Extending CLP(FD) with Interactive Data Acquisition for 3D Visual Object Recognition

M.Gavanelli, E. Lamma, P.Mello, M. Milano
DEIS, Univ. Bologna,
Viale Risorgimento 2 I-40136 Bologna, Italy.
E-mail: {elamma, mgavanelli,pmello,mmilano}@deis.unibo.it

M.Piccardi
Dip. Ingegneria, Univ. Ferrara,
Via Saragat, 1 - 44100 Ferrara, Italy.
E-mail:mpiccardi@ing.unife.it

R.Cucchiara
Univ. Modena,
Via Campi, 312 - 41100 Modena, Italy.
E-mail: rita.cucchiara@unimo.it

Abstract

This paper addresses the 3D object recognition problem modelled as a Constraint Satisfaction Problem. In this setting, each object view can be modelled as a constraint graph where nodes are object parts and constraints are topological and geometrical relationships among them. By modelling the problem as a CSP, we can recognize an object when all constraints are satisfied by exploiting results from the CSP field. However, in classical CSPs variable domains have to be statically defined at the beginning of the constraint propagation process. Thus, not only feature acquisition should be completed before the constraint solving process starts, but all image features should be extracted even if not belonging to significant image parts. In visual applications, this requirement turns out to be inefficient since visual features acquisition is a very time consuming task. We present an *Interactive Constraint Satisfaction* model for problems where variable domains may not be completely known at the beginning of the computation, and can be interactively acquired during the computational process only when needed (on demand). The constraint propagation process works on already known domain values and adds new constraints on unknown domain parts. These new constraints can be used to incrementally process new information without restarting the constraint propagation process from scratch each time new information is available. In addition, these constraints can guide the feature acquisition process, thus focussing attention on significant image parts. We present the Interactive CSP model and a propagation algorithm for it. We propose an implementation of the framework in Constraint Logic Programming on Finite Domains, CLP(FD).

1 INTRODUCTION

Constraint Satisfaction systems provide a simple but powerful framework for solving a variety of Artificial Intelligence (AI) problems. Constraint Satisfaction Problems (CSP, for short in the following) are defined on a finite set of variables each ranging on a (numerical or symbolic) domain and a set of constraints. We assume variable domains to be finite. A solution to a CSP is an assignment of values to variables which satisfies the constraints. Propagation algorithms [10] (e.g., forward checking, look ahead etc.)

have been proposed based on the active use of constraints during the search process. The idea is to remove, during search, by means of constraint propagation, combination of assignments which cannot appear in any consistent solution.

A CSP-based inference engine has been successfully used in many applications. In this paper we focus on a 3D object recognition application where a low level system provides a large amount of (constrained) data to be processed, i.e., visual features of *objects* in an image. Several examples of CSP-based reasoning systems have been proposed for object recognition (see for instance [15, 17, 24, 18]). Visual applications usually require some form of interaction between a low level module (image processor and feature extractor) providing (constrained) data, and a constraint solver. In classical CSPs, variable domains have to be completely known before the constraint propagation process starts. Data acquisition and its processing are sequentially performed thus leading to an inefficient behaviour of the whole system especially when the data acquisition process is computationally expensive. For example, a CSP module interacting with a low level visual system should first acquire all the visual features in the scene (thus requiring the low level system to process the whole image) in order to create variable domains, and then start the constraint propagation process for object recognition. In a vision system the data extraction phase is very time-consuming: usually it is much more computationally expensive than the constraint solving phase.

We argue that interleaving the acquisition of domain values and their processing could greatly increase the performances of the object recognition process. Domain value acquisition can be performed *on demand* only when values are effectively needed. This approach can be seen as a kind of *lazy domain evaluation*. Lazy evaluation [9] is known as a parameter evaluation mechanism which avoids a computation if its resulting value will never be used. Similarly, we avoid to consider values for constraint propagation if they are not needed. This idea has been already exploited in the field of constraint satisfaction in [2, 22] where as soon as one consistent value is found, the propagation stops in order to perform a minimal number of constraint checks.

Furthermore, a fundamental point which can be exploited in our framework is that the acquisition process can be guided by constraints, called *interactive constraints*, thus leading to retrieve only consistent values and minimizing useless data acquisitions. In a visual system, this feature allows to focus the attention of the feature extraction module on a restricted part of the scene, by propagating spatial and topological constraints; second, to constrain the feature space and assist the computation of visual features. Therefore, a CSP system should be able not only to prune the data set after it has been computed, but also to guide the data acquisition process.

For this purpose, we present an *Interactive Constraint Satisfaction* [21] model where domains can be partially known when the constraint satisfaction process starts and are dynamically acquired during the computation. We apply this framework to a 3D object recognition application.

A powerful language for modelling and solving CSPs is Constraint Logic Programming (CLP) [14]. In recent years CLP has been successfully used for solving hard combinatorial problems [3, 4, 10] modeled as Constraint Satisfaction Problems (CSPs). In this paper, we focus on Constraint Logic Programming on finite domains, hereinafter referred to as CLP(FD). We have implemented the interactive framework on top of the finite domain library of the ECL^{i}PS^e language [6]. The extension of the constraint solver is aimed at coping with interactive constraints and partially or completely un-

known domains.

The paper is organized as follows: in section 2 we show how to model the object recognition problem as a CSP. In section 3 the interactive CSP framework is described along with the interactive version of the Forward Checking algorithm presented in section 4. The extension of the ECL^{i}PS^e CLP(FD) library is sketched in section 5. Finally, section 6 shows the application of the framework to a 3D object recognition problem and experimental results. A discussion and a mention of future work conclude the paper.

2 MODELLING OBJECTS THROUGH CONSTRAINTS

3D object recogition involves two steps: object model building and model solving. We adopt a constraint based approach to object recognition. Thus, objects can be modeled by means of constraints, and recognized by means of constraint satisfaction.

In this section, we focus on model building by defining a general and reliable object model, in order to result possibly invariant to roto-translation in 3D space, to distance object-camera and other environment factors. As a consequence, the object model cannot be limited to a matching of measured features to absolute values previously defined, but should be based on geometric and topological relationships between features [5, 23]. 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).

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

As an example, if we want to model the L-shaped object shown in Figure 2.a we shall have a node corresponding to each surface (named respectively X_1,..., X_6) and will impose the constraints depicted in the graph in the same figure This graph model corresponds to a single view of the object.

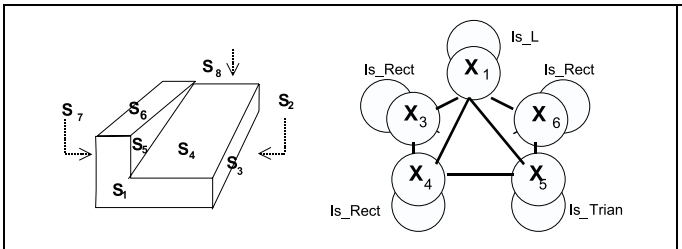

Fig 1.a an L-shaped object and its view-based model

```
object_L(X1,X3,X4,X5,X6) :-
        is_shape_L(X1),
        is_rectangle(X3),
        is_rectangle(X4),
        is_triangle(X5),
        is_rectangle(X6),
        touch(X1,X3),touch(X1,X4),
        touch(X1,X5),touch(X1,X6),
        touch(X3,X4),touch(X4,X5),
        touch(X5,X6),1
        is_normal(X1,X3). . .
        is_parallel(X3,X5)…
```

Fig.1.b part of the model

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.

Note that in 3D object modelling, we have the problem of multiple views and occlusion. We are developing an observer-independent model for 3D object, by using a single object graph and exploiting hiding constraints for non visible parts. This aspect will be briefly described in section 2.1, and it is subject of current research.

Suppose the vision system provides all the information about the surfaces in the scene. Therefore, each variable of the object model ranges on a domain containing all the surfaces. We can perform a straightforward but inefficient backtracking algorithm in order to find a solution. Since we model objects by means of constraints, we can exploit constraint propagation in order to prune the search space. Therefore, we intertwine a propagation process that removes combinations of assignments which cannot appear in any consistent solution, i.e., which cannot form an L-shaped object, and a labeling strategy that assigns to each variable a value (a possible segment). If a solution is found, we have identified the searched object in the scene, i.e., values for variables (surfaces composing the model) which are consistent with constraints.

2.1 Multiple view graph

By means of constraint satisfaction we can recognize a set of surfaces extracted from the image as a specific view of the 3D object. Since objects should be recognized from each point of view, a possible approach is to define a set of view-based constraint graphs, one for each object pose, and to find a constraint satisfaction solution for at least one of them. This technique is not particularly efficient; moreover the problem of defining the minimum set of reliable view models arises. Conversely, we propose an object-centered topological model, independent from the observer point of view. The model represents all surfaces forming the object as nodes and surface relations as constraints. The corresponding graph cannot merely be satisfied by the surfaces extracted from the image, since, for each point of view, only a subset of surfaces are visible.

Instead of modifying the constraint graph by relaxing some constraints on the basis of the observer point of view, we satisfy the constraints between visible surfaces and add to the object model *virtual surfaces* and *visual constraints* representing object part that are not visible from that point of view. The object model can be defined as a Visual Constraint graph, called *VC-Graph*.

A VC-Graph is a graph composed by $X_1, \ldots, X_n$ nodes representing all object surfaces. Each node has an associated domain $D_1, \ldots, D_n$ representing surfaces which can be assigned to the node. Arcs of the VC-graph are constraints between surfaces. In order to allow the match between the modeled object and surfaces retrieved from the image, we introduce virtual surfaces, which should be inserted in the domain of variables representing not visible surfaces. Thus, D_i contains $S_1, \ldots, S_m$ the set of real surfaces and $S'_1, \ldots, S'_m$ the set of virtual surfaces, opposite to real surfaces, and that are not visible when the corresponding real surface is visible. As a consequence, we should define constraints representing the fact that two surfaces cannot be viewed at the same time and the fact that a surface could partially occlude another. We call these surface relations visual constraints.

More formally, *hide*(X_1,X_2) states that X_1 and X_2 cannot be visible at the same time (it arises when two surfaces have an opposite normal as S1 and S8 in Figure 1); *occlude*(X_1,X_2) means that X_1 may partially occlude X_2 (as happens for S6 and

S4 in Figure 1). This causes a relaxation of unary constraints on X_2 (in Figure1 *Is_Rectangle(S4)* is not satisfied if S4 is occluded by S6). For example, consider the L-shaped object of figure 1 . The corresponding VC-graph is depicted in Figure 1 (where both topological and visual constraints are indicated). Suppose our view of the object is the one in Figure 1a. The model is satisfied if we assign to variable X_1 the L-shaped surface S1, to X_3,X_4,X_5, and X_6 surfaces S3, S4, S5 and S7 respectively and to X_2, X_7 and X_2 the virtual surfaces S4', S3', and S1' that satisfy the visual hiding constraint. With the VC-graph we can completely describe an object from each point of view: each single-view constraint graph is a sub-graph of the VC-graph. The nodes of a single-view sub-graph represent visible surfaces and the correspondent nodes in VC-Graph match real surfaces; instead, nodes absent from the sub-graph are non-visible surfaces represented by virtual surfaces.

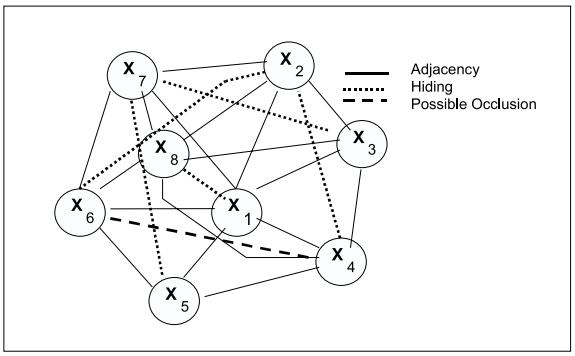

Figure 1: **VC-Graph of the *L-shaped* object**

3 INTERACTIVE CSP

In the previous section, we have presented how to model objects by means of constraints. Thus, we can use constraint satisfaction techniques in order to recognize an object. The problem with this kind of representation concerns the fact that we have to acquire from the vision system all the information about surfaces in the scene before processing them. The feature extraction, specially in 3D visual applications, is a computationally expensive task if compared with the constraint satisfaction process. Constraints expressing locality (such as *touch*) can be exploited in order to focus the whole system's attention on a limited image part, thus avoiding feature extraction on most of the image surface. Selective attention is a fundamental skill of biologic vision systems, since every image contains huge quantities of data and processing all of them would be unwise. We propose an alternative approach, based on what we call Interactive Constraint Satisfaction framework, that intertwines the vision feature acquisition with the constraint satisfaction process thus reducing the feature extraction computational cost.

We first start by giving some preliminaries on CSPs. A CSP is defined on a set of variables $X_1, \ldots, X_n$ ranging respectively on finite domains $D_1, \ldots, D_n$. A constraint $c(X_{i_1}, \ldots, X_{i_k})$ defines a subset of the cartesian product of $D_{i_1}, \ldots, D_{i_k}$, i.e., a set of configurations of assignments which may appear in a consistent solution.

In this paper, we focus on binary CSPs. A binary CSP can be represented by means of a *constraint network* where each node is a variable and arcs are constraints.

A standard constraint solver 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 data acquisition process which lasts during the whole computational process. Therefore, we have to change the classical CSP model, and allow the propagation algorithm to work on partially known domains.

To this purpose, we define an *Interactive CSP* (*ICSP*) model [21] which has to cope with incomplete domains. Domains can be partially known 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. The strength of this approach concerns the fact that the ICSP system can guide data acquisition by means of constraints, and incrementally process new information without restarting a constraint propagation process from scratch each time new data are available.

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

Definition 1 An *Interactive CSP* (ICSP) is defined on a set of variables $\{X_1, \ldots, X_n\}$ each ranging on a partially known domain $\{D_1, \ldots, D_n\}$ where each $D_i = \{Known_i \cup UnKnown_i\}$. $Known_i$ represents the known part, i.e., the set of available values, while $UnKnown_i$ represents information which is not yet available, i.e., the set of values will be retrieved in the future. Both $Known_i$ and $UnKnown_i$ can be possibly empty[1]. Also, for each i, $Known_i \cap UnKnown_i = \emptyset$. An *interactive constraint* among variables defines a (possibly partially known) subset of the cartesian product of variable domains. A solution to the ICSP is, as for standard CSPs, an assignment of values to variables which is consistent with constraints.

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 most general case, both X_i and X_j domains contain a non empty known and unknown part. In order to propagate the constraint $c(X_i, X_j)$ we have to propagate four kinds of constraints, respectively between the known and unknown parts of variable domains:

$c(Known_i, Known_j)$, $c(Known_i, UnKnown_j)$

$c(UnKnown_i, Known_j)$, $c(UnKnown_i, UnKnown_j)$

and collect the propagation results[2]. While the constraint check on known parts can be performed as usual, the check on at least one unknown part requires a data acquisition in order to acquire new information. In addition, the data acquisition can be guided by means of interactive constraints in the sense that data acquisition retrieves values which are consistent with constraints.

Let us see a simple example in the domain of integers. Consider two domain variables X and Y ranging respectively on the following domains $\{1, 3\} \cup X1$ and $\{-6, 4\} \cup Y1$. The known part of the domain of X contains two values $\{1, 3\}$, while its unknown part, $X1$, represents not yet available values for X. Similarly, the known

[1] When both are empty an inconsistency arises.

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

and unknown part of the domains of Y are $\{-6, 4\}$ and $Y1$ respectively. A constraint between X and Y, say $X \leq Y$, is satisfied if and only if variables X and Y assume consistent values in their known part (e.g., $X = 1$ and $Y = 4$ or $X = 3$ and $Y = 4$) or if the data acquisition process provides consistent values for the variables.

We now present intuitively how the constraint propagation algorithm works on the above example. Then, we sketch and explain the pseudo code that implements it. Constraint propagation between the known part of the constraints can be performed as usual. Note that no values however can be removed if both variable domains are partially known since they could be supported by future acquisition. The ICSP computation starts with a labeling step, as in CSP search. Instead, the consistency check between values 1 and 3 for variable X and the unknown part of variable Y, i.e., $Y1$, calls for a data acquisition that is aimed to collect at least one value for Y which is consistent with the known part of the domain of X. In other words, the system collects for variable Y at least one value which is greater or equal to 1 or 3. This is equivalent to pose a constraint on the unknown part of Y, e.g., $1 \leq Y1 \vee 3 \leq Y1$ and guide the data acquisition by means of these constraints by asking the low level system for those values that satisfy the above mentioned constraints. Similarly, the constraint propagation acts on the unknown part of X and the known part of Y. Finally, the constraint between unknown parts will check new acquired values as soon as they will be available.

In Figure 2 we sketch the pseudo-code for binary constraint propagation in the interactive framework. The procedure propagate_constraints works on a constraint c in order to reduce variable domains D. We have to distinguish three cases: the first case concerns the classical constraint propagation when both variables present a non-empty known part (procedure propagate_known). The second case regards the propagation between an unknown part and a known one (procedure propagate_partially_known). In this case, an acquisition should be performed which can be guided by the constraint itself on the basis of values contained in the known part of the domain. Procedure propagate_partially_known queries the low level system in order to retrieve for variable Y only values consistent with the known part of X. The last case concerns a constraint propagation on two unknown parts (procedure propagate_unknown). In this case the constraint is delayed since the knowledge acquisition could not be guided by any known value. The delayed constraint on $UnKnown$ domain parts guarantees that future acquisitions will be consistent with constraints.

Note that constraints on at least one unknown part intensionally represent potential solutions on not yet acquired data.

An interesting point concerns data acquisition. Data acquisition is guided by constraints containing one variable (we will call it *guiding variable*) which has a partially or fully known domain and one variable (we will call it *guided variable*) whose domain is completely unknown. We have to answer to basically two questions:

- how many values of the guiding variable domain to use in order to guide data acquisition;
- how many values of the guided variable to acquire;

Basically, a lazy approach tends to minimize data acquisitions, since it's often the most expensive task (in many problems it involves hard signal processing). Thus, the data

```
procedure propagate_constraints(D,c(A, B))
begin
    D_A = Known_A ∪ UnKnown_A;
    D_B = Known_B ∪ UnKnown_B;
    propagate_known(Known_A,Known_B);
    propagate_partially_known(Known_A,UnKnown_B);
    propagate_partially_known(Known_B,UnKnown_A);
    propagate_unknown(UnKnown_A,UnKnown_B);
end

procedure propagate_partially_known(Known_X,UnKnown_Y);
begin
    guided_acquisition(c(Known_X, UnKnown_Y));
end;

procedure propagate_unknown(UnKnown_X,UnKnown_Y);
begin
    delay(c(UnKnown_X, UnKnown_Y));
end;
```

Figure 2: **The interactive constraint propagation**

acquisition is stopped as soon as one consistent value has been retrieved for the guided variable and is guided by means of one value in the domain of the guiding variable. On the contrary, an eager approach collects all consistent values for the guided variable and guides data acquisition by means of all known values.

The best choice depends on the application we have to solve. In the field of object recognition, domain values come from an image processing and a feature extraction system, both of which are computationally expensive tasks. For 3D objects images are range (or depth) images and extracted features are surfaces. This activity greatly benefits of the exploitation of locality criteria that restrict the image part to be processed. As soon as a restricted part of an image is selected, retrieving one or all features belonging to that image part is computationally comparable with the retrieval of only one feature. Thus, in the visual application, we decided to implement a search algorithm capable of exploiting this property. Data acquisition will be performed by means of interactive constraints with only one value for the guiding variable (i.e. the guiding variable will be instantiated) and by retrieving all the consistent values for the guided variable. This idea leads to implement a Forward Checking-like algorithm, and is widely explained in the next section. It's worth noting that the ICSP framework comprises the classical CSP environment; so we are not forced to implement every constraint the interactive way. Some constraints are useful for interaction; e.g. the constraints expressing locality in the vision system. Other constraints may be used only for pruning the search space, as in the CSP case. When a non-interactive constraint has to deal with incomplete knowledge, it can simply suspend and wait for the interactive constraints to acquire the needed information items.

4 INTERACTIVE FORWARD CHECKING

One of the well known and widely accepted propagation algorithms for solving CSPs is the forward checking (FC) technique [8]. The FC algorithm intertwines a *labeling* step, where a variable X is instantiated to a value v in its domain, and a *propagation* step where domain variables linked with X by means of constraints are checked in order to remove values which are not compatible with v.

In our framework, we have to cope with partially known domains. Therefore, the operational behaviour of the FC algorithm should be changed accordingly. Intuitively, the first *labeling* step instantiates a variable X to a value v belonging to the known part of the domain if any. Otherwise, a data acquisition is performed retrieving a value v which is successively assigned to X. The *propagation* step considers domain variables $X_1, \ldots, X_k$ linked with X by means of constraints. This step removes from the known part of $X_1, \ldots, X_k$ domain those values which are not consistent with v, and (eventually) acquires consistent values for the unknown part[3].

Note that in the algorithm presented in Figure 2, only the first two procedures (propagate_known and the first propagate_partially_known) are performed since for the forward checking strategy one variable is always instantiated (thus completely known).

Let us consider an example in the field of 2D shape recognition. If we want to model a rectangle 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:

$allDifferent([X_1, X_2, X_3, X_4])$,
$touch(X_1, X_2)$, $touch(X_2, X_3)$, $touch(X_3, X_4)$, $touch(X_4, X_1)$,
$no\_touch(X_2, X_4)$, $no\_touch(X_1, X_3)$, $same\_lenght(X_2, X_4)$, $same\_lenght(X_1, X_3)$,
$perpendicular(X_1, X_2)$, $perpendicular(X_2, X_3)$, $perpendicular(X_3, X_4)$,
$perpendicular(X_4, X_1)$, $parallel(X_2, X_4)$, $parallel(X_1, X_3)$[4].

We consider only the *touch* constraint as interactive, because it expresses locality and allows selective attention. The task is to recognize the first rectangle in the scene depicted in Figure 3.a. Variable domains are segments retrieved from the image.

Initially, the variable domains are completely unknown:

$X_1 :: UnKnown_1$, $X_2 :: UnKnown_2$, $X_3 :: UnKnown_3$, $X_4 :: UnKnown_4$.

Figure 3.b shows an IFC computation until success is reached. The IFC algorithm starts with a labeling step on variable X_1. Since the domain of X_1 does not contain any acquired value, the labeling step performs a feature acquisition process (possibly guided by unary constraints on X_1). A segment a in the image is retrieved and assigned to X_1, i.e., $X_1 = a$.

Now, the plain FC algorithm collects all the variables linked to X_1 by means of constraints and removes from their domains all values which are inconsistent with a. Since the non interactive algorithm requires all the segments to be in the domains at the beginning of the computation, it has to check value a against all the segments

[3]This data acquisition is performed or not on the basis of a eager or lazy acquisition policy.

[4]The constraints *perpendicular*, *parallel* and *same_length* have an obvious semantics. The constraint *touch* (respectively *no_touch*) means that one ending point of the first segment must touch an ending point of the second one (resp. the ending points must not touch each other).

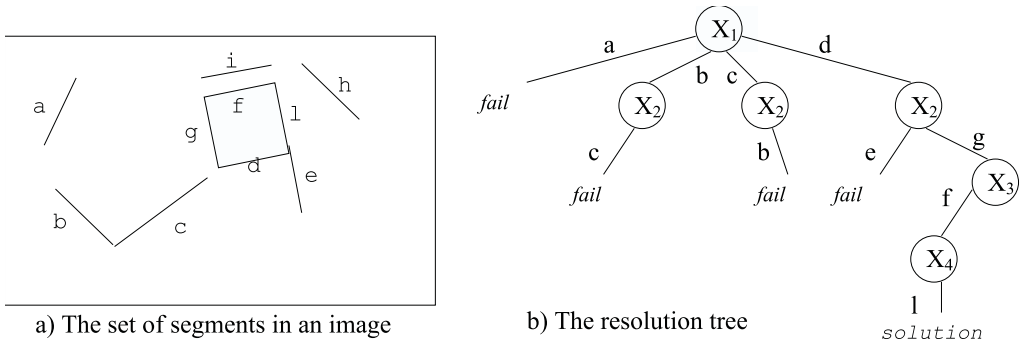

Figure 3: **Example of ICSP computation**

in X_2's domain (i.e. $\{a, b, \ldots, l\}$), before finding that no value is consistent. The IFC, instead, does not know the values in the domains of the next variables, so it requests values consistent with the interactive constraint $touch(X_1, X_2)$. Note that the feature acquisition process is guided by the above mentioned constraint exploiting the locality criteria embedded in the constraint *touch*. Therefore, the underlying visual system looks for segments touching a thus focusing attention around it. Only segment a touches itself, so the domain of variable X_2 becomes $\{a\}$; this value is deleted by the non interactive constraints $allDifferent$ and an inconsistency is raised. In this case, the IFC algorithm performs only *one* constraint check to detect inconsistency.

Computation succeeds when X_1 is assigned the value d, as shown in Figure 3.b. As concerns the number of values to be retrieved, in the forward checking algorithm we can decide to acquire all values consistent with the currently instantiated variable or only one value. In the first case, the domains of X_2 and X_4 become completely known and they are the same as the ones resulting from the classical forward checking algorithm after the instantiation of X_1 to d. In the second case, the domains of X_2 and X_4 are left partially known and constraints are imposed on the variables representing the unknown parts. This second approach is similar to performing a minimal forward checking algorithm [2] and will be briefly discussed in section 4.1.

In a visual system environment, we decide to collect all values for variables since feature extraction exploiting locality criteria is almost independent from the number of features extracted. Thus, the system collects four segments d, e, g and l[5] which are put in the domains of X_2 and X_4.

The unknown parts of X_2 and X_4 are now completely specified and the domain of X_2 and X_4 contain all values which touch d. Value d is removed from both these domains by non interactive constraints. Again, the non interactive FC has to check the value d assigned to X_1 against *all* the values in the other domains (i.e. all the segment in the image) to get to the same situation. The interactive, instead, uses the constraint *touch* to guide acquisition and the other constraints have only to check the *four* acquired values for consistency.

[5]Note that constraints describing the rectangle are symmetric. Symmetries should be avoided in a constraint satisfaction procedure [19]. Therefore, in practice, we do not put all the acquired segments in both domains. In the example, however, for the sake of simplicity, we omit the treatment of symmetries.

The next labeling phase assigns e to X_2; this choice leads to failure and the next step sets $X_2 = g$. A second constraint propagation process starts by considering all variables involved in a constraint with X_2. Consider the constraint, described in section 2, between X_2 and X_3 stating that the two variables represent touching segments, and a constraint with X_4 stating that the two variables should have the same length. The first constraint propagation process results in a feature acquisition, collecting all values that touch X_2, i.e. d, f and g. Note that, again, feature acquisition is focussed around g and does not need to consider the whole image. These segments are put in the (known part of the) domain of X_3 while the unknown part is deleted. The propagation step between X_2 and X_4 is the usual forward checking constraint propagation since the domain of X_4 is completely known. The FC algorithm continues the labeling and the constraint propagation process as usual since all the domains now are known.

The purpose of the interactive framework is to force the low level system to retrieve a number of segments which is significantly smaller than those retrieved by a non-interactive system which first collects all segments in an image and then starts the constraint propagation process. The number of extractions must be kept as small as possible, since the extraction process is the most expensive task: usually the CSP solving process employs a negligible computation time w.r.t the feature extraction time. In order to avoid re-extraction of formerly acquired values (due, e.g., to backtracking), each extracted element is stored in memory after interaction; in successive computation, every segment will first be searched for in memory and then requested to the low-level system. Note that the amount of memory required for this structure is proportional to the number of elements in a domain, which is negligible with respect to the structures needed by the FC algorithm [8].

In Figure 4, we have sketched the basic Interactive FC algorithm. It first selects a variable Var to be instantiated, then it performs an interactive labeling (interactive_label). This procedure takes a value in the known part of the selected variable domain if it exists, otherwise it acquires one value for the selected variable (procedure acquisition_var). The procedure collect_constraints collects all constraints containing the selected (instantiated) variable. Then, for each collected constraint, the constraint propagation algorithm presented in Figure 2 starts. Note that being Var instantiated, the procedure propagate_constraints always finds the first variable completely known and either performs the usual propagation if other variables contained in $Constr$ are also known, or retrieves values with the propagate_partially_known procedure.

4.1 Interactive Minimal Forward Checking

When the constraint check is a computationally costly operation, the Forward Checking algorithm is outperformed by the Minimal Forward Cheking (MFC) technique [2]. FC checks for consistency every element in the forward connected domains allowing to detect failure in early stages of computation and to limit the number of backtracking moves needed during search. If a domain becomes empty during search, FC detects inconsistency and backtracks. Anyway, in order to avoid backtracking, only *one* consistent domain value is needed.

The MFC algorithm looks for a consistent value in every future connected domain; the other elements are left unchecked. If the only consistent value is removed, another candidate must be found and checked for consistency with respect to all the past con-

```
procedure IFC(C, D)
begin
for all variables do
        begin
        select_variable(Var,D),
        interactive_label(Var,D),
        collect_constraints(C,D,Var,C1),
        for each constraint Constr in C1 do
              propagate_constraints(D,Constr),
        end;
end.

procedure interactive_label(Var,D)
begin
        if unknown(Var)
              then acquisition_var(Var,D_Var),
        label(Var)
end;
```

Figure 4: **The incremental forward checking algorithm**

nected variables. This reduces the number of constraint checks performed on average during search, but requires complex data-structures managing and, in a CLP framework, the need for a modified backtracking mechanism. As pointed out in [2], this algorithm is not so good when constraint checks are not computationally expensive.

The Interactive Minimal Forward Checking keeps only one consistent value in the known part of domains, while the other values are left unextracted. If the consistent value is removed by constraint propagation, another value must be acquired. This technique is very useful when the extraction cost is proportional to the number of extracted elements. In a vision system, when exploiting locality criteria, the cost of extracting one or all surfaces close (touching) a given one is almost equivalent because extracting all consistent value is performed as a single operation. Since for the vision system extracting all the features in an area is nearly as costly as extracting only one feature, the IFC algorithm is much more promising.

5 EXTENSIONS OF THE CLP(FD) LIBRARY

We have implemented the Interactive Constraint Satisfaction framework on top of the finite domain library of the ECL^{i}PS^e language [6]. We have chosen to exploit Constraint Logic Programming [14] on finite domains (CLP(FD)) since it is a very effective programming paradigm for solving CSP [10, 11]. The CLP(FD) solver has been extended by means of user defined constraints in order to cope with partially known domains.

In particular, the implementation has concerned:

- an extension of the constraint solver in order to cope with interactive constraints and partially or completely unknown domains;

- some user-defined interactive constraints performing data acquisition (when working on unknown domain variables) and classical constraint propagation (when working on known domain variables).

As concerns the extension of the constraint solver, we have implemented a set of low-level predicates which allow the user to process partially known domain variables, modify them and write new constraint predicates. In particular, we have extended the following ECL^{i}PS^e predicates

- `dvar_remove_element` which removes an element from a variable domain (this predicate has been implemented also for removing the greatest or the smallest domain element);
- `dvar_update` which updates a domain variable;
- `dom_member` which selects a domain element.

In addition, a new predicate `specify_domain` has been defined in order to perform data acquisition and introduce in the domain new values acquired during the computation.

On the basis of this solver extension, we have implemented some interactive constraints performing the propagation explained in the previous sections. As an example, we sketch here the code of an interactive constraint `itouch` between two variables representing two segments.

```
itouch(S1,S2) :-
    (dvar_domain(S1,_)
      ->  (dvar_domain(S2,_)
            ->  touch(S1,S2)
            ;   itouch_propagate(S1,S2)  )
     ;    (dvar_domain(S2,_)
            ->  itouch_propagate(S2,S1)
           ;    make_suspension(itouch(S1,S2),4,Susp),
                insert_suspension((S1,S2),Susp,specify of dom_pd,dom_pd)
    )).
itouch_propagate(S1,S2) :-
    (nonvar(S1)
       -> dvar_domain(S1,Dom1), dom_to_list(Dom1,L1),
          acquisition(L1,L2),   specify_domain(S2,L2)
       ;  make_suspension(itouch(S1,S2),4,Susp),
          insert_suspension(S1,Susp,any of fd,fd),
          insert_suspension(S2,Susp,specify of dom_pd,dom_pd)  ).
```

If both variables are known, the non-interactive constraint `touch(S1,S2)` is called. Otherwise, if one of the two variables is unknown a data acquisition starts for the unknown variable on the basis of the known one. If both variables are unknown the constraint is suspended. Note that in the case of forward checking strategy, this latter case never happens since constraints are checked with one variable bound to a value.

The extension of the CLP(FD) solver does not affect the declarative semantics of CLP itself, but only its operational behaviour; in fact, the interactive constraints and the non interactive one hold for the same pairs of values.

6 3D OBJECT RECOGNITION VIA ICSP

In this section, we present the application of ICSP to the problem of 3D object recognition in images. The 3D object recognition is a hard open problem in computer vision since its solution requires a formal definition of different elements: the object model, the visual features used in the recognition process (i.e. the visual elements extracted from the images), and, finally, the inference engine exploited in the recognition process. An efficient choice of visual features for describing and recognizing 3D objects (especially CAD-made objects) consists in extracting from images the set of visible surfaces, and then computing some geometric and topological relations between them [7].

Using surfaces as features, a model M of a 3D object can be formalized as a pair (X, R) where $X = (X_1, \ldots, X_n)$ is the set of surfaces and R is a set of unary, binary or n-ary relations between surfaces. Using only unary and binary relations, a 3D object model can be described by means of a constraint graph where object surfaces are nodes (variables) and relations between surfaces are arcs. A graph representation of object models has been used in many different contexts of 2D shapes [18], and extended to the 3D scene recognition [17, 16].

Unary constraints refer to visual properties of single surfaces, such as colours, texture, or shape. Binary constraints are geometric relationships between surfaces: the most commonly used is the adjacency constraint that holds if two surfaces have a common edge. Graphs exploiting surface adjacency are known as Surface Adjacency Graph (SAG) [12].

Therefore, solving the 3D object recognition problem as a standard CSP can be seen as a result of the following process:

- extract from the image all surfaces $S_0, \ldots, S_m$;
- create variable domains $D_0, \ldots, D_n$ in the defined graph;
- find an assignment of surfaces to variables satisfying all unary and binary constraints.

In the recognition system, we used the shape propriety `is_a_xyz(`X_i`)` (where `xyz` is a specific shape: rectangle, trapezium, L_shape etc) as unary constraint, the adjacency constraint `touch(`X_i`,`X_j`)` as interactive, and geometric relations `is_normal(`X_i`,`X_j`)`, `is_parallel(`X_i`,`X_j`)` as (non interactive) constraints. Note that all these constraints are independent from rotations, translations and distance between the observer and the object.

Figure 2.a shows an L-shaped 3D object and its corresponding SAG. Figure 2.b contains the model written in ECL^{i}PS^e. For the sake of simplicity, we refer in this section to a single view graph, but we are currently testing the object-centered model described in section 2.1.

The CSP-based approach for object recognition suffers of a severe limitation in term of efficiency when applied of real vision applications: the knowledge acquisition step, that is the surface extraction, is a very time consuming step working on range images (i.e., images where each pixel contains the information of the 3D distance from the point of view).

The time spent for extracting all surfaces may result too heavy in many robotic or industrial applications: for example Figure 5 represents a 3D scene with several

partially overlapped objects; the problem is to find an object that satisfies the defined model (the L-shaped of Figure 2, i.e. the block in the middle-left part of the image). The first frame of Figure 5 shows the original image, the second frame the segmented image where all surfaces are segmented and the third the extracted information on edges and corners in order to check adjacency and geometric constraints. These operations take some minutes on standard workstations.

Figure 5: **The block7 image**

In this framework the adoption of an interactive version of CSP, as defined in previous sections allowed a strong performance improvement. The ICSP-based recognition executes the following steps:

- the constraint solver interacts with the low-level image processing system and calls for an initial unconstrained surface (S1);
- the interacting constraint propagation starts, and whenever a variable domain is not known, new variable values satisfying constraints are requested.

In particular, instead of using a function `get_all_surfaces()` at the beginning of CSP, the ICSP, during the procedure propagate_partially_known executes the function `get_surface_touching(Xi)` so that only adjacent surfaces are directly computed. This allows to improve the speed of the ICSP since variable domains turn to be smaller and, more important, the visual system focusses only on significant image parts. Thus, the guided interaction prevents the low-level system of acquiring many useless surfaces.

Several tests are performed on images of a specific data-base of range images: we have created a modified version of the Washington State University database [13] by assembling several images in order to obtain new images containing many different possibly overlapped objects. Results are obviously data dependent since the speed achieved in finding the first object satisfying the model depends on the position of the object in the image in function of the search direction.

Image	ICSP	CSP	Speedup
BLOCK_1 (320x320) *	136.60	279.66	2.04
BLOCK_2 (320x320) *	129.80	276.07	2.12
BLOCK_3 (320x320)	125.01	256.51	2.05
BLOCK_4 (320x320)	39.93	263.50	6.59
BLOCK_5 (320x320) *	156.50	309.90	1.98
BLOCK_6 (320x320)	34.89	301.43	8.63
BLOCK_7 (400x400)	178.77	442.51	2.47
BLOCK_8 (400x400)*	549.10	518.59	0.94
BLOCK_9 (400x400)	215.85	555.76	2.57

Table 1:**Computational results (* no L-shape object in the image)**

Results in Table 1 refer to a database of 9 images and describe the time (in seconds) spent for extracting the first L-shaped object: the CSP and the ICSP approaches are compared. Other results are described in [20].

Queries to the ICSP component have the following structure:

```
:- icsp(Type, L).
```

where `Type` is the model we want to recognize, `L` is a list of surfaces representing the object recognized. For example, for recognizing an L-shaped object, we query the system with:

```
:- icsp(l-shaped, L).
```

As a result, when the object is recognized the system returns a list of surface identifiers (feature identifiers in general) composing the object. The surfaces are stored in a file containing the area, the list of vertexes, the shape, the list of touching surfaces, the centroid coordinates and the surface normal.

In general the approach ICSP outperforms the standard CSP approach manly because feature extraction (the most time consuming operation) is substantially reduced. In some images (marked with *) the object was not present and in such cases all images must be explored and all surfaces must be computed in both techniques. When all surfaces are extracted in the image, the gain using an interactive approach is not particularly high. If the object is not present, the labeling procedure tries to instantiate the first variables to all the possible surfaces, so the low-level system must scan the whole image. Anyway, surface data are recorded, so a feature is never extracted twice. Moreover, each variable domain is probably smaller, during computation, than in the CSP case; in fact it contains only values consistent with interactive constraints. For this reason, even when the object is not present, we can have a gain in performance. In other cases (such as in image BLOCK_8) the ICSP suffers a little penalty, due to the higher number of choice points added. On the other hand, when the object is present, great improvement can be obtained, since extraction stops as soon as one solution is found. This extraction avoidance, combined with the fact that domains are kept smaller, deals a speedup of 2 to 8. This performance improvement proves the efficacy of an interactive approach of recognition that allows of using and managing all the actual knowledge required by the process.

The advantage of the approach is twofold: from the visual system viewpoint, on average, we acquire a smaller number of features since we guide the extraction by means of constraints. From the constraint solver point of view, we work with smaller domains thus increasing efficiency. 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.

7 CONCLUSION AND FUTURE WORK

We have presented a model for interactive CSP which can be used when data on the domain is not completely known at the beginning of the computation, but can be dynamically acquired on demand by a low level sensor system. More important, it is used in order to guide the search by generating new constraints at each step.

We have implemented the framework by extending the ECL^{i}PS^e CLP(FD) library and applied it to a case study of object recognition and identification in a vision system. Objects are modeled by means of constraints and constraint propagation is the general-purpose tool for detecting a solution. Therefore, information can be acquired on-demand from the low level visual system thus reducing computationally-expensive low level tasks.

We are currently studying how the Interactive CSP framework can be applied to other fields. In particular, it has been successfully applied to planning problems [1] where it has been used in order to progressively collect information on the real world. In addition, we are considering to define a general method for solving CSPs based on the interactive framework.

Future work concerns both the improvement of the Interactive CSP model and its propagation algorithms and the visual application. As concerns the ICSP model, we are currently testing other constraint propagation algorithms, like arc-consistency, in order to determine in which cases it could be applied instead of forward checking in the visual recognition. In the field of object recognition, future work is aimed at extending the system for integrating more complex visual features and for modeling the visual target in terms of hierarchical ICSP, for taking into account complex and structured objects. Moreover, the approach is currently under testing for 3D recognition on range images with an object centered model that solves the limitation of using different constraint graph for different views of the same object. Thus, we are currently studying the object centered model (briefly presented in section 2.1) that exploits visual constraints and virtual surfaces in order to recognize an object independently from the observer point of view. Detailed information on ongoing research can be found at:

`http://www-lia.deis.unibo.it/Research/Areas/icsp/icsp.html`

Acknowledgements

This work has been partially supported by CNR, Committee 12 on Information Technology (Project SCI*SIA) and MURST Project on "Intelligent Agents: Interaction and Knowledge Acquisition".

References

[1] R. Barruffi and M. Milano. Interactive constraint satisfaction for information gathering in planning. In *ECAI*, 1998.

[2] M.J. Dent and R.E. Mercer. Minimal forward checking. In *6th IEEE International Conference on Tools with Artificial Intelligence*, pages 432–438, 1994.

[3] M. Dincbas, P. Van Hentenryck, and H. Simonis. Solving the car sequencing problems in Constraint Logic Programming. In *Proceedings of European Conference on Artificial Intelligence ECAI88*, 1988.

[4] M. Dincbas, P.Van Hentenryck, and M.Simonis. Solving large combinatorial problems in logic programming. *Journal of Logic Programming*, 8(1-2):75–93, 1990.

[5] B. Draper, A. Hanson, and E. Riseman. Knowledge-directed vision: control, learning and integration. In *Proc. of IEEE, vol. 84, n. 11*, pages 1625–1681, 1996.

[6] ECRC. *ECL^{i}PS^e, User Manual Release 3.5.*

[7] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume II. Addison Wesley, 1992.

[8] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[9] P. Henderson and J.H. Morris. A lazy evaluator. In *3rd ACM Symposium on Principles of Programming Languages*, pages 90–103, 1976.

[10] P.Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[11] P.Van Hentenryck, H.Simonis, and M.Dincbas. Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

[12] A. Hoover, D.Goldgof, and K. Bowyer. The space envelopement: a representation for 3d scenes. *Computer Vision and Image Undestanding*, 69(3):310–329, 1998.

[13] A. Hoover, G. Jean-Baptiste, X. Jiang, P.J. Flynn, H.Bunke, D.B.Goldgof, K. Bowyer, D.W. Eggerf, A.Fitzgibbon, and R.B. Fisher. An experimental comparison of range image segmentation algorithms. *IEEE Transactions on PAMI*, 18(7):673–689, 1996.

[14] J.Jaffar and M.J.Maher. Constraint logic programming: a survey. *Logic Programming*, Special Issue on 10 years of Logic Programming, 1994.

[15] T.H. Kolbe, L. Plümer, and A.B. Cremers. Using constraints for the identification of buildings in aerial images. In *Proceedings of the 2nd International Conference on the Practical Application of Constraint Technology (PACT'96)*, pages 143–154, London, 1996.

[16] M.Herman and T.Kanade. Incremental reconstruction of 3d scene from multiple complex image. *Artificial Intelligence*, 30:289–341, 1986.

[17] M.H.Yang and M.Marefat. Constrained based feature recognition: handling non uniquitess in feature interaction. In *IEEE International Conference on Robotics and Automations*, 1996.

[18] J.A. Murder, A.K.Mackworth, and W.S.Havens. Knowledge structuring and constraint satisfaction: the MAPSEE approach. *IEEE Trans. on Pattern Analysis and machine intelligence*, 10(6):866–879, 1988.

[19] J.F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. Technical report, ILOG Headquarters, 1993.

[20] R.Cucchiara, M.Gavanelli, E.Lamma, P.Mello, M.Milano, and M.Piccardi. Interactive constraint satisfaction by visual constraints for 3d object recognition. In *LIA-TR98-001 University of Bologna*, 1998.

[21] R.Cucchiara, M.Gavanelli, E.Lamma, P.Mello, M.Milano, and M.Piccardi. Constraint satisfaction and value acquisition: why we should do it interactively. In *Proceedings of IJCAI'99*, 1999. To appear.

[22] T. Shiex, J.C. Regin, C.Gaspin, and G. Verfaillie. Lazy arc consistency. In *AAAI*, 1996.

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

[24] D.L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI-TR-271, A.I. Lab., M.I.T., Cambridge, Mass., 1972.