CONCEPT LEARNING FROM EXAMPLES WITH APPLICATIONS TO A VISION LEARNING SYSTEM

A L RALESCU* and J F BALDWIN

Information Technology Research Centre
University of Bristol
England

ABSTRACT

This paper describes a learning technique based on the use of examples and counter-examples of the concept being learned. The technique blends together the conceptual graphs theory and the support logic programming. Each of these is being used for both knowledge representation and inference.

An application to a vision learning system as well as a discussion of such a system is included.

INTRODUCTION

The idea of concept learning using examples and counter-examples of that concept has been previously considered in cognitive psychology (Rosch and Mervis 1975, and Medin and Schaffer 1978, Smith and Medin 1981), as well as in Artificial Intelligence (Winston 1975).

The use of examples for defining concepts is justified especially for those concepts which cannot be defined in terms of necessary and sufficient conditions, or alternatively for those concepts for which there exist individual instances which do not share all of their features. Such concepts have long been pointed out (Wittgenstein 1953). In fact it can be argued that, except for a limited class of concepts (such as mathematical concepts) most of the concepts which humans learn and use cannot be defined by necessary and sufficient conditions. A general learning technique from examples and counter-examples based on the conceptual graphs theory and support logic programming has been presented in Ralescu and Baldwin (1987).

In the current paper the problems raised by an attempt to apply this technique to vision are investigated. The paper is organized around a practical example, the problem of learning the concept "car" from photographs (examples) of cars. Subsequently the problem of recognizing a car from photographs is considered. A critical discussion of the system obtained is also included.

An intelligent learning system must be able to build a model (here called memory-aggregate, henceforth denoted MA) of a concept based on the examples provided for that concept and be able to update this model at any stage, whenever new examples are provided, or old examples are disqualified (as examples). The learning process should not allow the system to be tricked into considering indefinitely the same example.

*On leave from the Computer Science Department, University of Cincinnati, Ohio, U.S.A.

This work was carried out under contract to Marconi Command & Control Systems, U.K.

In the query phase the system should be able to compare a description of a concept used as a query to the MA of the concept being investigated. The result of this comparison should take into account features shared by the query and the MA, uncertainty and lack of information associated with either one of them. The mechanism used for comparisons should be such that it allows for comparing any two descriptions, therefore making it possible to enforce constraints between various examples.

The use of the conceptual graphs, first introduced by Sowa (1984) provides a powerful means to express the knowledge contained in the examples: each example (query, MA) is represented as a conceptual graph, i.e. a graph containing two kinds of nodes: concept nodes and relation nodes. A comparison between two such graphs will take into account not only the features (nodes) appearing in them but also their structure.

The support logic programming is a programming paradigm developed by Baldwin (1986). It is part of the AI language FRIL (Baldwin, Pilsworth and Martin 1987). The support logic theory allows for evidential reasoning under various forms and degrees of uncertainty. In particular, for the learning technique described in this paper, it allows for partial matching between graphs (a query and MA, or two examples). The effect of using the support logic inference mechanism is that features appearing in the MA, arising from different examples, will increase the support for a given description (query) containing these features to represent an instance of the concept summarised in the MA. It therefore allows for an "interpolation" between examples, for their amalgamation.

1. BUILDING THE MEMORY-AGGREGATE

Consider two examples of car (figures 1 and 2 in Appendix) given as segmented photographs. Each region obtained from segmentation is labeled by the name of the car part to which it corresponds. For example the regions in figure 1 can be labeled as follows: "car_top" for region A, "rear_window" for region B and "body" for region C. The result of labeling these regions are the concept nodes to appear in the graph associated with figure 1: [car_top: *], [rear_window: *], [body: *]. A concept node contains two fields: a type field, to the left of ":" and a referent field to the right of ":" (Sowa 1984).

The relation nodes associated with figure 1 may describe:

1. the relation of each of the concept nodes to the overall image (or the concept being learned, [car :*X], in this case). According to this criterion each region is part of (or contained in) this example of a car.
2. spatial relationships (left_of, right_of, above, below) between the regions (concept nodes).
3. other relations (properties) of these regions: shape, colour, etc.

The conceptual graph resulting from figure 1 is then:

```
(1)  G1: [car: *X]-
                    ->(contains)->[car_top: *]-
                                  ->(left_of)->[rear_window: *],
                                  ->(right_of)->[rear_window: *],
                ->(above)->[rear_window: *],
                ->(above)->[body: *],
        ->(contains)->[rear_window: *]-
                ->(left_of)->[car_top:*],
                ->(right_of)->[car_top: *],
                ->(above)->[body: *],
                ->(below)->[car_top: *],
        ->(contains)->[body: *]-
                ->(below)->[car_top: *],
                ->(below)->[rear_window: *].
```

A "*" in the referent field indicates a generic concept, a "*X" identifies the being defined by the conceptual graph in which it appears.
The following steps are taken to build the MA:

1. the referent fields of the concept nodes are modified to accomodate sets (set coercion according to Sowa 1984).

2. each individual in the referent field of a concept is assigned a weight reflecting the importance of that individual for that concept; the weights assigned within one referent field form a frequency distribution.

3. each relation node is assigned a weight reflecting its importance to the concept node preceding it; the weights assigned to the relation nodes corresponding to the different branches starting out of any concept node form a frequency distribution.

Initially, in the absence of any other information about G1, the relation nodes are assigned equal weights. Similarly, the individuals within the same referent field are assigned equal wieghts. The graph associated with G1 is now:

```
(1')
 G1: [car:*X]-
        ->(contains,1/3)->[car_top:{(*,1)}]-
                ->(left_of,1/4)->[rear_window:{(*,1)}],
                ->(right_of,1/4)->{(*,1)}],
                ->(above,1/4)->[rear_window:{(*,1)}],
                ->(above,1/4)->[body:{(*,1)}],
        ->(contains,1/3)->[rear_window:{(*,1)}]-
                ->(left_of,1/4)->[car_top:{(*,1)}],
                ->(right_of,1/4)->[car_top:{(*,1)}],
                ->(above,1/4)->[body:{(*,1)}],
                ->(below,1/4)->[car_top:{(*,1)}],
        ->(contains,1/3)->[body:{(*,1)}]-
                ->(below,1/2)->[car_top: {(*,1)}],
                ->(below,1/2)->[rear_window:{(*,1)}].
```

Note that a relation may appear more than once in a the same position relative to a given concept node (eg the three occurrences of the relation "contains"). However, the branches corresponding to each occurrence are unique.

Similarly, for figure 2 the associated graph is:

```
(2)
 G2: [car:*X]-
        ->(contains,1/2)->[car_top:{(*,1)}]-
                ->(above,1)->[body:{(*,1)}],
        ->(contains,1/2)->[body:{(*,1)}]-
                (below,1)->[car_top:{(*,1)}].
```

The final step in building the MA is to perform a maximal join (starting out from the node with referent field *X) with a computational extension, of the graphs associated with the examples provided. For the examples G1 and G2 the resulting MA is:

```
(3)
 MA: [car: *X]-
        ->(contains,2/5)->[car_top:{(*,1)}]-
                ->(left_of,1/5)->[rear_window:{(*,1)}],
                ->(right_of,1/5)->[rear_window:{(*,1)}],
                ->(above,1/5)->[rear_window:{(*,1)}],
                ->(above,2/5)->[body:{(*,1)}],
        ->(contains,1/5)->[rear_window:{(*,1)}]-
                ->(left_of,1/4)->[car_top:{(*,1)}],
                ->(right_of,1/4)->[car_top:{(*,1)}],
                ->(above,1/4)->[body:{(*,1)}],
                ->(below,1/4)->[car_top:{(*,1)}],
        ->(contains,2/5)->[body{(*,1)}]-
                ->(below,2/3)->[car_top:{(*,1)}],
                ->(below,1/3)->[rear_window:{(*,1)}].
```

The computational extension corresponds to updating the weights within a graph according to the following formula for updating a frequency distribution: the frequency distribution
(4) {(x1,m1/m), (x2, m2/m), (x3, m3/m)} where
  m1 + m2 + m3 = m
is updated by the frequency distribution
(5) {(x1, n1/n), (y, n2/n)}, where
  n1 + n2 = n
to produce the frequency distribution
(6) {(x1, p1/p), (x2, p2/p), (x3, p3/p), (y, p4/p)}, where
  p = m + n, p1 = m1 + n1,
  p2 = m2, p3 = m3, p4 = n2.

The importance factors (weights) are used as follows:
- Given a branch, the importance of a relation to a concept immediately preceding it is the weight recorded next to the relation (eg, the importance of "contains" for the concept [car: *X] on the branch
[car: *X]->(contains,2/5)->[car_top:{(*,1)}] is 2/5)
- Within a concept node, the importance of an individual is the weight recorded next to the individual in the referent field.
- The importance of a concept to another concept is 0 if there is no branch connecting the two concepts; otherwise, it is the product of the weights attached to the relations which are on the branch connecting the two concepts (eg, the importance of [car_top:{(*,1)}] to the concept [car:*X] is 2/5, (1/5)(1/4), (1/5)(1/4) and (2/5)(2/3) corresponding to the branches starting out of [car:*X] and passing through [car_top:{(*,1)}]).
- The importance of an individual, i, which appears in the referent field of a concept C1, to a concept C2, is equal to the importance of i in C1 multiplied by the importance of C1 to C2.

Other information, such as "location" of the car could have been extracted from the photographs used as examples for car. Suppose that the car in figure 1 is located in a garage and that the car in figure 2 is located on a road. To include this information the graph G1 would be modified as follows:

- Each occurrence of the relation "contains" will have the weight 1/4
- A new branch,

[car: *X]->(is_located,1/4)->[location:{(garage,1)}]
will be included.

The graph G2 will be also modified:

- Each occurrence of "contains" will have the weight 1/3.
- A new branch,

[car: *X]->(is_located,1/3)->[location:{(road,1)}]
will be included.

These modifications will cause the following changes in the graph for the MA:

- The relation "contains" will have the weights 2/7 for two of its occurrences and 1/7 for the third one.
- A new branch,

[car: *X]-
->(is_located, 2/7)->[location:{(garage,1/2), (road,1/2)}]
will be included.

The total importance of the relation contains to the concept [car: *X] is now 5/7 (previous to changes it was 1), the importance of the individual road to the concept in which it appears is 1/2, and to the concept [car: *X] is (2/7)(1/2)=1/7.

The MA "remembers" features, measures of importance for these features and relations between these features.

The process of updating the MA can be resumed at any moment. The only thing needed is a mechanism which guards the system from becoming biased, by the repeated use of the same example. This problem will be treated later.

2. QUERY ANSWERING.

A segmented photograph becomes a query when the system is asked to compare it to the MA of a given concept. The steps taken to answer the query are as follows:

1. The photograph is translated into a conceptual graph; the concept nodes are in terms of unlabeled "region" nodes. The relation nodes describe spatial relationships between these various regions.
2. The regions are labeled by the concept types appearing in the MA for the concept being tested.
3. The graph resulted from steps 1 and 2 above, call it Q, is compared to the MA. The result of this comparison will be a support pair (N, P) whose meaning is :

N = necessary_support(Q represents the concept c/the model of c is MA)

P = possible_support(Q represents the concept c/ the model of c is MA).

The support pairs are to be consistent with the theory of support logic (Baldwin 1986).

The computation of the support pair for comparison (step 3) is considered first. The following algorithms are used in this computation:

Algorithm 1: support for matching branches

Necessary support: Consider only those branches of the MA and Q, starting out of the concept being investigated (referent field *X) which potentially match (that is, have the same relations and same types in corresponding concept nodes, but not necessarily same referent fields). For each such pair of branches let (r,c) denote the first relation/concept pair and let p denote the importance of the relation r as it is recorded in the MA. Let (l,u) denote the support of matching the corresponding occurrences of c (the pair (l,u) to be calculated according to the Algorithm 2 below). Compute p.l and multiply the result by the necessary support of the remaining branch, which is computed by applying this algorithm recursively. Finally, sum up the results over all branches considered.

Possible support: a) With the same notation as above use the previous algorithm with l substituted by u and "necessary support" by "possible support". The result is the possible support obtained from those branches in Q and M which are potentially matching.

b) Consider the remaining branches in MA and Q. For each such branch in MA which contains conflicting information with one of the remaining branches in Q its importance is subtracted from the possible support computed so far.

c) For each branch MA which cannot be potentially matched by a branch in Q its importance is added to the possible support computed so far, to yield the final possible support.

The importance of a branch from MA is computed by multiplying the importance factors associated with the relations along that branch. The branches in Q not considered in (a) and (b) above are ignored.

When two branches to be matched are reduced to a concept node then the support pair for the matching is the support pair for concept matching.

Algorithm 2: support for matching concepts

Necessary support: Sum up the weights corresponding to individuals which appear in both Q and the MA.

Possible support: Sum up the weights corresponding to individuals in Q which conflict with the information in the MA. This is the necessary support against concept matching; subtract it from 1 to obtain the possible support for concept matching.

Both of the above algorithms have the following properties:

i) only information which is identical in both Q and the MA contributes to the increasing of the necessary support.

ii) information which in the MA but not in Q will increase the possible support (the fact that this information is not in Q does not necessarily mean that it cannot be there).

ii) only conflicting information will decrease the possible support.

iv) information which is in Q but not in the MA is ignored since it is irrelevant (indeed were this information relevant it is expected to have appeared already in at least one example and therefore in the MA).

Note that the matching operation is not symmetrical, that is the support to match "Q against MA" is not the same with the support to match "MA against Q". This asymmetry is actually desirable given the interpretation chosen for the support pair. It should be noted that the matching operation can be altered to become symmetric, if needed for other types of applications.

It is worthwhile to note that the supports computed according to Algorithms 1 and 2 above, derived mainly on a intuitive basis will in fact satisfy all the requirments of the support logic theory (Baldwin 1986).

To illustrate the process of computing supports for a query, suppose that G2 as given by (2) is a query and that MA is as given in (3). Suppose that the supports for concept matching are:

(l_car_top, u_car_top) = (1, 1)
(l_body, u_body) = (1, 1)
Then

N2 = (2/5)(1)(2/5)(1) + (2/5)(1)(2/3)(1) = 44/75 = 0.4266
P2 = (2/5)(1)(2/5)(1) + (2/5)(1)91/3)(1) +
(2/5)(1)(1/5 + 1/5 + 1/5) +
(1/5)(1)(1/4 + 1/4 + 1/4 + 1/4) = 1

Next consider a query, call it G3, which is G2 plus the information that it does not contain a rear window, i.e.:

```
(6)  G3: [car: *X]-
         ->(contains)->[car_top:{*}]-
                   ->(above)->[body:{*}],
         ->(contains)->[body:{*}]-
                   ->(below)->[car_top:{*}],
         ->(NOT(contains))-[rear_window:{*}].
```

Then the supports are

N3 = 0.4266 (same as N2)
P3 = (2/5)(1)(2/5)(1) + (2/5)(1)(2/3)(1) + (2/5)(1/5 + 1/5 + 1/5)
+ (2/5)(1/3) + 1/5 - 1/5 = 0.8

Note the difference between P2 and P3.

Use of uncertain concepts:

Uncertain concepts may be present in the query as well as in the MA (although it could be argued that the examples, and hence the MA should not contain uncertainty). Concepts in a query may have been themselves derived via the learning procedure described here, applied at a different level. In this case the process of computing the supports for matching should be able to handle the case when a concept node appears qualified by a support pair. For illustration purposes suppose that the query G4 is like G2 plus the information "contains a part which is left of, right of, below the car_top and above the body, and has support (0.5, 0.75) to be a rear window". That is:

```
(7) G4: [car: *X]-
        ->(contains)->[car_top:{*}]-
                  ->(above)->[body:{*}],
        ->(contains)->[body:{*}]-
                  ->(below)->[car_top:{*}],
        ->(contains)->[rear_window:{*}](0.5, 0.75)-
                  ->(left_of)->[car_top:{*}],
                  ->(above)->[body:{*}],
                  ->(right_of)->[car_top:{*}],
                  ->(below)->[car_top:{*}].
```

To compute the match for the concepts [rear_window:{*}] and [rear_window:{*}](0.5, 0.75) the following FRIL program will be used:

```
((match C1 E1 C2 E2)(is C1 in E1)(is C2 in E2))          :( (l
u)(0 0) )
((is ([rear_window:{*}]) in query ))            :(l1 u1)
((is ([rear_window :{*}]) in ma)                :(l2 u2)
```

where (l u) is the support of matching C1 and C2 computed according to given Algorithm 2. The first clause of this program states that the support of the match between the two concepts C1 and C2 will be obtained from the support (l u) and the supports (l1 u1) and (l2 u2). The support (0 0) will be assigned to the match if the conjunction ((is C1 in E1) (is C2 in E2)) fails, i.e. has support (0 0). For our example (l u)=(1 1), (l1 u1)=(0.5 0.75), (l2 u2)=(1 1). The final support for the match of
[rear_window:{*}] and [rear_window:{*}](0.5 0.75) will be (as given by the FRIL rules of computing supports):
N4 = 0.527
P4 = 0.802

Note the different values for the supports, (N2, P2)=(0.427, 1), (N3 P3)=(0.426 0.8), (N4 P4)=(0.527 0.802).
If G5 is G4 in which (l1 u1)=(1 1) then the corresponding supports are (N5 P5)=(0.627 1).

Uninstantiated queries:

These queries correspond to the segmented photographs (with unlabeled regions) which are presented to the system to be recognized. The question "Does this represent a car" rather than "What does this represent" is asked. Before a support pair can be computed for the answer to such a question the regions describing the photograph must be labeled (step 1 mentioned above). The labels for these regions are suggested by the labels used in the MA for the concept being investigated.

Suppose the following is a conceptual graph corresponding to an unlabeled photograph:

```
(8) Q: [c: *X]-
        ->(contains)->[region_1: #1]-
                  ->(above)->[region_2:#2],
        ->(contains)->[region_2:#2]-
                  ->(below)->[region_1:#1].
```

Note that actually (8) is nothing but G2 from which the labels (concept types) have been replaced by some general label (region). The referents #1, #2 are used to indicate the same instance of a region. The types region_1, region_2 may be thought of as subtypes of a type region).

Various label assignments are possible for the concept nodes of Q. Each of the regions can be replaced by car_top, rear_window, or body types:

Case 1: [region_1:#1] is replaced by [car_top:{*}], [region_2:#2] is replaced by [rear_window]. Then Q becomes:

```
Q1: [car: *X]-
        ->(contains)->[car_top:{*}]-
                  ->(above)->[rear_window:{*}],
        ->(contains)->[rear_window:{*}]-
                  ->(below)->[car_top:{*}].
```

The support of matching with MA is (NQ1 PQ1) = (0.13 1)

Case 2: Same as Case1 for region_1, [region_2:#2] is replaced by [body:{*}]. Then Q becomes:

```
Q2: [car: *X]-
          ->(contains)->[car_top:{*}]-
                    ->(above)-.[body:{*}],
          ->(contains)->[body:{*}]-
                    ->(below)->[car_top:{*}].
```

The support of matching Q2 to MA is (NQ2 PQ2)=(0.4267 1)

Case 3: [region_1:#1] is replaced by [rear_window:{*}], [region_2:#2] is replaced by [car_top:{*}]. Then Q becomes:

```
Q3: [car:*X]-
          ->(contains)->[rear_window:{*}]-
                    ->(above)->[car_top:{*}],
          ->(contains)->[car_top:{*}]-
                    ->(below)->[rear_window:{*}].
```

The support pair for matching Q3 to MA is (NQ3 PQ3)=(0 1).

Case 4: same as Case 3 for [region_1:#1], [region_2:#2] is replaced by [body:{*}]. Then Q becomes:

```
Q4: [car:*X]-
          ->(contains)->[rear_window:{*}]-
                    ->(above)->[body:{*}},
          ->(contains)->[body:{*}]-
                    ->(below)->[rear_window:{*}].
```

The support for matching Q4 to MA is (NQ4 PQ4)=(0.183 1).

Case 5: [region_1:#1] is replaced by [body:{*}], [region_2:#2] is replaced by [car_top:{*}]. Then Q becomes:

```
Q5: [car: *X]-
          ->(contains)->[body:{*}]-
                    ->(above)->[car_top:{*}],
          ->(contains)->[car_top:{*}]-
                    ->(below)->[body:{*}].
```

The support of matching Q5 to MA is (NQ5 PQ5)=(0 1).

Case 6: [region_1:#1] same as in Case 5, [region_2:#2] is replaced by [rear_window:{*}]. Then Q becomes:

```
Q6: [car: *X]-
          ->(contains)->[body:{*}]-
                    ->(above)->[rear_window:{*}],
          ->(contains)->[rear_window:{*}]-
                    ->(below)->[body:{*}].
```

The support for matching Q6 to MA is (NQ6 PQ6)=(0 1).

The answer to the query Q is then chosen to be the description corresponding to the highest support pair obtained, (NQ2 PQ2) = (0.4266 1), corresponding to G2.

This example of an unlabeled query hints at the problems to be expected when such a query is to be answered.

Obviously there is a combinatorial problem: the number of possible label assignments increases exponentially with the number of concept nodes in Q and MA.

Looking at those combinations which have support (0 1) it is to be noted that they contained "wrong" combinations of the parts of a car but this does not seem to matter: a support (0 1) can be obtained by matching any other query which does not contain any of the features of a car.

The combinatorial problem is probably best solved by using heuristics and/or possibly some kind of computer-human interaction to guide the process of label assignment. For instance an algorithm of selecting the labels to be used in an assignment could require at each step to chose only those concept nodes corresponding to weights greater than a preassigned value.

The second problem mentioned above, which amounts to the inability of the query answering mechanism to distinguish between "wrong" features combinations and missing features can be solved by using counter-examples.

3. USE OF THE COUNTER-EXAMPLES.

In this paper a description G, is a counter-example for a concept C if G is a "near miss" in the sense that G may have all (or most) of the features of C, without, however, being an instance of C. It follows that G is either an illegal combination of the features of C, or that it lacks some features which are necessary features for C, or that it has additional features which are sufficient for a concept to be in the class of concepts different from C. The counter-examples are to be handled also by building a memory-aggregate for counter-examples, (Ralescu and Baldwin 1987), henceforth denoted by MA_CE. However, not all the information presented in a counter-example is used to build the MA_CE: only that *information which cannot be matched by the MA* (that is, that information which is not shared by examples).
The MA_CE is then used as follows : a query is matched against it and a support pair, (NQ_C PQ_C), is calculated using the Algorithms 1 and 2. The pair (NQ_C PQ_C) represents the support for a query, given the MA_CE, not to represent an instance of the concept investigated. Then (1-PQ_C 1-NQ_C) is the support for the same query to represent an instance of the concept investigated. Let (NQ PQ) stand for the support pair of matching the MA. The two support pairs are then combined, according to the support logic rule for combining different proof paths (Baldwin 1986) into a final support pair (N P).

For example one may consider that Q3, Q5 and Q6 correspond to counter-examples of a car (each of them describes some impossible combination for the parts of a car). The necessary supports for matching each of these descriptions to the MA were 0, hence none of them has any information in common with any of the examples of a car.
The MA_CE based on Q3, Q5 and Q6 is :

```
(9) MA_CE:[NOT(car): *X]-
          ->(contains,1/3)->[body:{(*,1)}]-
                    ->(above,1/2)->[car_top:{(*,1)}],
                    ->(above,1/2)->[rear_window:{(*,1)}],
          ->(contains,1/3)->[car_top:{(*,1)}]-
                    ->(below,1/2)->[rear_window:{(*,1)}],
                    ->(below,1/2)->[body:{(*,1)}],
          ->(contains,1/3)->[rear_window:{(*,1))}]-
                    ->(below,1/2)->[body:{(*,1)}],
                    ->(above,1/2)->[car_top:{(*,1)}].
```

Suppose now that Q6 is a query. Matching Q6 to the MA_CE the support is (0.33 1) for Q6 not to describe an instance of a car. This is equivalent to support (0 0.67) for Q6 to describe an instance of a car. Combining the support (0 1)

(Case 6 above) and the support (0 0.67) the final support obtained is (0 0.67).

Two other points are considered here:

1. It is obvious that the answer to any query will be a support pair. A "yes/no/I don't know" type of answer is sometimes required or desirable. A decision rule, suitable to the application considered, may be used to obtained more definite answers. Such a decision rule may be as follows:
Let (N1 P1), (N2 P2) be support pairs obtained by matching a query, Q, against a MA, MA_CE (for a concept C), respectively. Let (N P) denote the final support pair obtained by combining (N1 P1) and (1-P2 1-N2). Then decide that:
i) Q is an instance of C if (N P) is in some sense greater than or equal to (N1 P1). In the equality case (N1 P1) must also be different from (0 1); if not, decide that it is uncertain whether Q is an instance of C.
ii) otherwise decide that Q is not an instance of C.

2. It has been mentioned at the begining of this paper that it is necessary to provide a mechanism which would prevent an example from being overused (and hence producing a bias in the system). This problem is shortly considered here. A decision rule according to which a candidate for a new example, call it E, is accepted or not may be as follows:

Compare E to the MA and MA_CE for the concept being investigated. Let (N1 P1) and (N2 P2) be the support pairs corresponding to these comparisons. Then if $N2 > 0$ then E cannot be accepted as an example. Otherwise, E can be an example if (N1 P1) satisfies some constraint, e. g. (N1 P1) is "smaller" than a preassigned (N P).

This decision rule favors examples which bring more new information over those which are very similar to the MA without excluding the latter ones.

A similar decision rule may be implemented to decide if a new candidate, call it C_E, for a counter-example should be accepted:
Compare C_E to MA . Let (N1 P1) be the support pair corresponding to this comparison. Then if $N1 = 1$ (or "very close" to 1) decide that C_E cannot be a counter-example. Otherwise extract from C_E the information which is C_E but not in MA and the information which is conflicting to some information in MA. This part of the C_E is eventually responsible for C_E being a counter-example. Let C_E1 denote the conceptual graph corresponding to this information extracted from C_E. Compare C_E1 to MA_CE and let (N2 P2) be the support pair corresponding to this comparison. Decide that C_E can be a counter_example if (N2 P2) is smaller than a preassigned value (N P). Otherwise, C_E may still correspond to a counter-example, but it does not bring much novelty: it is similar to counter-examples already used.

CONCLUSIONS

A concept learning technique based on the use of examples and counter-examples in the context of a vision learning system has been presented. Only those parts relevant to the vision problem have been incorporated. It should be mentioned that no specific changes, of the general technique, Ralescu and Baldwin (1987), have been made in order to adapt the learning to the vision problem. Hence there may be some defficiencies associated with it (eg the combinatorial problem of the process of assigning labels to uninstantiated queries). It should be stressed that these defficiencies are due to the domain of application, and it is expected that other difficulties may be associated to other domains of application. Solving these difficulties, while very important for the ultimate performance of a system, is not part of the general approach.

The idea of building a memory-aggregate used in conjunction with a query answering mechanism based on the support logic theory seems to provide an adequate model for the learning process.

REFERENCES

1. BALDWIN J F (1986), Support Logic Programming, Int. Journal of Intelligent Systems Vol. 1,pp 73-104. Also in: Fuzzy Sets Theory and Applications, Proc. of NATO Advanced Institute, Ed. Jones A I et al., Reidel Pub. Co.

2. BALDWIN J F, PILSWORTH B W, MARTIN T P (1987) FRIL Manual, Equipu A.I.R. Ltd., Bristol

3. MEDIN D L, SCHAFFER M M, (1978) A context theory of classification learning, Psychological Review 85: 207-238

4. RALESCU A L, BALDWIN J F (1987) Concept learning from examples, University of Bristol I.T.R.C. Technical Report 112

5. ROSCH E, MERVIS C B (1975), Family resemblance studies in the internal structure of categories, Cognitive Psychology 7: 573 - 605.

6. SMITH E E, MEDIN D L (1981) Categories and Concepts, Harvard University Press.

7. SOWA J F (1984) Conceptual Structures, Addison-Wesley.

8. WINSTON P H (1975) Learning structural descriptions from examples, in The Psychology of Computer Vision, (ed. P. H. Winston), McGraw Hill.

9. WITTGENSTEIN L (1953) Philosophical Inestigations, G. E. M. Anscombe, Oxford: Blackwell.

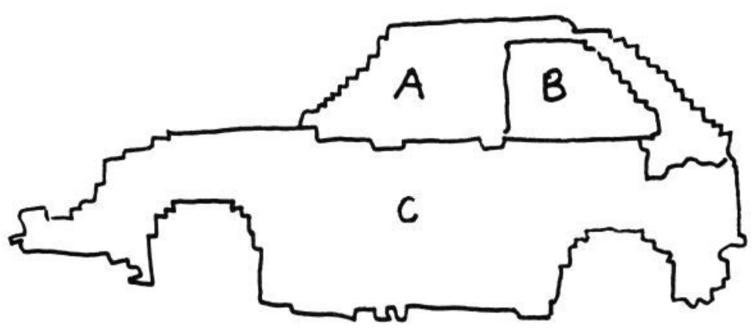

Fig. 1

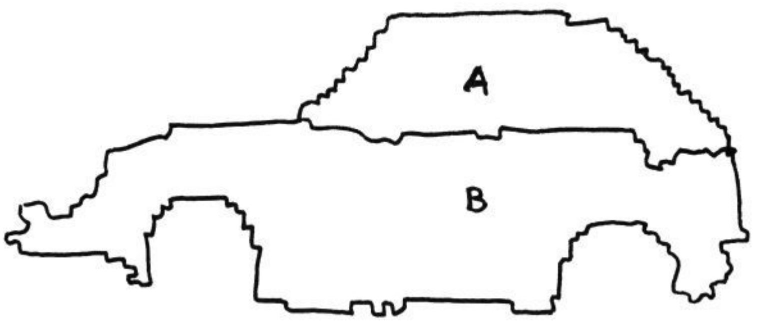

Fig. 2