Synthesis of Object Models from Partial Models: A CSP Perspective

Marco Alberti[1] and **Evelina Lamma**[2]

Abstract.

In this work we present an approach for the synthesis of object models (expressed as Constraint Satisfaction Problems, CSPs) from views or partial models (expressed, in their turn, as CSPs as well).

The approach we propose is general enough to consider different types of features and relationships in the views. This is achieved by introducing the notion of model representation, where features, relationships and their domains are expressed. The (complete) model can be synthesized through a proper algorithm, which provides a labeling between the (complete) model and the partial models' components. The generated CSP representing the synthesized model must satisfy (or, better, entail) any constraint among features and any relationship occurring in each partial model.

The framework is applied for synthesizing object models (i.e., CSP descriptions). We provide two basic approaches for synthesizing a minimal or a correct model, and we experiment them by considering some case studies in artificial vision.

1 Introduction

In Computer Vision, many different types of problems are addressed. Among them, both Image Recognition and Visual Search require the process to have a model or a set of models at disposal. In both problems, the association process between the image and the model is very complex, and heavily influenced by the type of model itself. The nature of the model will also influence the matching algorithm, and the expressiveness of possible queries.

Constraints have been proved a very general methodology for modeling objects in Computer Vision, since they can use many different representations: they can take into account many different feature types such as surfaces, lines, etc., and different kinds of relations among features. Furthermore, constraint-based models can be easily defined independently from visual deformations. They are also additive in the sense that if a model is not selective enough to distinguish an object, it can be easily extended and refined later on by adding new constraints.

For these reasons, constraints have been used for object modeling and for solving the matching problem of Computer Vision since the beginning of the study of Constraint Satisfaction [11]. The model of the object must be reliable and general: for this reason, it is often based on geometric and topological relations among features (see, for instance, [4, 6, 10]). Each object can be represented by a constraint graph, where each characterizing primitive feature is a node and relations among object parts (i.e., features) are represented by arcs. An equivalent representation can be given as Constraint Satisfaction Problem (CSP, for short in the following).

In many cases, the model of the object is not directly available and its creation is left to a human operator. When this operation is difficult, an automatic model construction procedure would be useful.

It is reasonable to suppose that the data for the model construction problem are images representing different views of the object. In the literature, this problem has been approached for particular cases. For example, the construction of viewer-centered 3-D models from intensity images is considered in [9]; GEON-based (see Biederman's theory of recognition by components in [3]) model construction has also been addressed, for example in [12] or in [13].

In this paper, we present an approach for the synthesis of object-centered models, considering partial models (views) as input data. Both the input views and the target models can be represented as CSPs, and the model construction problem itself can be considered a CSP as well.

Our approach differs from those of the previously cited works in that it assumes no strict hypotheses about the nature of the primitives chosen to describe the objects and their views, and the possible relations among them; indeed, the concepts our work is based upon are not even intrinsically linked to Computer Vision. The algorithms here proposed are designed to deal with high level descriptions of the object views; these descriptions are supposed to be acquired by a low level processing system, such as a range image segmentation system. The nature itself of the available information does not affect the algorithms directly. This lets our work achieve higher generality, and a wider range of applicability (possibly even outside the field of Computer Vision), although with the drawback of a lack of error tolerance with respect to input data and, in general, of less accurate domain-specific results, if compared to existing works (at least, at this stage of our research).

The notion of model representation is first introduced (in Section 2), where features, relationships and their domains are expressed. The (complete) model can be synthesized through a proper algorithm (discussed in Section 3), which provides a labeling between the (complete) model and the partial models' components. The generated CSP representing the synthesized model must satisfy (or, better, entail) any constraint among features and any relationship occurring in each partial model. In this respect, the object model generation can be considered, in its turn, a CSP, since it consists in finding a labeling for the elements of the generated object model (i.e., the variables of the problem) consistent with the constraints of the problem (i.e., with the constraints involved in the partial models or views).

We provide different implementations of the approach (in SWI-Prolog [1] and by exploiting the Constraint Handling Rules [7] li-

[1] Dipartimento di Ingegneria, Università degli Studi di Ferrara, Via Saragat 1, 44100 Ferrara, Italy - email: malberti@ing.unife.it

[2] Dipartimento di Ingegneria, Università degli Studi di Ferrara, Via Saragat 1, 44100 Ferrara, Italy - email: elamma@ing.unife.it

brary of SICStus Prolog [2]) in order to synthesize a *minimal* or a *correct* model (see Section 3). Section 4 discusses experimental results obtained by applying the implemented algorithms to some case studies in artificial vision. We then conclude and mention future work in Section 5.

2 Problem representation

To achieve higher generality, in this section we define the concepts involved in the problem formulation, at the highest possible level of abstraction. We avoid mentioning here a priori hypotheses about the kind of primitives chosen for the model representation. Concrete examples are shown in section 4.

2.1 Model representation

A model is a triple (E, A, R) in which:

- E is the element set.
  An element is an entity which is considered atomic at the chosen level of abstraction. E is supposed to be a finite set.
- A is the attribute set.
  An n-ary attribute is meant to describe a feature of an n-ple of elements. Attributes may be real (i.e., representing a feature whose value can vary continuously and subject to measurement and representation errors) or discrete (i.e., representing a feature whose value belongs to a discrete set). Attribute values are vectors, whose number of components is called the *dimension* of the attribute. Formally, an n-ary attribute with dimension m is described by a function
  $a : S \subseteq E^n \rightarrow B$
  where B is the set of finite open intervals in $\Re^m$ for real attributes and the cartesian product of m discrete sets for discrete attributes.
- R is the relation set.
  Relations are defined in the ordinary mathematical way, i.e., an n-ary relation r is a subset of E^n, if E is the elements set.

Given such representation, it is straightforward to express the model as a CSP, where E is the set of variables, A and R define constraints respectively over and among elements of E.

2.2 Labeling

A *labeling* function defines the correspondence between a view of an object and a (complete) model. For any given element of a view the value of a *labeling* function is the correspondent element of the model. More formally, a *labeling* of a view with a model is a function:

$l : E_V \rightarrow E_M$

where E_V is the element set of the view and E_M is the element set of the model.

The labeling must be injective (i.e., any element of the view must have only one correspondent element in the model).

Given this function, the functions

$l_n : E_V^n \rightarrow E_M^n$

$[e_{V1}, e_{V2}, ..., e_{VN}]^T \mapsto [l(e_{V1}), l(e_{V2}), ..., l(e_{VN})]^T$

are also defined.

In order to identify correct labeling functions, we introduce the notion of compatibility constraints over attributes and relations as follows .

Two values of a discrete attribute are compatible if they are equal.

Two values of a real attribute are compatible if their intersection is non-empty.

An n-ary attribute a_V defined for a view is compatible for a given labeling l with the correspondent attribute a_M defined for a model if for any n-ple e_V of elements of the view for which the attribute is defined, the attribute is also defined for $l_n(e_V)$ and the values are compatible.

An n-ary relation r_V defined for a view is compatible for a given labeling l with the correspondent relation r_M defined for a model if for any n-ple e_V of elements of the view:

$e_V \in r_V \Leftrightarrow l_n(e_V) \in r_M$.

If all the attributes and the relations of a view are compatible with their corresponding elements of a model for a labeling l, l is said to be a *compatible labeling* of the view with the model.

If there exists a compatible labeling of a given view with a given model, the view is said to be a *view of* the model.

Notice that compatibility constraints imply that only view-invariant attributes and relations can be used for views and model descriptions.

2.3 Problem statement

The problem considered here can thus be stated as:

"Given a set of views of an object, find a model such that all of the views are *view of* the model".

In the following, we will refer to this as a *model synthesis* problem.

A solution to the problem is defined by both the model and the labeling functions for the views.

2.4 Quality of solutions

For any model synthesis problem, there is at least one trivial solution, obtained by the juxtaposition of the views. This is achieved by constructing a model such that each element of each view is labeled with a distinguished element of the model. Obviously, this solution is of no use whenever a more concise model is of interest to be used later on, for instance, in visual search.

In this case, it makes more sense to search for solutions by performing a better synthesis of views, i.e., by labeling elements of different views with the same element of the model, keeping the *view of* relation between the views and the model valid.

The set of all the possible solutions is quite huge for practical problems. However, we identify two classes of especially significant solutions among them:

- *minimal solutions*, the solutions whose model has the minimal cardinality compatible with compatibility constraints;
- *correct solutions*, the solutions whose labeling is *coherent*, i.e., if two elements of different views represent the same entity of the object, then they are labeled with the same element of the model and vice-versa.

Notice that, with the notions above, minimal solutions are possibly incorrect.

3 Searching for solutions

In this Section we present two basic approaches which have been followed and implemented for searching solutions to the synthesis problem. They focus respectively on one of the two classes of solutions mentioned in the previous Section, namely *minimal* and *correct* solutions.

3.1 A standard backtracking approach

The first approach generates minimal solutions for the synthesis problem. The approach starts by considering the minimal cardinality equal to the maximum view cardinality (which is a lower bound for the model cardinality, because of the injectivity of the labeling function). This assumption may be revised later on, if proved wrong during model construction. In this case, the process is newly started, with a cardinality increased by one.

3.1.1 Algorithm

The algorithm we have devised follows a standard backtracking approach. Basically, two ways of performing backtracking are considered: the first is over the number of elements of the generated model, the second over the possible labeling between views' elements and model's elements.

The main cycle of this algorithm is sketched here, using Prolog syntax:

```
get_model(Views,Model,Labels):-
  get_min_card(Views,MinCard),
  build_model(MinCard,Views,Model,Labels).

build_model(Card,Views,Model,Labels):-
  label_views(Card,Views,
  ([],[],[]), % the empty element,
              % attribute and relation lists
  Model,Labels),
  !.
% choice point on model cardinality
build_model(Card,Views,Model,Labels):-
  NewCard is Card+1,
  build_model(NewCard,Views,Model,Labels).

label_views([],_,Model,Model,[]).
label_views([View|MoreV],Card,
            OldM,NewM,[ViewL|MoreL]):-
  label_view(View,Card,OldM,IntM,ViewL),
  label_views(MoreV,Card,IntM,NewM,MoreL).

label_view(([],_),_,Model,Model,[]).
label_view(([Elem|MoreE],Info),Card,
           OldM,NewM,[(Elem,Label)|MoreL]):-
  %choice point on labeling
  choose_label(Card,Label),
  update_model(Info,Elem,Label,OldM,IntM),
  label_view((MoreE,Info),IntM,NewM,MoreL).
```

Once a cardinality for the model to be constructed is chosen, the approach starts by constructing the labeling functions for each view while constructing the model according to the views' data, applying a *standard backtracking* procedure.

The procedure for labeling a view (*label_view/5* predicate) takes the view's data and the temporary model (resulting from previous views processing) as input parameters, and returns the labeling function for the view and the updated temporary (final if the view is the last) model as output parameters.

For each element in a view, an attempt is made (*choose_label/2* predicate is non-deterministic) to assign it one of the possible labeling values (i.e., one of the still not assigned model elements), and to update the model (*update_model/5* predicate) with the view's data concerning the element.

If the updating succeeds, the program considers the next element, and so on for all the elements and views. Anyway, all the other possible labeling values are left open as choice-points for possible backtracking.

If the updating fails (for example because it would imply different values for a discrete attribute), the program tries another labeling value. If no value allows a successful updating, the choice points for previous elements and views are considered. If no possible choice produces a consistent model, then the model cardinality has been underestimated, and the whole process is repeated after increasing the cardinality by one (choice point for the *build_model/4* predicate).

3.1.2 Implementation

The algorithm which generates minimal solutions for the synthesis problem has been implemented in the Prolog language and tested using SWI-Prolog (see [1]).

The program takes two files as input: the *template file*, describing arity and dimension of attributes and arity of relations, and the *views file* which contains the description of object views. Both files (sequences of Prolog facts) are parsed, and loaded into an appropriate data structure to increase efficiency.

The program produces two output files (again, sequences of Prolog facts): the *model file*, describing the model, and the *labeling file*, with the values of the labeling function for all views.

3.1.3 Complexity

This algorithm guarantees the minimality of the solution, but its computational complexity (factorial over the number of elements of the model and exponential over the number of views in the worst case) may, in many practical cases, make it not viable.

Thus, we have also implemented a modified version of the program which limits the backtracking to a single view, i.e., adds a new element to the model when it cannot find a consistent labeling for a view, without reconsidering the labeling for the previous views.

This version can generate non-minimal solutions, but its complexity is dramatically reduced.

3.2 A CSP approach

A searching algorithm for correct solutions must face many inherent difficulties:

- the number of solutions is huge for practical problems;
- there are no given criteria to determine whether a solution is correct;
- the available information about views may not be sufficiently characterized to guide the labeling choices.

Thus, it is very important to design the algorithm so that it uses the available information to guide the process of solution building.

For this purpose, the approach which has appeared most appropriate is to use the conceptual paradigm of *Constraint Satisfaction Problems* to express the problem.

3.2.1 *The synthesis problem as CSP*

As well as the model descriptions, the synthesis problem can be stated as a *Constraint Satisfaction Problem* (CSP) on finite domains.

This can be achieved associating view elements to variables and model elements to domains. The constraints among variables are *alldifferent* among variables representing elements of the same view, and inequality constraints among n-ples of elements of different views with incompatible values of attributes and relations.

A solution of such CSP will also be a solution of the synthesis problem, because the inequality constraints guarantee the *view of* relation between any of the views and the model.

3.2.2 *The ad hoc constraint handler*

Although there are several available systems for solving CSPs on finite domains (for example, CLP(FD) extensions of Prolog systems), the direct use of any of them, as well as some (however solvable) practical problems, would have generated difficulties concerning search strategy and complexity.

Thus, we have developed a constraint handler tuned for this problem, using the Constraint Handling Rules (CHR) language (see [7]), which is perfectly suitable for high-level design of constraint systems. We used the reference implementation of the language, the CHR library of SICStus Prolog (see [2]).

The views descriptions to be processed are contained in a file to be consulted by the SICStus interpreter. A utility program is provided which converts the *views file* for the Prolog program into the equivalent CHR program.

The handler is based on a *forward checking* algorithm which uses the available inequality constraints to reduce the domains of the variable during labeling. There are, however, some differences between our approach and that of most CSP(FD) solving systems. Mainly:

- instead of taking the variable domains as data for the problem, the handler builds the domains dynamically. Specifically, when a variable is found which has an empty domain, instead of considering this as a failure and generating backtracking, the handler adds a new element to the domains of all uninstantiated variables. This is equivalent to adding a new element to the model as soon as, for the chosen labeling, the model cardinality is not sufficient.
  This choice has the main aim of limiting complexity, because allowing unconditioned backtracking would have generated the same complexity problems highlighted in the previous Section for the Prolog program; however, after a full solution has been built, it is possible to backtrack the labeling choices;
- if all the possible inequality constraints were derived from attribute and relation values and stored before the beginning of the labeling procedure, they would number in tens of thousands for practical problems. Moreover, most of them could not be used for domain reduction at early labeling steps, because, except for unary attributes and relations, inequalities concern *n*-ples of elements. This would be yet another heavy complexity related problem.
  Instead, the handler only derives and stores, during the labeling procedure, those inequality constraints which are applicable to domain reduction, i.e., those which have an already instantiated member. A check is also done to prevent multiple imposition of constraints, which would be harmless but uselessly expensive.

It is apparent that preventing backtracking implies that any non-coherent labeling will not be recovered until a full solution has been constructed, which will be necessarily be erroneous.

To address this problem, we have implemented a heuristic system which, at each labeling step, chooses the variable to be instantiated according to the *first fail* principle, and assigns it the value which is most probably the correct one according to a heuristic function[3].

The function has been designed to assume higher values for those variable-value couples which make a better match between the available information about the variable (taken from the view) and about the value (taken from the model), applying the *least constraining principle*. The function can be customized by assigning weights to attributes and relations and to specific values of these, so defining which are considered more significant and should guide the labeling procedure.

4 Experimental results

To test both the algorithms, we consider a case which is quite common in Computer Vision applications: we assume as data of the problem the description of multiple views of polyhedra, using information of the same kind of what can be extracted by a segmentation system from range images. A sample polyhedron with sample views is represented in Figure 1.

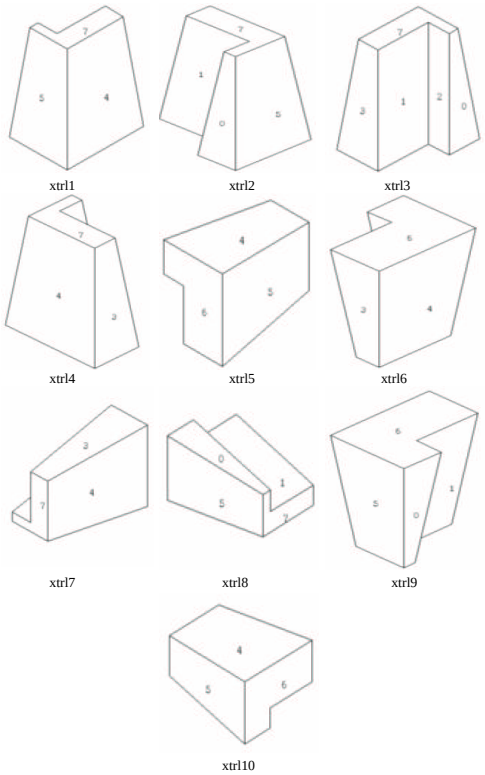

Figure 1. An example of multiple views of an object

Indeed, the input files for the programs have been obtained by automatic processing of files whose format is the same of the output files generated by the segmentation system adopted in [5].

In more detail,

- model elements are surfaces;

[3] It is however possible to disable the *first fail* criterion, and sort, instead, the possible labeling by the heuristic function value.

- model attributes are:
  - *shape*: a discrete unary attribute with dimension 1, representing the number of vertexes of its argument;
  - *angle*: a real binary attribute with dimension 1, representing the angle between the normal of its two arguments;
  - *destr*: a real ternary attribute with dimension 1, representing the angle between the vectorial product of the normal of the first two arguments and the normal of the third (which reduces to 0 for a right-handed triple of versors and to π for a left-handed triple of versors);
- model relations are:
  - *touches*: a binary relation representing the existence of a common edge of its two arguments;
  - *commonvertex*: a ternary relation representing the existence of a common vertex of its three arguments;

Two different working hypotheses can be made:

- *variable orientation*: view images are taken by a single sensor by rotation of the object;
- *fixed orientation*: view images are taken by multiple sensors, with the object position fixed, as in stereo-vision for instance.

The difference is that in the first case the spatial orientation of the same surface is in general not the same in different views, so it cannot be used as an attribute (it is not *view invariant*), whereas in the second it can be used in the form of the *normal* unary real attribute with dimension 3 during model generation, although, obviously, it cannot be part of the model definition when it is used for object recognition.

We have run the tests in both conditions over a sample database of four synthetic objects.

Table 1 summarizes, for both variable and fixed orientation, how many out of four minimal and correct solutions are found by:

- the Prolog program with full backtracking (Prolog A);
- the Prolog program with only view backtracking (Prolog B);
- the CHR program using no heuristics (CHR no heu);
- the CHR program using heuristics (CHR heu).

Program	Variable Orientation		Fixed Orientation	
	minimal	correct	minimal	correct
Prolog A	4 / 4	0 / 4	4 / 4	3 / 4
Prolog B	3 / 4	0 / 4	4 / 4	3 / 4
CHR no heu	0 / 4	0 / 4	4 / 4	3 / 4
CHR heu	0 / 4	0 / 4	4 / 4	4 / 4

Table 1. Test results

It can be immediately noticed that results, especially as far as correct solutions are concerned, are much better in the case of fixed orientation: this is quite obvious, since space orientation allows the programs to uniquely identify the surfaces unless the object has parallel surfaces. When this is the case, only the CHR program, by using appropriate heuristics, constructs a correct solution.

As far as minimal solutions are concerned, the algorithm with view backtracking only (Prolog B) behaves almost as fine as the full backtracking version (Prolog A), and, most of all, its computation time is much shorter and less influenced by the data.

However, it is worth to remind that all of the programs find actual solutions to the synthesis problem: this means that, if the chosen views set is representative enough, all the obtained models are suitable for object recognition applications.

5 Conclusions and Future Work

In this work, we have introduced a (CSP) perspective for the synthesis of object models from object views, represented as partial models. Both partial and generated models can be expressed, in their turn, as CSPs as well. The approach here presented shifts up, at model generation level, the nice properties of constraint-based models in Computer Vision, such as reliability and generality with respect to considered features and relationships among them.

We have introduced the model generation problem as a CSP problem, and we have proposed an approach for its solution as general algorithm. Different implementations have been provided, respectively by using SWI-Prolog and by exploiting the Constraint Handling Rules library of SICStus Prolog. The implementations respectively generate *minimal* and *correct* models, and have been tested on some examples in the Visual Search domain.

Future work will be devoted to integrate the approach here presented with the Interactive CSP approach of [8], where domain variables are generated interactively. This can be useful in the implementation of the proposed CSP approach for the synthesis problem, being the number of model elements (i.e., domains in the CSP approach) not known at the beginning of the generation process.

Possible future improvements of the CHR algorithm are the integration of domain-specific knowledge (such as geometric consistency checks over the resulting models) and the implementation of a selective backtracking strategy, which would allow retracting the most uncertain (with respect to the implemented heuristics) labeling choices.

REFERENCES

[1] SWI-Prolog Home Page, http://www.swi-prolog.org.
[2] SICStus Prolog Home Page, http://www.sics.se/sicstus.
[3] Irving Biederman, 'Recognition by Components - a Theory of Human Image Understanding', *Psychological Review, no. 2, 1987*, **94**, 115–147, (1987).
[4] R. A. Brooks, 'Symbolic Reasoning among 3-D models and 2-D images', *Artificial Intelligence*, **17**, 285–348, (1981).
[5] R. Cucchiara, M. Gavanelli, E. Lamma, P. Mello, M. Milano, and M. Piccardi, 'From Eager to Lazy Constrained Data Acquisition: A General Framework', *New Generation Computing*, **2001:19**(4), 339–367, (2001).
[6] B. Draper, A. Hanson, and E. Riseman. Knowledge-directed vision: control, learning and integration, 1996.
[7] Thom Frühwirth, 'Theory and Practice of Constraint Handling Rules', *Journal of Logic Programming, Special Issue on Constraint Logic Programming*, **37**(1-3), 95–138, (October 1998).
[8] Evelina Lamma, Paola Mello, Michela Milano, Rita Cucchiara, Marco Gavanelli, and Massimo Piccardi, 'Constraint Propagation and Value Acquisition: Why we should do it Interactively', in *IJCAI*, pp. 468–477, (1999).
[9] A.R. Pope and D.G. Lowe, 'Learning Object Recognition Models from Images', in *ICCV93*, pp. 296–301, (1993).
[10] D. Vernon, *Machine Vision: Automated Visual Inspection and Robot Vision*, Prentice Hall, 2001.
[11] D. I. Waltz, 'Generating Semantic Descriptions from Drawings of Scenes with Shadows', in *The Psychology of Computer Vision*, ed., P. H. Winston, chapter 3, McGraw-Hill, New York, (1975).
[12] K. Wu and M.D. Levine, 'Recovering Parametric Geons from Multiview Range Data', in *CVPR94*, pp. 159–166, (1994).
[13] Kenong Wu and Martin D. Levine, 'Segmenting 3D Objects into Geons', in *ICIAP*, pp. 321–334, (1995).