Learning Object Models in Visual Semantic Networks

A. Gupta \hspace{1cm} A. Bagchi

National Center for KBCS \hspace{1cm} CSSC

Indian Statistical Institute, Calcutta, India

Abstract

Computer vision applications often require a library to store descriptions of visual object models. Acquisition of new models in such a library is treated as a supervised learning scheme, where the system and a Knowledge Engineer interact to discover generalized object descriptions from a set of positive examples. Heuristic algorithms for this purpose are proposed on a novel model representation scheme called Visual Semantic Networks.

1 Introduction

Computer vision applications often require a library to store descriptions of visual object models. Object recognition involves matching an input object description with an appropriate model. Strategies for searching such libraries correctly and efficiently are discussed in [2]. Our purpose is to provide such a library with a capability to "learn" new object models in the presence of a Knowledge Engineer (KE for short). This model acquisition procedure is executed in two steps. First the new object model is inserted into the library; the KE helps the insertion by telling the system the broad class (say, spanner) to which the new object (say, single-ended ring spanner) will belong. The system, as described in the section 2.2, inserts the object description part by part into the appropriate place in the library. In the second phase, the system assimilates the new object model by making inductive generalizations about the class as a whole. In other words the introduction of the new model may help the system to make some inductive inferences that are generally true for the class. These inferences, if confirmed by the KE, result in an explicit updation of the system's knowledge about the larger class.

In order to attain this goal a representation framework has been devised for the library. The representation scheme, called Visual Semantic Network (VSN), is described in the following section. Essentially, in VSN every class of objects is represented by a separate cluster. Each cluster contains a two level IS-A tree, whose root is the general object name and the leaves are the specific object names. This tree is followed by a structure created by the combination of the individual PART-OF trees of each actual object. Associated with this structure is the actual physical descriptions of the objects and their parts and subparts. The IS-A tree and the PART-OF complex together form the symbolic concept space of visual objects, while the description structures are concrete feature space for the same. Learning thus consists of a change in the concept space, by looking for specific configurations in the concept space itself, or in the corresponding feature space. Connell and Brady [1] applied a generalization technique to create general object models from exemplars. Although their system is the basic inspiration of this work, their representation and generalization methodology is very different from ours. In section 3, learning is defined more precisely and heuristic algorithms for learning are presented. It is shown that learning in our case, although clearly exemplar based, cannot be exactly classified under any one of the existing learning paradigms, namely, Michalski's induction with preference criteria [3], conceptual clustering [4,5,6], explanation-based generalization [7] and case-based learning as in the Protos system [8]. In section 4, we draw conclusions on the results obtained so far and discuss our future plans.

2 A Brief Account of VSN

2.1 Definition

2.1.1 VSN for a Single Object

The definition of the VSN depends on the following elements:

1. A visual entity is an abstract concept consisting of
   - a name by which it is referred and compared with other visual entities. The names, and hence visual entities are of two types: either they are objects or they are parts.
   - a role-set which is a set of functions executed by the visual entity. An assumption of the domain is that two visual entities having the same name must have the same role-set. In our application domain of mechanical tools, a role specification has the form \([\text{action, receiving agent}]\), where the interpretation is "Some external agent performs action on receiving agent using name (of visual entity)." In our domain there is a special receiving agent called self referring to the object itself. Thus the role of the
2. A spatial relation is a binary constraint with a set of free parameters, used to specify how two spatially related descriptions constrain each other. It is assumed that the set of such binary relations is fixed at the time any given universe is defined. The free parameters encode numerical information necessary to specify geometric, topological and measure-theoretic restrictions. For example, is concentric with \((\alpha, \beta, \delta)\) may be a spatial constant between two contours, to mean that their common axis is given by \(\alpha y + \beta x + \delta = 0\). In the previous paragraph, we used the expression "positionally equivalent" to mean two distinct descriptions \(d_1\) and \(d_2\) such that their entire neighborhood (consisting of connected spatial relations and the distant descriptions) is otherwise identical. Again two spatial relations may be compared by a function \(\text{sr} \cdot \text{matches}\).

3. Two partial ordering relations \(\preceq_p\) and \(\preceq_s\) are used in the definition of VSN. \(\preceq_p\) stands for the PART_OF relation while \(\preceq_s\) for the IS_A relation described above. Unlike many standard semantic networks, both these partial orderings are present here leading to interesting interactions described later. For a single visual object the \(\preceq_p\) generates a tree-structured poset. It is to be recalled that these two relations hold in the concept space (also called name space in VSN terminology) and not in the feature space. There is a simple law of inheritance of role-sets in the PART_OF tree. Thus, if B and C are parts of A then the functions of A is the union of the functions of B and C. The obvious exception to this inheritance are the parts whose role-set involves an internal receiving agent such as self.

4. The unary decomposition operator \(\triangledown\) operates on a description \(d_i\) to produce a graph \(G \subset D \times R \times D\) where \(D\) and \(R\) stand for a set of descriptions and spatial relations respectively. Physically, the decomposition operator takes a coarser description (say the head of a hammer) and splits it into finer descriptions (such as the striker end, the connector and the claw) and their mutual spatial relationships. It is obvious that for a visual entity \(v\), \(\triangledown(v)\) would contain descriptions only for all its immediate parts. The graph thus produced by one decomposition is called a description plane in VSN while successive decompositions of individual descriptions produce a tree of description planes, together called a description structure.

For a pictorial view of this description, see Figure 1.

2.1.2 The General Definition of VSN

Next we shall generalize the above definition to accommodate multiple objects. Treatment of this part will be kept informal for a clearer understanding — for a formal treatment of the same see [9]. Obviously, with insertion the two-level IS_A structure is only augmented at the leaf. The PART_OF structure is affected more seriously. At any arbitrary time, suppose there are \(k\) objects of the same class. For the \(i\)-th object the structure is a tree. However, some nodes of the tree (each node designating a name) would also be shared by the \(j\)-th object, and thus have more than one incoming arcs. It is assumed that a node at a certain level in the \(i\)-th PART_OF tree will not appear at a different level for the \(j\)-th PART_OF tree. If a color is assigned to each object, and the arcs of the PART_OF tree of that object is assigned that color, then the general PART_OF structure becomes an edge-colored multigraph (ECMG).

The description structure for a single object was loosely referred to as a tree of graphs. But a formal treatment requires it to be defined as a variant of a directed hypergraph. The reason is simply that, here only one of a subset of nodes (a single description plane) connects to a complete subset (another description plane) by a single arc (formally, a hyperarc). If the subset is called a hypernode, then the hypernodes and the hyperedges constitute a tree. Now if colors are introduced as in the name plane, the following transformations take place. Some new hypernodes with different colors are introduced. Some old hypernodes expand, such that a part of a description plane of one color gets shared by a description plane of a different color. That is an union operation occurs between the graphs of different colors at any level. There is also a difference operation: a case where two description planes, otherwise equivalent in structure, have a

1 assuming that symmetry is not accounted for in our system.
mismatch at one node. Here, a "dummy" node is created to enforce a structural match of the two graphs, and a fork of bifurcating colors is created along with to depict the specialization. Considering the tree of hypernodes, hyperarcs of multiple colors are introduced between the same hypernodes. Just as in the name plane, strict levels of hierarchy will be maintained for each color, forcing in dummy hypernodes if necessary.

2.2 Insertion in VSN

As already mentioned before, the learning step for VSN is always preceded by an insertion. In implementation these two operations are necessarily interleaved, but we show them separately for convenience. Suppose there are \(k\) object models in the system at this time and the \((k+1)\)-th object is being inserted. First let us describe the procedure for the name plane.

Insertion Algorithm

\[
\begin{array}{l}
\text{if } k = 0 \\
\quad \text{request KE and get the general class name } \delta \\
\quad \text{else} \\
\quad \quad \text{assign color } c_{k+1} \text{ to new object} \\
\quad \quad \text{create new object } P(d_{k+1}) \\
\quad \quad \text{connect it to } \delta \text{ with lsa link} \\
\quad \quad \text{make } c_{k+1} \text{ the current node } n \\
\quad \quad \text{for each part of } n \text{ recursively do} \\
\quad \quad \quad \text{if } \text{PART-OF}(n) \text{ does not already exist then} \\
\quad \quad \quad \quad \text{create node } \text{PART-OF}(n) \\
\quad \quad \quad \quad \text{add link from } \text{PART-OF}(n) \text{ to } n \text{ with color } c_{k+1} \\
\quad \quad \quad \quad \text{make } \text{PART-OF}(n) \text{ the current node } n \\
\quad \quad \text{complete function inheritance by clarification from KE if required} \\
\end{array}
\]

Notice that the first level of parts together with the objects constitute a bipartite graph, while the rest of the levels form forests of trees rooted at the first level parts (see Figure 1). For the description structure insertion is a little more involved. In the previous sections we have already introduced the notions of forking and dummy nodes in VSN. The basic operation with respect to a dummy node \(d^*\) is a procedure called Match-or-Fork. In this procedure we use the notation \(P(d)\) to denote the set of participating colors for the description \(d\). Just as the name of a visual entity may be shared across several visual objects, descriptions are also shared, especially for parts that are generic for the entire class.

Procedure Match-or-Fork \((d_i)\)

\[
\begin{array}{l}
\text{if } d^* \text{ corresponding to the name of } d_i \text{ exists then} \\
\quad \text{if } \text{matches}(d_i, d_i) \text{ for any } d^* \text{ which is a specialization of } d^* \\
\quad \quad \text{then} \\
\quad \quad \quad \text{add color } c_{k+1} \text{ to } P(d_i) \\
\quad \quad \quad \text{add color } c_{k+1} \text{ to link}(d_i, d^*) \\
\quad \quad \text{else} \\
\quad \quad \quad \text{create a new } d_i \text{ node with } P(d_i) = c_{k+1} \\
\quad \quad \quad \text{create a new link}(d_i, d^*) \text{ with color } c_{k+1} \\
\quad \quad \text{else} \\
\quad \quad \quad \text{create dummy node } d^* \\
\end{array}
\]

This procedure forms the basis of the Merge Plane algorithm, that takes the input description plane \(D_{in}\) and attempts to merge it with an existing model plane \(D_m\). The corresponding model plane is detected by traversing the decomposition trees of both the input and the model in a top down fashion and locating the requisite description plane from the immediate part-set of any name in the previous step. This is possible because there is a single description plane for all the object-level visual entities of any cluster.

Procedure Merge Plane

let \(D(n)\) be the set containing the parts of node \(n\) in the input

let \(M(n)\) be the corresponding descriptions in \(D_m\)

for each \(d_i\) in \(D_{in}\) Find(corresponding \(d_i\) in \(D_m\))

\[
\begin{array}{l}
\text{match-or-fork}(d_i, d_i) \\
\text{for each relation } r_{ij} \text{ emanating from } d_i \text{ to some } d_j \text{ in } D_{in} \{ \\
\quad \text{if } \text{matches}(r_{ij}, r_{ij}) \text{ in } D_m \text{ then} \\
\quad \quad \text{add color } c_{k+1} \text{ to } r_{ij} \text{ in } D_m \\
\quad \quad \text{else ask KE if new relation should be added between existing model nodes} \\
\}
\]

Further details of the procedure can be found in [9]. In the insertion procedure Merge Plane is invoked during a recursive depth-first traversal of each PART-OF set of each name plane node. Figure 1 shows the steps of insertion on a small example case.

3 Learning in VSN

3.1 Insertion vs. Classification

Most concept acquisition systems contain a classification step by which the system first assigns a new example with an initial class which is then confirmed or rejected by an expert. In contrast, our system uses the initial class assignment and builds up the optimally shared component hierarchy of object concepts. One reason is that the system expects all the insertion to be done during training sessions (not necessarily all at a time though), where single instances of each visual object are presented. Naturally, for the purpose of the library the system retains all the instances presented to it. Why then do we consider the insertion to be part of learning at all? The reason is simply that the tree-traversal procedure associated with our insertion maximally aligns the new instance with those object descriptions that are closest to it by name as well as by descriptions. So essentially, it serves to properly situate the new instance with respect to every instance already learnt. As with the Protos system [5], in a domain such as ours, the intra-cluster variability (poly-morphism in their terminology) is very pronounced, and hence it has to use a heuristic feature-to-feature and feature-to-category matching to perform the classification. Because of the structured nature of our domain, we do not perform a case-based comparison, which requires a bottom-up indexed association between the features of the new instance and the indi-
vidual cases. On the other hand, our traversal is not along a generalization hierarchy as in the case of all incremental conceptual clustering techniques [4,5,6]. Insertion along the PART. OF multigraph effectively produces a partial order on the set of part names by the combination of objects sharing the part. Thus, the multigraph represents a dual partial order, one along \( \subseteq_p \) and the other along \( \subseteq_F(n) \), the participating color set of each name \( n \). This is explained following the algorithms for generalisation.

Our next task is to define learning with respect to the IS.A tree as well as the product space of these two partial orders. This is done in the next subsection. Here we note that this task has three associated difficulties.

1. Since the entire PART. OF tree of a visual object constitutes its property, we need the posets to be closed under unions, intersections and differences for correct property inference.

2. Construction of semantic network-like structure usually involves restructuring a lattice with every insertion such that the resulting structure remains a lattice. This often implies an automatic introduction of intermediate elements into the lattice. For example, if there is a trivial lattice consisting of the entries "person", and "graduate student", introduction of the entry "undergraduate student" forces the intermediate entry "student" after "person". But such intermediate entries may not have any semantic significance in our domain.

3. One must arrive at the same updated structure independent of the order in which exemplars are introduced. Of course, since update is incremental, the updated knowledge can never be claimed to be complete, and may need further restructuring with new exemplars.

3.2 Definition of Learning

One advantage of our application domain is that we can make good use of the following assumptions about our domain.

- Parts having the same name perform the same function, however, the inverse relation is not guaranteed.
- Two nodes having the same name within a cluster will not belong to two different levels.
- Except the case of multiplicity, the one-to-many relation between a part name and a feature always designates a fork.
- Features that fork in the description plane, often explain the functional variability of a part.
- For any cluster there is more sharing of names in the upper levels of the PART. OF structure, and the degree of sharing diminishes as finer parts are defined.

These assumptions may be considered to constitute a very weak domain theory for the VSN world. The process of learning then would take advantage of these and attempt to make stronger cluster-specific propositions. The propositions we are considering would attempt to answer the kind of questions listed below.

- what is the part hierarchy for a typical generalised object (root of the two-level IS.A tree)?
- Are there different sets of parts within a cluster that are functionally equivalent?
- Can the two-level IS.A structure can be split into further subclasses?

3.3 Learning Mechanisms in VSN

First consider the algorithm to define generalization on this structure.

Algorithm 1
Let us call the root of the IS.A tree as \( T \). The element \( T \) is the root of the generalised structure \( G \). Let \( A \) be the set of nodes in ECMG that have incident arcs of all colors. Choose a special color \( \tilde{g} \) to depict the arcs of \( G \). If nodes \( a_1, a_2 \in A \) are connected by all colors, they are connected by color \( \tilde{g} \). For any \( a_i \in A \), if parent.(\( a_i \)) is an object, connect \( a_i \) to \( T \) with color \( \tilde{g} \). Any remaining member of \( A \) is discarded. It can be proven that \( G \) is a supremum semilattice. The generalisation produced by the algorithm is shown in Figure 2.

Although intuitively simple and easy to implement, this algorithm would always tend to undergeneralize, because it fails to see that disjoint unions of single-color branches also produces generalisation. In fact looking for disjoint unions in the PART. OF ECMG, proves to be an interesting characteristic of the domain. The rule, illustrated by Figure 3, may be stated as follows.

Heuristic 1 with each node maintain an inlist and one outlist of colors per immediate part
- if the inlist contains all colors defined so far and all outlists are either equal to the inlist or their disjoint union is the inlist, then include node in the generalisation set \( A \) above
- Notice the node marked "another" in Figure 3. The introduction of this node in the PART. OF tree makes the previously inserted entries just instances of the "another" part, and is equivalent to Michalski’s concept of constructive induction on the general object. A slightly different variation of the same heuristic can bring out the semantics of generalisation even when the disjoint union of the outlists is a subset of the inlist. In that case the above heuristic gets extended in the following way:

Heuristic 1* with each node maintain an inlist and one outlist of colors per immediate part
- if the inlist contains all colors defined so far and all outlists are either equal to the inlist or their disjoint union is the inlist, then include node in the generalisation set \( A \) above else if there exists one outlist \( I \subset \) inlist such that
other outlists are either equal to \( I \) or their disjoint union is \( I \), then the node exhibits a restricted generalization for set the colors in

Let us refer back to our earlier comment on the dual partial order produced by the generalization procedure. Consider any single level of the ECMG structure. As the number of colors incident to a node denotes its "popularity", the parts of the same level can be arranged into a partial order of popularity. Now if restricted generalisation is computed at every level of the ECMG structure, the result is a partial order of popularity defined on an existing poset of aggregation. The significance of this is that the several semantic attributes of a part (such as its uniqueness within any subset of objects in a cluster) can be defined on this composite partial order. We shall not go into these definitions here because they are not part of learning as such.

Apart from the induced nodes, the same heuristic can be used to discover potential subclassification of the initial IS-A tree. In this case, the condition gets modified to the following:

at any level of the ECMG structure after the first level of parts, if the outlists may be grouped to form a partion of the inlist, regardless of the number of colors in the inlist, then there is a possibility that a subclassification exists between the color groups on either side of the partition. This possibility is strengthened if the potential color groups have minimal overlap in the role-sets of the parts undergoing the partition. If the KE confirms that such a subclass exists, then the change is incorporated in the system.

Modifying the IS-A structure can be also brought about by trying to explain the ECMG after insertion based on the domain theory. Consider for example, the case of an adjustable screw driver having a shank holder for holding different types of driving ends. With the addition of the shank holder, the system finds that the post-insertion ECMG subsumes the pre-insertion ECMG. It next observes that the role-set of the new part involves a receiving agent defined within the tree itself, although the role-set of the object adjustable screw driver is not different from that of its siblings. The system reasons that the newest object is a specialization of some kind. Hence it enquires from the KE whether an adjustable screw driver can be a philips or a flat screw driver also. As the KE responds in the affirmative, the system enquires if all philips or flat screw drivers are necessarily adjustable. Then the system suggests to the KE that it suspects a hidden subclass among screw drivers such that the cross-product of the subclasses (i.e., adjustable and not adjustable, vis a vis flat or philips) are all valid entities.

A similar inductive inference can be drawn even if the proper subset condition does not hold, by making use of the configurations of the description structure. Consider for example, the steps in the insertion of several different spanners, as shown in Figure 4. The first instance is entered as a ring spanner and the second as a D-spanner. The system notices the fork in the dummy structure\(^2\) and explains that the IS-A tree has been created to reflect this forking. The third instance, a double ended D-spanner (DEDS) may be entered in two ways. In the first case, it is introduced as a D-spanner also, the system enquires if the new example is a variant of the existing entry. If the response is "yes", it requests the name of the current and the previous variants and creates the chains DEDS \( \prec \) DS \( \prec \) Spanner and SEDS \( \prec \) DS \( \prec \) Spanner. The resulting forking of the handle description is reasoned to account for this subclassification. If a double-ended ring spanner (DERS) is introduced next, the system initially attempts to follow the same sequence of actions until it discovers that the current specialization appears to be explained by forking the same description node as a previous explanation. With this observation, the system tries to compare the two dummy nodes created so far. It discovers that the same set of colors participates independently in both the dummy nodes. It also observes that there is a fundamental difference between the objects participating in these two dummy nodes. While those forking through gripper, essentially form two-node graphs, those forking through handle produce three node graphs. With these two heuristic observations\(^3\), the system asks similar questions to the KE to arrive at the final classification. Note that the discovery of this subclassification needed one more example than the previous case. If, on the other hand, the KE introduced the third instance as a separate entity called DEDS \( \prec \) Spanner, the system would have detected that the present example uses two forks, including that of the gripper, and the details of the feature aligns with one of the previous cases. This would trigger the enquiry procedure to determine if the new example should really contribute to further subclassification. We have shown here that although locally order dependent, the system uses the weak domain theory to eventually correct itself to attain the final classification. This, we claim, is a unique property of our system. As a rejoinder to the above procedure, consider next that the KE introduces a spanner with a hole for hanging it. If the subclass hierarchy has already been formed, the system will start by enquiring whether a spanner with a hang-hole can be a D-spanner, and so forth to eventually extract from the KE that only single-ended spanners can have a subclass based on the presence or absence of hang-holes. If however, the classification does not exist at the time this example is introduced, the system would wait until it observes that the color-set participating in this new object is actually a subset of the color-set participating in the handle-based fork. This would raise the trigger that the hole-based fork has a dependence on the handle-based fork and consequently the final classification will be attained.

Now let us consider another group of cases wh

---

\( ^2 \) This is an essential difference with the previous example where a new node was introduced in the description plane, but no forking occurred.

\( ^3 \) Though there can be more such heuristics, we found that independence of the participating color-sets, and path lengths of objects to be most useful in our application domain.
Figure 1: The representation of two screw drivers after insertion, both in the name plane and the description structure.

Figure 2: The generalisation produced in the name plane by the Algorithm 1. The general part hierarchy are designated by arcs labeled $\rightarrow$. 
Figure 3: The generalization produced in the name plane by Heuristic 1. The arcs labelled $\varepsilon$ are to be read as INSTANCE_OF.

Figure 4: Another case for discovering subclasses: two different dummy nodes are created and are found to be independent of each other. Note in contrast that a restricted subclass is formed when one dummy node is found to be dependent on another. See text for further explanation.
the degree of overlap between previous examples and the new example is minimum, a situation that is likely to occur if a jeweller's screw driver is introduced to an ECMG consisting of regular screw drivers. Let us suppose the parts of a jeweller's screw driver are thumb cavity, body, and end. The body is longer and is also used for holding the tool. The end covers both the tip and a part of the shank. Obviously, the structure trees of the names do not match, but if we look at functions, the thumb cavity and the body will both correspond to the handle, while the body will also correspond to the shank, thus revealing the dual role of the body. Through this correspondence the system discovers that although the parts are named differently, the function tree for both are very similar, leading to the overlap required for generalisation. Now if the functions do not overlap completely, the system associates with each part an index of functional essentiality computed by number of colors × a decreasing function of the level of the part in the ECMG.

The total mismatch is defined by adding the individual indices of the parts not matched. A threshold is defined on this heuristic and any new object failing to meet the threshold is marked for separate treatment. If an object is so marked the system tries to find functional correspondences between the new object and the existing model-base. This is done by using a table with the format

| function | model member | new member |

If it is found that the ECMG structure of the model and the example show considerable overlap when part names are replaced by their role-sets, (as in the example) the system enquires from the KE whether it should create a new subclass or just retain a functional equivalence table (or both), and updates its knowledge accordingly. If however, even the degree of functional overlap is below a threshold, the system enters the consultation of retrieval efficiency during object recognition tasks. Currently we are evaluating the efficiency of retrieval with and without knowledge updation.

References