Interactive Constraint Satisfaction and its Application to Visual Object Recognition

R. Cucchiara, E. Lamma, P.Mello, M. Milano, M. Piccardi

Abstract

In classical Constraint Satisfaction Problems (CSPs) variable domains have to be statically defined at the beginning of the constraint propagation process. In some applications, however, the data acquisition/generation process is a computationally expensive task. We present an *Interactive Constraint Satisfaction* model for problems where knowledge is not completely known at the beginning of the computation, but can be interactively acquired during the computational process. Some variable domain values can be already available when the constraint propagation process starts, while others can be dynamically acquired during the computation only when needed (on demand). The constraint propagation process works on already known domain values and adds new constraints on unknown 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 guide the data acquisition process. We present the Interactive CSP model and a propagation algorithm. We propose an implementation of the framework in Constraint Logic Programming on Finite Domains, CLP(FD). A case study in the field of visual object recognition is considered in order to show the effectiveness of the proposed approach.

Keywords: Constraint Satisfaction, Constraint Logic Programming, Vision Systems

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 [7] (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 the search, by means of constraint propagation, combination of assignments which cannot appear in any consistent solution.

E. Lamma, M. Milano are with DEIS, Univ. Bologna, Viale Risorgimento 2 I-40136 Bologna, Italy. E-mail: {`elamma, mmilano`}`@deis.unibo.it`

R. Cucchiara, P.Mello, M. Piccardi are with Dip. Ingegneria, Univ. Ferrara, Via Saragat 1 41100 Ferrara, Italy. E-mail: {`rcucchiara, pmello, mpiccardi`}`@ing.unife.it`

Work partially supported by CNR, Committee 12 on Information Technology (Project SCI*SIA).

A CSP-based inference engine can be used in many applications. In particular, we focus on those applications where a low level system provides a large amount of (constrained) data to be processed. A typical example is a vision system used for computing visual features of *objects* in an image. Several examples of CSP-based reasoning systems have been proposed for object recognition (see for instance [11] and [12]).

These applications usually require some form of interaction between a low level module providing (constrained) data, and the CSP module. 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 processing the whole image) in order to create variable domains, and then start the constraint propagation process.

We argue that interleaving the generation/acquisition of domain values and their processing could greatly increase the performances of the problem solving strategy. 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 [6] 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 and check if they are not needed. This idea has been already exploited in the field of constraint satisfaction in [2, 14] 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 generation/acquisition process can be guided by constraints, called *interactive constraints*, thus leading to retrieve only consistent values and minimizing useless knowledge acquisitions. Note that driving the value acquisition process by means of constraints results in partially shifting the constraint satisfiability check from the constraint solver to the value generating module. In a visual system, this feature allows to focus the attention of the feature extraction module in 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* model where domains can be partially known when the constraint satisfaction process starts and are dynamically acquired during the computation.

In recent years Constraint Logic Programming (CLP) has been successfully used for solving hard combinatorial problems, [3, 4, 7] modeled as Constraint Satisfaction Problems (CSPs). CLP [9] is a class of programming languages combining the advantages of Logic Programming (LP) and the efficiency of constraint solving. 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 [5].

The main contributions of the paper are the following:

- define the *Interactive Constraint Satisfaction* model (section 2);
- provide a suitable algorithm for interactive constraint propagation, called *Interactive Forward Checking* (section 3) that copes with partially known domains;
- present the implementation of the framework in CLP(FD) (section 4);

- present a case study in the field of visual object recognition together with experimental results (section 5).

2 Interactive CSP

In this section, we define the interactive CSP model. 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 $D_1, \ldots, D_k$, i.e., a set of configurations of assignments which can 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 CSP 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 partial acquisition of data 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 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 finite 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, while $UnKnown_i$ is a domain variable itself representing information which is not yet available. 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 in the case of the standard CSP, an assignment of values to variables which is consistent with constraints.

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 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:

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

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

[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.

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 $[1, 3, X1]$ and $[-6, 4, Y1]$. The known part of the domain of X contains two values $[1, 3]$, while its unknown part is a domain variable itself, $X1$, representing not yet available values for X. Similarly, the known 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 defined 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.

In figure 1 we sketch the pseudo-code for the 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. Procedure propagate_partially_known queries the low level system in order to retrieve values consistent with the known part of X for variable Y. 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 acquisition will be consistent with constraints.

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

An important point to be discussed concerns the number of values retrieved by the data acquisition process. Basically, one (lazy) choice is to stop the data acquisition as soon as one consistent value has been retrieved. In addition, a lazy approach can avoid to start a knowledge acquisition for a variable associated with a non empty known part. In this case, the propagate_partially_known procedure just produces a set of constraints on the unknown domain variable. On the contrary, an eager choice collects all consistent values. In the first case, after the knowledge acquisition, the variable representing the unknown part of the domain cannot be removed from the domain itself since the data acquisition is not complete. In the second case, instead, we can remove this variable and result exactly in the same domain obtained after a classical constraint propagation.

In the above mentioned example, the constraint propagation between the known part of the constraints can be performed as usual. The consistency check between values 1 and 3 for variable X and the undefined part of variable Y, i.e., $Y1$, calls for a data acquisition that is aimed to collect one or each value for Y which are consistent with the defined part of the domain of X. In other words, the system collects for variable Y one or each value which are greater or equal to 1 and 3. This is equivalent to pose a constraint on the undefined 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 undefined part of X and the defined part of Y. Finally, the constraint between undefined part will check new acquired values as soon as they will be available.

```
procedure propagate_constraints($D$,$c(A, B)$)
begin
        $D_A = Known_A \cup UnKnown_A$;
        $D_B = Known_B \cup UnKnown_B$;
        propagate_defined($Known_A$,$Known_B$);
        propagate_partially_defined($Known_A$,$UnKnown_B$);
        propagate_partially_defined($Known_B$,$UnKnown_A$);
        propagate_undefined($UnKnown_A$,$UnKnown_B$);
end

procedure propagate_partially_defined($Known_X$,$UnKnown_Y$);
begin
        guided_acquisition($c(Known_X, UnKnown_Y)$);
end;

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

Figure 1: The interactive constraint propagation

3 Interactive Forward Checking

One of the well known and widely accepted propagation algorithms for solving CSPs is the forward checking (FC) technique [7]. 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 behavior of the FC algorithm should be changed accordingly. Intuitively, the first *labeling* step instantiates a variable X to a value v in its 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 data for the unknown part[3].

Note that in the algorithm presented in Figure 1, only the first two procedures (propagate_defined and the first propagate_partially_defined) are performed since for the forward checking strategy one variable is always instantiated (thus completely known).

Let us consider an example. The task is to recognize a rectangle in a scene by its four edges X_1, X_2, X_3 and X_4 defined as the variables in the CSP. Variable domains are segments retrieved from the image. Initially, the variable domains are completely unknown:

$X_1 :: [UnKnown_1], X_2 :: [UnKnown_2],$
$X_3 :: [UnKnown_3], X_4 :: [UnKnown_4]$

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

The FC algorithm starts with a labeling step on variable X_1. Since the domain of X_1 does not contain already acquired values, the labeling step performs a feature acquisition process (possibly guided by unary constraints on X_1). A segment s_1 in the image is retrieved and assigned to X_1. Now, the FC algorithm collects all the variables linked to X_1 by means of constraints and removes from the known part of their domains all values which are inconsistent with s_1. If variable domains are completely unknown, or the first step removes all values from the known part, a data acquisition is performed in order to retrieve those values consistent with s_1. Suppose we have two constraints stating that variables X_2 and X_4 should be perpendicular to X_1 in any consistent solution, i.e., $perpendicular(s_1, X_2)$, $perpendicular(s_1, X_4)$.

The constraint check results in a data acquisition process since both X_2 and X_4 domains are still unknown. Note that the feature acquisition process can be guided by the two above mentioned constraints. Therefore, the underlying visual system looks for one or all segments which are perpendicular to s_1 thus focusing attention around s_1.

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 domain 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 s_1. In the second case, the domains of X_2 and X_4 are left partially known (i.e., the variable representing the unknown part cannot be removed from the domain). This second approach is similar to performing a minimal forward checking algorithm [2].

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, suppose that the system collects three segments s_2, s_3 and s_4[4] which are put in the domains of X_2 and X_4.

The unknown parts of X_2 and X_4 can be "removed" from the corresponding domains since they represent all values which are not perpendicular to (and thus not consistent with) s_1. Now, a labeling step starts for X_2 thus assigning s_2 to X_2. A second constraint propagation process starts by considering all variables involved in a constraint with X_2. Suppose we have a constraint between X_2 and X_3 stating that the two variables should be perpendicular, 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 perpendicular to X_2, say s_5 and s_6. 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 in general a number of segments which is significantly lower than those retrieved by a non-interactive system which first collects all the segments and then starts the constraint propagation process. Note that in the worse case, the number of features extracted is equal in the two frameworks.

In figure 2, we have sketched the basic Interactive FC algorithm. It first selects a variable Var to be instantiated, then it performs an interactive labeling procedure (interac-

[4]Note that constraints describing the rectangle are symmetric. Symmetries should be avoided in a constraint satisfaction procedure [13]. 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.

```
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 2: The incremental forward checking algorithm

tive_label) which 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 1 starts.

4 Implementation

We have implemented the Interactive Constraint Satisfaction framework on top of the finite domain library of the ECL^{i}PS^e language [5]. We have chosen to exploit Constraint Logic Programming [9] on finite domains (CLP(FD)) since it is a very effective programming paradigm for solving CSP [7, 8]. 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_domain` which extracts a partially known domain from a variable;

- `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,Dom1)
      ->   (dvar_domain(S2,_)
             ->   touch(S1,S2)
             ;    (nonvar(S1)
                    -> 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)
           ))
     ;    (dvar_domain(S2,Dom2)
             ->   (nonvar(S2)
                    -> dom_to_list(Dom2,L2),
                       acquisition(L2,L1), specify_domain(S1,L1)
                       ;  make_suspension(itouch(S1,S2),4,Susp),
                          insert_suspension(S2,Susp,any of fd,fd),
                          insert_suspension(S1,Susp,specify of dom_pd,dom_pd)
                  )
           ;    make_suspension(itouch(S1,S2),4,Susp),
                insert_suspension((S1,S2),Susp,specify of dom_pd,dom_pd)
           )
    ).
```

In particular, if both variables are known the non-interactive constraint is called `touch(S1,S2)`. 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 the CLP itself, but only its operational behaviour.

5 A Case Study on Object 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 the efficient description of object models by means of visual primitives and their relations. Each object 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). A graph representation of object models has been used in many different contexts of 2D shapes [12], and extended to the 3D scene recognition [11],[10].

As objects can be modeled by means of constraints, objects can be recognized by means of constraint satisfaction. 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 topological relationships between them can be represented by binary constraints (examples of geometric constraints are angular relationships between contours, lines, or surfaces, topological constraints are spatial relationships, such as *is adjacent to*, *touch*, *is contained* or others).

For instance, if we want to model a rectangular shape 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:

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

In the defined set of constraints, only some can be used for guiding the data acquisition process (interactive constraints), while others can only be checked by the constraint solver as usual. In particular, the constraints involving some form of spatial locality could be exploited to limit the image processing in a specific region of interest. In the above mentioned example, the *touch* and *perpendicular* constraints are used interactively for guiding the visual process, because they issue a focused visual search (of object edges) in a spatial neighbourhood of a given vertex. We are currently exploring a set of guidelines for problem modeling aimed at defining which constraints should be treated as interactive and which should be considered as standard constraints.

The ICSP-based approach results particularly efficient if the size of variable domains is large, as in the case of images containing a large set of candidate objects, since the constraint propagation and the interaction with the vision system considerably limits the search space by identifying which objects to look for.

In order to explore this aspect, we have tested our approach on different object recognition contexts, both for recognizing 2D shapes and 3D objects. In this paper we report the case study of searching 2D rectangles (as modeled above) in images containing many overlapped rectangles. Images are taken from the image database of the well-known DARPA image understanding benchmark [16]. The benchmark consists of a model-based vision task, that is the recognition of a complex object made up of many basic rectangles, tied each other by spatial relations rather than physical contours. The first step of recognition consists of reliable detection of the basic rectangles, which vary in size, intensity, orientation, and location; detection is disturbed by sensor noise, leading to miss some rectangle edges or even whole rectangles. The second step consists of matching sets of rectangles against an object database, in order to detect possible objects.

While the benchmark was defined by the vision community as a general framework for performance evaluation in image understanding, in our work we have used images from the DARPA benchmark (512 x 512 range and intensity images) as an established data set for experimenting the constraint-based approach.

The images contain tens of rectangles partially overlapped and cluttered and the experiments aim at performing a visual search with selective attention; in particular the goal is to find at least one object in the scene according to the defined model; the process can be extended for finding a number of objects in the image.

Images have been initially processed with standard filtering operators in order to smooth the sensor noise [15]. Then edges are extracted and a "k-curvature" algorithm [16] selects possible corners and therefore detects the segments, which represent the features used as nodes in the model. Thus, the vision system is able to acquire, if required, a single segment in a region of interest of the image, a set of segments satisfying some constraints, or possibly all the segments of the image.

Performances in terms of the time spent for achieving object recognition depend on image complexity, which is a function of the number of possible model instances that can be found and the number of features to be evaluated (belonging to both the actual model instances and other shapes). In this paper we report experiments on images containing the same number of not-occluded objects (10), but an increasing number of features that can be extracted, and thus an increasing size of the variable domains.

Figure 3 shows results of six different images containing from 53 to 213 segments in which to recognize 1 to 10 non-occluded rectangles. The encouraging performance of ICSP are described in the tables showing the execution times in seconds measured on a Sparc 10 workstation (column N) spent to find 1 to 10 (all) solutions, both with the standard CSP algorithm working on the whole segment set extracted from the image (column CSP) and with the proposed ICSP that interacts with the segment acquisition system (column ICSP).

In all cases, our approach significantly outperforms the classical constraint satisfaction approach. We have also reported the number of retrieved segments in both cases: column Nseg CSP reports the number of segments retrieved in the CSP framework. Obviously, the number of segments is constant because we retrieve all segments before starting the constraint propagation process. Column Nseg ICSP, instead, reports the number of segments retrieved by the ICSP framework. In general, in the extended framework the number of failures and consequent backtracks are less than the standard case. However, in the worse case, the number of segments retrieved and the number of choice points in the derivation are the same.

The main advantage of the approach is twofold: from the visual system viewpoint, in the average case, we acquire a fewer number of segments 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.

6 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 finite domain 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-

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	1.66	53	0.08	4
2	3.23	53	0.45	8
3	4.40	53	0.75	13
4	5.38	53	1.13	17
5	6.78	53	1.60	22
6	7.58	53	1.90	26
7	0.94	53	2.23	31
8	10.28	53	2.63	37
9	10.85	53	2.80	42
10	12.93	53	3.37	47

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	4.05	60	0.56	8
2	6.90	60	1.03	12
3	7.57	60	1.63	16
4	8.45	60	1.83	20
5	8.81	60	2.47	29
6	9.52	60	2.93	37
7	9.82	60	3.13	43
8	10.20	60	3.70	47
9	11.00	60	4.32	52
10	14.55	60	5.12	56

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	9.23	71	0.36	9
2	12.68	71	1.33	17
3	15.85	71	1.98	25
4	18.77	71	2.95	31
5	21.11	71	4.28	39
6	21.60	71	4.97	47
7	22.10	71	5.68	51
8	25.00	71	6.72	55
9	25.11	71	7.00	63
10	26.35	71	8.07	71

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	2.33	90	0.30	7
2	5.25	90	0.83	14
3	5.98	90	1.22	18
4	6.95	90	1.42	24
5	7.30	90	1.97	35
6	8.03	90	2.37	42
7	8.30	90	2.55	55
8	8.73	90	2.73	62
9	9.55	90	3.40	73
10	10.55	90	4.07	79

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	235.1	174	6.22	21
2	239.5	174	7.32	25
3	357.1	174	10.2	43
4	387.1	174	12.6	50
5	465.5	174	20.8	75
6	554.9	174	25.8	90
7	793.9	174	42.5	114
8	831.8	174	50.5	129
9	910.5	174	59.8	137
10	1023.4	174	65.4	157

N.	CSP	Nseg CSP	ICSP	Nseg ICSP
1	139	213	7.6	26
2	144	213	9.4	31
3	159	213	13.6	51
4	175	213	16.8	60
5	203	213	25.7	90
6	205	213	32.7	113
7	300	213	53.3	140
8	309	213	63.9	168
9	349	213	73.8	181
10	377	213	82.2	198

Figure 3: Experimental Results

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 succesfully 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 methodology for solving CSPs based in 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; in this case feature extraction algorithms are computationally very expensive, and focussing the attention on a limited part of an image could result in a considerable gain in efficiency.

References

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

[2] M.J. Dent and R.E. Mercer. Minimal forward checking. In *6th IEEE International Conference on Tolls 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] ECRC. *ECL^iPS^e, User Manual Release 3.5.*

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

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

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

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

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

[11] 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.

[12] 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.

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

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

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

[16] C. Weems, E. Riseman, A. Hanson, and A. Rosenfeld. The DARPA image understanding benchmark for parallel computers. *Parallel and Distributed Computing*, 11:1–24, 1991.