_{\langle Y,-3\rangle} = \{\langle X,-7\rangle, \langle X,X2\rangle\}$
$S_{\langle Y,2\rangle} = \{\langle X,1\rangle, \langle X,-7\rangle, \langle X,X1\rangle\}$
$S_{\langle Y,Y1\rangle} = \{\langle X,1\rangle, \langle X,-7\rangle, \langle X,X1\rangle\}$
$Counter[(X,Y),1] = 2, Counter[(X,Y),X1] = 3, Counter[(X,Y),-7] = 3$
$Counter[(Y,X),-3] = 2, Counter[(Y,X),2] = 3, Counter[(Y,X),Y1] = 3$

Finally, if in the initial condition we acquire for variable X value 7, support sets and counters change as follows:

$S_{\langle X,1\rangle} = \{\langle Y,2\rangle, \langle Y,Y1\rangle\}$
$S_{\langle X,7\rangle} = \{\langle Y,Y1\rangle\}$
$S_{\langle X,X2\rangle} = \{\langle Y,-3\rangle, \langle Y,2\rangle, \langle Y,Y1\rangle\}$
$S_{\langle Y,-3\rangle} = \{\langle X,X2\rangle\}$
$S_{\langle Y,2\rangle} = \{\langle X,1\rangle, \langle X,X2\rangle\}$
$S_{\langle Y,Y1\rangle} = \{\langle X,1\rangle, \langle X,X2\rangle\}$
$Counter[(X,Y),1] = 2, Counter[(X,Y),X1] = 3, Counter[(Y,X),-3] = 1,$
$Counter[(Y,X),2] = 2, Counter[(Y,X),Y1] = 2$

4 A Case Study on Shape Recognition

In this section, we consider an example of application of ICSP to the recognition of shapes in images. In model based machine vision a critical point is an efficient description of object models by means of its visual primitives and their relations. Each object we want to recognize can be represented by means of a constraint graph where each object part or characterizing primitive feature is modeled by a node (variable) of the corresponding CSP and spatial or shape relations among object parts can be represented by arcs (constraints).

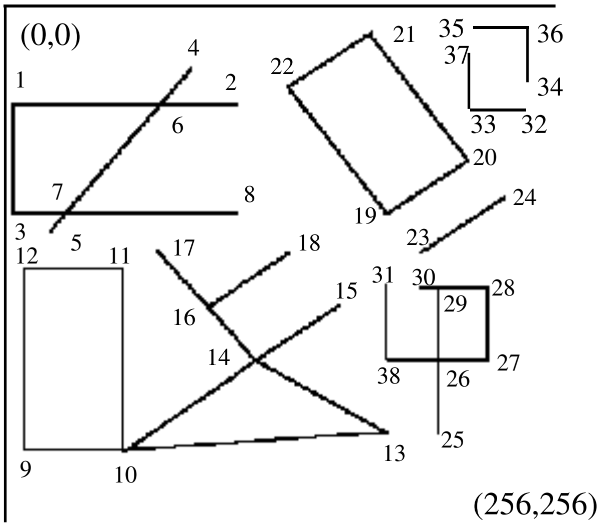

Figure 4: Example of a visual scene

In particular, we consider an example for the recognition of (one or more) rectangles in a scene depicted in figure 4. It is an image of 256 x 256 binary valued pixel, obtained after an edge detection of a grey-level image. Objects can be modeled by means of constraints and recognized by means of constraint satisfaction. A more detailed description of the problem, its solutions and related works can be found in [3].

Specific aspects of the single primitives may be modeled as unary constraints, such as the minimum length of an object part, its color, the planarity of a surface and so on, while geometric and topologic relationships between them can be represented by binary constraints (examples of geometric constraints are angular relationships between contours, lines, or surfaces, topologic constraints are spatial relationships, such as is *connected to*, *touch*, *is contained* or others).

Example 4.1 *If we want to model a rectangle, as shown in figure 5, we can identify four nodes corresponding to the four edges composing the rectangle (numbered respectively X_1, X_2, X_3 and X_4) and we impose the following symmetric constraints:*
$touch(X_1, X_2)$, $touch(X_2, X_3)$, $touch(X_3, X_4)$,
$touch(X_4, X_1)$, $no\_touch(X_2, X_4)$, $no\_touch(X_1, X_3)$,
$same\_lenght(X_2, X_4)$, $same\_lenght(X_1, X_3)$, $perpendicular(X_1, X_2)$,
$perpendicular(X_2, X_3)$, $perpendicular(X_3, X_4)$, $perpendicular(X_4, X_1)$,
$parallel(X_2, X_4)$, $parallel(X_1, X_3)$.

Some of these constraints are redundant. This redundancy is useful when constraints are propagated by means of an incomplete algorithm such as for example arc-consistency.

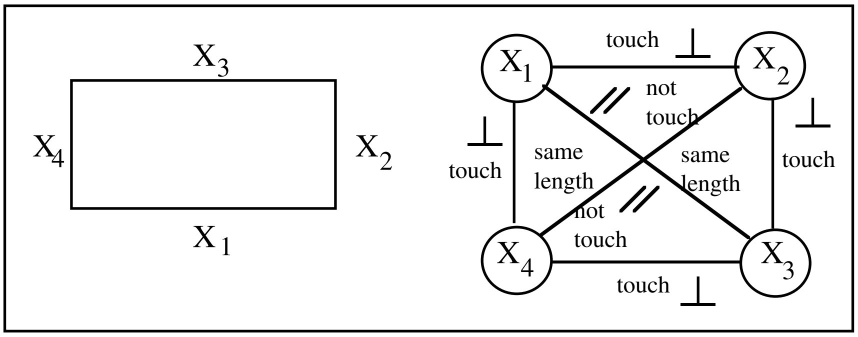

Figure 5: Constraint-based model of a rectangle

We first describe how to solve the problem by means of a classical CSP approach. Then, we present an alternative solution based on the ICSP model proposed in section 2.

In a classical CSP model all the segments should be collected as a first step. All the segments in figure 4 can be collected in the following form: they have an associated name, the coordinates (x1, y1, x2, y2 for the two staring and ending points), the length and the angle with the horizontal axes. These characteristics are sufficient for checking the constraints on the rectangle.

In the following, we will refer to a segment from a to b as (a, b). The 41 segments of figure 4 represent the domain of variables X_1, X_2, X_3 and X_4 (corresponding to the four edges of the rectangle in figure 5). Let us consider the result of an arc-consistency propagation method. A constraint network is arc-consistent if and only if for each value of a variable domain, there exist at least one value in each other variable domain which is consistent with constraints. We will refer to one domain propagation only since, in this case, all constraints are symmetric. Therefore, the four variables have always the same domain during the computation and before the labeling step.

Consider segment $(1,3)$. This segment is not consistent with constraints since there does not exists in other domains a segment with the same length. Therefore, it is deleted from domain D_1. The segment $(3,8)$ is, instead, consistent since it satisfies all the constraints. Note that both constraints $touch(X_1, X_2)$ and $touch(X_4, X_1)$ are satisfied by the same segment ($(1,2)$) in the domain of X_2 and X_4. The arc-consistency is an iterative algorithm which stops when the network has reached the *quiescence*, i.e., no more propagation is possible. After the first arc-consistency iteration, the variable domains contain only segments $(3,8)$, $(1,2)$, $(9,10)$, $(10,11)$, $(11,12)$, $(9,12)$, $(16,18)$, $(14,15)$, $(31,38)$, $(26,29)$, $(27,28)$, $(28,29)$, $(26,27)$, $(33,37)$, $(33,32)$, $(34,36)$, $(35,36)$, $(26,25)$. At a second iteration, segments $(16,18)$, $(14,15)$ are no longer supported by $(14,16)$ which has been deleted at the first step, segments $(3,8)$, $(1,2)$ are no longer supported by $(1,3)$ and segment $(31,38)$ is no longer supported by $(26,38)$. Remaining segments are $(9,10)$, $(10,11)$, $(11,12)$, $(9,12)$, $(26,29)$, $(27,28)$, $(28,29)$, $(26,27)$, $(33,37)$, $(33,32)$, $(34,36)$, $(35,36)$, $(26,25)$.

Note that not every segment in the remaining domain is part of a consistent rectangle since arc-consistency is not a complete propagation algorithm. In particular, segments $(33,37)$, $(33,32)$, $(34,36)$, $(35,36)$, $(26,25)$ which are not part of any rectangle could be deleted by a more powerful constraint propagation algorithm (e.g., path-consistency). Now a labeling step starts instantiating variable X_1 to $(9,10)$. Constraint propagation now reduces domains of other variables to $X_2 :: [(9,12),(10,11)]$, $X_4 :: [(9,12),(10,11)]$ and X_3 can be instantiated to $(12,11)$.

Consider now the approach based on the ICSP. The main advantage is that we do not need to have all the information about segments in order to start the propagation. Knowledge acquisition of all the information from the visual scene is usually very expensive. Therefore, we suppose that at the beginning all variable domains are undefined, i.e., the defined part is empty. Now we can ask the visual system to give us the first information about segments. This first query to the visual system can be guided in some sense if we have unary constraints on variables. For example suppose we have a color constraint on one of the segments, e.g., $red(X_1)$, we can guide the visual system to acquire first a red segment thus concentrating on "red zones". In general, the first acquisitions can be guided by means of heuristics on the problem, say first we acquire segments belonging to closed figures. In this case, we avoid this guide since we do not have neither unary constraints nor heuristics. Therefore, the system gives us the first segment for X_1, say $(1,3)$. This domain is posted in the domain of X_1 and not in other domains in order to remove problem symmetries [13]. Therefore, the domain of X_1 is $D_1 = [(1,3), X'_1]$ where X'_1 represents the undefined part of X_1. Now, since all other variables have only an undefined domain, $(1,3)$ is consistent with constraints. However, new constraints are generated in order to check the consistency of future acquired information and to guide the search. In particular, all the constraints containing variable X_1 are "splitted" into two parts: the first results from the propagation of the undefined part of all other variables with the defined part of X_1:

$touch((1,3),X_2)$, $touch(X_4,(1,3))$, $no\_touch((1,3),X_3)$, $same\_length((1,3),X_3)$,
$perpendicular((1,3),X_2)$, $perpendicular(X_4,(1,3))$, $parallel((1,3),X_3)$

the second is derived from the propagation of the undefined part of all other variables with the undefined part of X_1:

$touch(X'_1,X_2)$, $touch(X_4,X'_1)$, $no\_touch(X'_1,X_3)$, $same\_length(X'_1,X_3)$,
$perpendicular(X'_1,X_2)$, $perpendicular(X_4,X'_1)$, $parallel(X'_1,X_3)$.

The first part is used in order to guide the knowledge acquisition from the visual system in a *goal driven* way. In fact, for variable X_2, we are looking for a segment that touches and is perpendicular to $(1,3)$. Such a segment exists and is, for example,

$(1,2)$. Variable X_2 ranges now on the following domain: $D_2 = [(1,2), X_2']$. Again all the constraints involving X_2 can be "splitted" by constraint propagation in two parts. Note that constraints involving the defined part of X_1 and the defined part of X_2 are solved and can be ruled out. The new constraints derived from the defined part of X_2 and the undefined part of other variables are the following:

$touch((1,2), X_3)$, $no\_touch((1,2), X_4)$, $same\_length((1,2), X_4)$,
$perpendicular((1,2), X_3)$, $parallel((1,2), X_4)$.

Constraints involving only indefinite parts are:

$touch(X_1', X_2')$, $touch(X_2', X_3)$, $no\_touch(X_2', X_4)$, $same\_length(X_2', X_4)$,
$perpendicular(X_1', X_2')$, $perpendicular(X_2', X_3)$, $parallel(X_2', X_4)$.

Now, knowledge acquisition for X_3 can be driven from constraints and fails since there does not exists any segment for X_3 which touches and is perpendicular to $(1,2)$ and does not touch and is parallel to $(1,3)$.

A successful computation can be started when the system finds one of the segments belonging to a rectangle, such as for example $(12,9)$. The computation can be guided by means of constraints and the attention focuses only on the part of the scene containing the rectangle.

The main advantage of the approach is that in the average case, we acquire from the visual system a fewer number of segments since we guide the extraction by means of constraints. Note also that this kind of acquisition corresponds to an a-priori application of consistency techniques since the visual system provides only consistent values with constraints. Moreover, the applied consistency is stronger than arc-consistency since we query the visual system with different constraints on many variables. For example, in the above mentioned example, when we query the system for the acquisition of a value for X_3, we ask for a segment which both touches and is perpendicular to $(1,2)$ and does not touch and is parallel to $(1,3)$. This corresponds to a path-consistency since the consistency check involves three variables. In fact, in the case of arc-consistent segments $(33,37)$, $(33,32)$, $(34,36)$, $(35,36)$, suppose we have already acquired value $(33,37)$ for X_1 and $(33,32)$ for X_2. The acquisition of a value for X_3 fails since there does not exists any segment that does not touch $(33,37)$ but touches $(33,32)$.

5 Conclusions and Future Works

We have presented a model for interactive CSP which can be used when knowledge on the domain is not completely defined at the beginning of the computation but can be acquired on demand by a low level sensor system. We have applied our framework to a case study of object recognition and identification in a visual system. Objects are modeled by means of constraints and constraint propagation is the general-purpose tool for detecting a solution.

The propagation has mainly two purposes: first, to reduce the search space by pruning those assignments which cannot appear in any consistent solution. Second, it is used in order to guide the search by generating new constraints at each step. Therefore, information can be acquired on-demand from the low level system thus reducing computational expensive low level tasks. Moreover, newly acquired information does not need to be checked against the whole network but only against newly generated constraints.

A very effective programming paradigm for solving CSP [14, 15] is Constraint Logic Programming on finite domains (CLP(FD)). CLP (see [5, 6]) is a class of programming languages based on the Logic Programming (LP) paradigm [8, 7]. We are currently extending the CLP framework in order to cope with the ICSP model thus leading to an expressive and powerful language for visual applications.

We have applied our framework to a case study of object recognition and identification in a visual system. Object are modeled by means of local visual features and their constraints. This approach is in accordance with some proposals exploiting relational graphs and constrained search. A famous example is BONSAI which uses interpretation tree search with pruning on unary and binary predicates for 3D object recognition [1]. Other examples can be found in the survey [4]. Differently from these proposals ICSP is able for reducing computational load of the visual system by guiding the image processing and feature extraction tasks and performing a selective visual search.

Future works in the field of visual object recognition concern the extension of the system for integrating more complex visual features and for modeling the visual target in terms of hierarchical ICSP. A proposal of a Hierarchical Constraint-based Vision Architecture is described in detail in [2]. Moreover, the same approach should be used for 3D object recognition using different image sources (cameras, laser finders and so on) and for data and sensor fusion applications. All these contexts, in facts, may be improved in performance and efficacy by a profitable interaction between the high level module (the matching module) and the lower level data processing and by an efficient use of constraints for modeling also relations between elements of the target.

References

[1] R.T. Chin, C.R. Dyer, "Model Based Recognition in Robot Vision" in *ACM computer surveys*, vol 18 n.1 67-108 (1986).

[2] R. Cucchiara, E. Lamma, P.Mello, M.Milano, M. Piccardi, "Object Recognition on a Constraint-based Vision Architecture"submitted to CAMP97.

[3] R.Cucchiara, E.Lamma, P.Mello, M.Milano, " An Interactive Constraint Based System for Selective Attention in Visual Search", DEIS Tech. Rep. 97-001, 1997.

[4] P.J. Flynn, A.K.Jain, "BONSAI : 3d object recognition using constrained search", in *Proc. of IEEE Third Conference of Computer Vision*, pp. 263-267 (1990).

[5] J.Jaffar , J.L.Lassez, "Constraint Logic Programming", in *Proceedings of the Conference on Principle of Programming Languages*, Munich 1987.

[6] J.Jaffar, M.J.Maher, "Constraint Logic Programming: a Survey", in *Journal of Logic Programming, Special Issue on 10 years of Logic Programming*, 1994.

[7] J.W.Lloyd, "Foundation of Logic Programming", Second Extended Edition, Springer-Verlag 1987.

[8] R.Kowalski, "Logic for Problem Solving", North-Holland,1979.

[9] V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey", in *AI Magazine*, vol. 13, 1992, pp. 32–44.

[10] A.K. Mackworth, "Consistency in Networks of Relations", in *Artificial Intelligence*, vol.8, 1977, pp. 99–118.

[11] R.Mohr, T.C.Henderson, "Arc and Path Consistency Revisited", in *Artificial Intelligence*, vol.28, 1986, pp. 225–233.

[12] U.Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing", in *Information Sciences*, vol.7, 1974, pp. 95–132.

[13] J.F. Puget, "On the Satisfiability of Symmetrical Constrained Satisfaction Problems", Tech. Report ILOG Headquarters, 1993.

[14] P.Van Hentenryck, "Constraint Satisfaction in Logic Programming", MIT Press, 1989.

[15] P.Van Hentenryck, H.Simonis, M.Dincbas, "Constraint Satisfaction using Constraint Logic Programming", *Tech. Report* CS-91-62, Dept. of Computer Science, Brown University, November 1991.

[16] D. Vernon, "Machine Vision: Automated Visual Inspection and Robot Vision" Prentice Hall, 1991.