Rough Set Approach to CBR

Jan Wierzbicki

Polish-Japanese Institute of Information Technology
Koszykowa Str. 86
02- 008 Warsaw, Poland
e-mail:jwierzbi@oeiizk.waw.pl

Abstract. We discuss how Case Based Reasoning (CBR) (see e.g. [1], [4]) philosophy of adaptation of some known situations to new similar ones can be realized in rough set framework [5] for complex hierarchical objects.
We discuss how various problems can be represented by means of complex objects described by hierarchical attributes, and how to use similarity between them for predicting the relevant algorithms corresponding to these objects. The complex object attributes are of different types: basic attributes related to problem definition (e.g. features of object parts), attributes reflecting some additional characteristic of problem (e.g. features of more complex objects inferred from properties of their parts and their relations), and attributes representing algorithm structures (e.g. order and/or properties of operations used to solve the given problem).
We show how to define these particular attributes sets, and how to recognize the similarity of objects in order to transform algorithms corresponding to these objects to a new algorithm relevant for the new incompletely defined object [1,4].
Object similarity is defined on several levels; basic attribute recognition level, characteristic attribute recognition level and algorithm operation recognition level. Dependencies between attributes are used to link different levels. These dependencies can be extracted from data tables specifying the links.
We discuss how to classify new objects, and how to synthetize algorithm for such new object, on the basis of algorithms corresponding to similar objects. The main problem is the generation of rules enabling to create operation sequences for a new algorithm. These rules are generated using rough set approach [5].

1 Problem Representation As an Complex Object

We discuss methods of solving various problems, which can be specified, by means of some hierarchical attributes. Examples of such problems are simple mathematical tasks or finding the way out in the maze.

The problem-case is represented as a complex object (constructed hierarchically) defined by some attributes and its information signature (attribute value vector). Information signature of any object $O \in U$ is defined by attribute value vector, i.e., $Inf_A(O) = \{(a, a(O)) : a \in A\}$, where $a(O)$ is the value of attribute a, on the object O [5].

W. Ziarko and Y. Yao (Eds.): RSCTC 2000, LNAI 2005, pp. 503−510, 2001.

Objects described by the same attributes are represented in information system [5]. A complex object can be described as an uncomposed one - using some general attributes (e.g. specifying general object type - e.g. geometrical figure or maze, its main components - e.g. two triangles, square maze) (Table 1a), or it can be described by some more detailed attributes after decomposition it up to some level, (Table 1b). Values of attributes are binary ones $(0, 1)$: 1 - means the attribute value is present (e.g. crossings in the maze are perpendicular) or its real value (which can be taken from the content of task) is known (e.g. square edge length is known), 0 - means the attribute value is not present or its real value is unknown.

Some of complex object attributes values point out to its subobjects (specified in other information systems) and specify how they are related (Table 1b).

Among object attributes we distinguish attributes, called algorithm (operations) attributes, with values informing if the operation is enabled. Such enabled attributes fire the corresponding operations transforming the object, and let to solve a given problem or to make some decisions.

Table 1a

object	g_1	...	g_n	op_1	...	op_n
O_1	0		0	1		0
O_2	0		1	1		1
O_3	1		1	0		1
...						
O_n	0		1	0		1

Table 1b - (objects of type A&B)

object	a_1	...	a_n	b_1	...	b_m	r_1	...	r_i	c_1	...	c_k	op_1	...	op_l
O'_1	1		0	1		1	1		1	0		0	1		0
O'_2	0		1	1		1	1		1	0		0	1		0
O'_3	1		1	0		1	0		0	0		1	1		0
...															
O'_n	0		0	1		0	0		1	1		1	1		1

$(g_1, .., g_n)$ are general attributes describing the object, $(op_1, .., op_m)$ algorithm attributes (operations) transforming the object.

In Table 1b several subsets of the complex object attribute set are distinguished:

(i) problem definition attributes - basic attributes - $(a_1, .., a_n)$, $(b_1, .., b_m)$,
(ii) relational attributes - declaring relations between component objects of the complex object - $(r_1, .., r_i)$,
(iii) problem additional characteristics attributes - $(c_1, .., c_k)$,
(iv) algorithm operations transforming object - the given problem - $(op_1, .., op_l)$.

By solving the problem we mean finding for a given object all values of attributes corresponding to operations to be executed to solve the problem - e.g. the way out in the maze is found (decision - answer for question if we are in the point of maze exit, is true), or to find value vector of some other missing object attributes, e.g. finding area of geometrical figure.

We solve the given problem by computing step by step algorithm operations for which its value vector is 1 (true) in an order specified by hierarchical structure of tables. This let us to find some required values of object attributes. To execute the particular operation, we need some other object attributes values. Some of them are from the main information system, but some of them must be taken from other information systems for some properly chosen subobjects (Fig.1).
We know attribute values for a, b, A, D, but we are looking for attribute values of c, d.

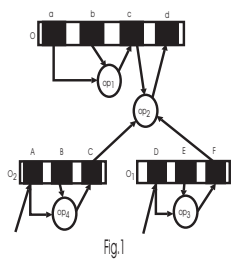

Fig.1

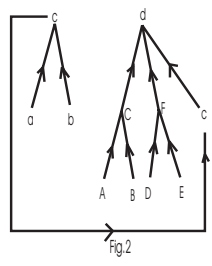

Fig.2

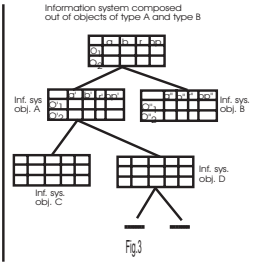

Fig.3

Algorithm operations: $op_1(a,b) = c$; $op_2(C,F,c) = d$; $op_3(D,E) = F$; $op_4(A,B) = C$, can be represented by dependencies [5]: $ab \to c$; $CFc \to d$; $DE \to F$; $AB \to C$, which in turn represent graphically a dependency (Fig.2) $ABDEab \to d$.

One may notice, that to execute the main object algorithm operation op_2 we need attribute values of c, C, F, which are obtained from operations op_1, op_3, op_4. Operation op_1 is the main object algorithm operation, while operations op_3, op_4 are subobjects algorithms operations.

Each complex object (from information system) representing the problem - case, is represented in a hierarchical system (Fig.3).

The components of such system are information systems [5] related in a specific order. Component objects are complex objects (defined hierarchically) or simple object (defined by primary attributes - primary concepts of domain in which the problem is included). Basic attributes are taken from the problem definition (content); other specify (map) subobjects - component objects (from other information systems - Table 2, Table 3). They are formulated (by the expert) in the way which let to specify some attributes and its value vectors from the sets of basic, relational and characteristic attributes for the subobject [2]. Thus we may obtain particular subobject (subobjects) of the complex object.

Table 2 - (objects of type A)

object	a'_1	...	a'_n	b'_1	...	b'_m	r'_1	...	r'_i	c'_1	...	c'_k	op'_1	...	op'_l
O'_1	0		0	1		1	1		1	1		1	1		1
O'_2	0		1	0		0	0		0	0		0	0		1
O'_3	1		1	1		1	1		1	0		0	0		0
...															
O'_n	0		1	0		1	0		1	0		1	0		1

Table 3 - (objects of type B)

object	a''_1	...	a''_n	b''_1	...	b''_m	r''_1	...	r''_i	c''_1	...	c''_k	op''_1	...	op''_l
O'_1	1		0	1		1	1		1	1		1	1		1
O'_2	0		0	0		0	1		1	0		0	1		1
O'_3	0		1	0		1	1		1	1		0	0		1
...															
O'_n	0		1	0		1	0		1	0		1	O		1

Relational attributes declare relations between component objects from which the complex object is constructed. Characteristics attributes are defined by an expert or system rules, and declare an extra description of the complex object, which let the system to input the proper object transformation rules. Algorithm operations are attributes transforming the object in order to find the missing value vector of some attribute [2].

From each information system one may derive some rules defining algorithm attributes (operation) using the basic, relational and characteristic object attributes. The rules are of the form $\alpha \rightarrow op(O)$ for some $op \in OP$, where α is a conjunction of descriptors (a, v) for some $a \in A$ and $v \in V_a$. On the right hand side of the rule more than one operation attribute can appear.

To execute for main complex object an algorithm operation we may need values obtained by subobject algorithm operations. To specify such situation we introduce special information table (Table 4.), consisting of necessary additional information from lower levels of hierarchical structure.

Table 4.

op'_1	...	op'_n	op''_1	...	op''_m	w_1	...	w_i	op_1	...	Op_k
1		0	0		1	1		1	1		0
0		1	1		1	0		1	1		1
...											
0		0	0		1	1		0	1		1

$(op'_1, .., op'_n)$ and $(op''_1, .., op''_m)$ operations for subobject algorithm and $(op_1, .., op_k)$ are operations of main object algorithm; and $(w_1, .., w_i)$ some additional attributes specifying hierarchical structure of main algorithm operations.

Example 1. Let us consider a simple geometrical figure (or maze) (Fig.4). This figure represents a hierarchical system. We may decompose such a figure to component objects (Fig.5). Such object decomposition can be done in different ways (Fig.5a). The most difficult problem is to define (find) the proper decomposition method for the case. Component object may represent a complex object itself, and can be decomposed to its component objects - complex or simple (Fig.6). Component objects of a complex object are related by relation defined by relational attributes - e.g. describing how such component objects are placed towards themselves.

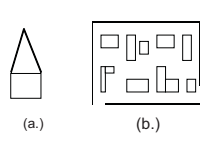

Fig.4

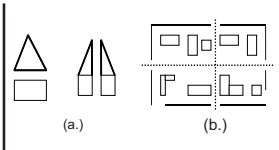

Fig.5

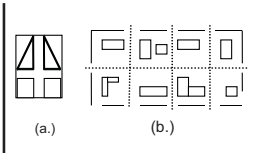

Fig.6

Each component object is labeled by algorithm transforming it - e.g. which let to obtain some of its attributes from others attributes or components (Fig.7).

Each component object can be related to another component object of a different complex object. Such relation let us to transform this object to the other one.

2 Similarity (Closeness) of Objects

One of the main problem to be solved is to construct relevant similarity measures between complex objects. The similarity of objects is relevant, if on the basis of it we may specify a proper set of algorithm attribute value (operations) for

the new object, on the basis of known objects at some level. To do this complex objects must be decomposed up to some level. We compare complex objects (from the same or different information systems) by checking consistency of attributes and its value vectors. We start to check similarity of such objects by comparison attributes and its value vectors for the main complex object and next, by comparison attributes and its value vectors for its component objects. The number of concordant attributes and their values for the main complex objects and its component objects define the level of similarity, and thus some relations on the basis of which one may define similarity.

Let us consider Example 1. Complex objects are decomposed (Fig.4-Fig.6) in order to find the common set of object attributes, which can be measured. If such decomposition is proper, we may define on the basis of concordant attributes and their value vectors relations of similarity degree of objects. We measure similarity on several levels. First it is the similarity of objects from main information system (Fig.4), next it is the similarity of subobjects (from subinformations systems) (Fig.5, Fig.6). Similarity of main objects is more general, while similarity definition for subobjects is more detailed. If the main complex objects are not similar in a sufficient degree on high level attributes, we may try to define their similarity in a more detailed way, by taking into account their subobjects and similarity between them.

Objects from different information systems are described by different attributes and their value vectors, that is why we may define two types of similarity relation:

- consistency of attributes, and consistency of their value vectors,
- consistency of attributes, and inconsistency of their value vectors.

Objects from the same information system are described by the same attributes but with different their value vectors. For such objects there is one relation type - consistency or inconsistency of their value vectors.

One object can occur to be similar to several other objects by using the different similarity relation, depending on a quantity of consistent attributes and their value vectors and level of such consistency.

For any given object O it is extracted from hierarchical information system a set **O** of objects similar to O. Algorithms corresponding to objects from **O** should determine a proper algorithm for O. This is done by learning procedure. However, some simple heuristics can be also used based on similarity of objects (Fig. 8) what will be discussed later.

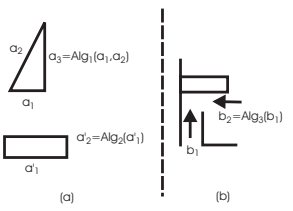

Fig.7

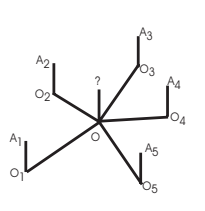

Fig.8

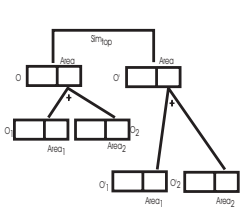

Fig.9

O= $\{O, O_1, .., O_5\}$ is the set of objects similar to O and $A_1, .., A_5$ are algorithms

transforming objects $O_1, .., O_5$.

Example 2. If on the top level $OSim_{top}O'$ (where Sim_{top} - similarity on top level) and the value of attribute e.g. Area for O has been computed by decomposition of O into O_1 and O_2 and by performing the addition of $Area_1(O_1)$ and $Area_2(O_2)$, then we try to find first a decomposition of O' into O'_1 and O'_2 and next to compute its area, using the same operation of addition (Fig. 9). In some cases, for O'_1, O'_2 we search on corresponding levels of decomposition for similar objects and we follow the decomposition procedure for them.

3 New Object and Its Decomposition

Let us consider a new complex object. Now, the main problem is how to construct hierarchical structure of component objects for the new object by taking into account its similarity to the others objects and relations between extracted similar objects on the basis of their attributes. The new object is matched against complex objects represented in our knowledge base. The main goal of general strategy is to isolate possible component objects, by using similarity to others component objects and relations between them. In this way objects similar to a given new object in a satisfactory degree are extracted. On the basis of the extracted complex objects the proper operations of algorithms are selected: Attributes of extracted object and rules for the information system to which new object is assigned, point out operations of algorithm transforming such new object.

Here one can find the problem that not all values of attributes of the new object are known. Let us consider the rule $\{(a, a(O), (b, b(O)), (c, c(O))\} \rightarrow op(O)$. For a new object attributes can be known only to $(a, a(O))$ and $(b, b(O))$. There is a problem if the system should start operation $op(O)$ or not. We may try to solve such problem by taking into account some attributes or rules for the subinformation systems for component objects. Considering subinformation systems one may find the missing attribute $(c, c(O))$ which let to start the operation $op(O)$, or just to skip this operation, and execute successfully the following operation of the main object algorithm. In some of the problems some missing attributes can be skipped, e.g. going trough the maze - we may try to find another way, but in some problems the missing attribute can not be skipped easily, e.g. in mathematical tasks - we have to find particular attribute value vector in order to solve the whole problem. That is why for some objects we have to declare very precise algorithm, with all attributes defined precisely (e.g. mathematical tasks), but for some objects we may declare more general algorithm with more general attributes (e.g. maze problem) which can be modified during its execution.

Example 3. We are looking for similar object to the new one. This step in CBR cycle [1] is called Retrieve. Let us consider a new complex object - new geometrical figure or maze. To assign the new proper algorithm for such object we have to decompose it up to some level. First we try to find the most similar main

information system for the new object, taking into account general problem definition attributes - basic attributes and general relational attributes - declaring general relations between component objects of the complex object. If some algorithm operations - transforming the new object are known, e.g. we have some experience in going trough the new maze - we passed some its corridors successfully, we take them into account as well, while the most similar information system for the new object is assigned. Next we try to specify the most similar object or objects from the chosen information system for new object, taking into account basic attributes, detailed relational attributes, problem additional characteristics attributes and some known algorithm operation - attributes. The object or objects with maximum number of attributes and their values vectors consistent with attributes and their values vectors for the new object is chosen. If the most similar object or objects for the new object are extracted, we specify component objects (by its attributes) of the complex object. To do this we consider information systems for the component objects. In this way the most similar complex object (objects) is chosen for the new object. Known attributes for the new object and rules (obtained from information system to which the new object is assigned) defining object algorithm attributes let us to obtain some algorithm attributes (operations) for the new object (Fig. 8, 9).

It can happen that some chosen operations for the new object will not return the expected attribute values. In such case these "wrong" algorithm operations must be corrected and new missing operations specified. These steps in CBR cycle are called Reuse and Revise.

Let us consider the new algorithm Alg_n (defined by the rules) for the new object. $Alg_n = (op_{n1}, op_{n2}, .., op_{nm})$. We start to perform the operations $op_{n1}, .., op_{nm}$. To execute some of the operations we may need values got from previous operation or operations. If such needed value is missing the next operation can not be executed. In such situation we try to find the missing value (e.g. the edge length of some geometrical figures). To do this we consider another information system (sub-information system) (Table 2 or 3) for the component object of the complex object (we perform algorithm for the component object from which we try to get the missing value). If the missing value is obtained, we perform next operations of the main algorithm, if not we must modify some operations of the main algorithm.

For a given object with a strict structure, e.g. some mathematical tasks, to execute corresponding to this object algorithm it is necessary to perform all operations, and that is why we need all attributes value vectors. If some values are missing and if we can't obtain it from sub-information systems, we can't execute the operation. For some another objects, e.g. maze, if some attributes value vectors needed to execute the algorithm operation are missing, we may try to skip such operation, and execute another one. To do so, we must sometimes return to some already executed operations, and next perform some other operations. For example, if we can't pass the chosen corridor in the maze, we must go back to the corridors crossing and choose another corridor. In this way we correct the wrong algorithm operations.

4 Conclusions

We have presented the main idea on which a software system for problem solving is under the development.

We have discussed how to represent the various problems by means of complex objects represented by some hierarchical attributes, and how to use similarity between them for predicting the relevant algorithms corresponding to these objects.

The most difficult problem is the proper decomposition of the new complex object. On the basis of object attributes and their value vector we may predict similarity of the new object to the known ones. The level of such similarity specifies chances for developing the proper algorithm for the new object. Here we may notice three categories of the new objects: those which are similar to the known objects in a satisfactory degree, partial satisfactory degree and unsatisfactory degree. Any object similar in a satisfactory degree to the known objects allows to construct a correct algorithm. For an object similar in a partial satisfactory degree there is only a chance to construct a correct algorithm. Finally, for any objects similar in an unsatisfactory degree it is not possible to construct a correct algorithm.

We have distinguished at least two types of objects, those which are specified by some precise attributes - e.g. in case of mathematical tasks, and those which are specified by less precise attributes - e.g. in case of maze problems. For these two types different types of algorithms must be created.

We have outlined methods of retrieving similar objects for the new case, and reusing known algorithms for new objects using ideas of CBR cycle [1].

Acknowledgment. The author is due to thank Professor Andrzej Skowron for formulating the subject and for his numerous discussions and helpful critical remarks throughout the investigation.

References

1. A.Aamodt & E.Plaza (1994). *Case Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches.* AI Communications.
2. K.Hammond (1986). *CHEF: A model of case-based planing.* In Proc. of AAAI-86, Cambridge MA: AAAI Press/MIT Press.
3. J.Kolodner (1993). *Case Based Reasoning.* Morgan Kaufmann.
4. J.Wierzbicki (1998) - *"CBR for complex objects represented in hierarchical information systems"*, (procedings First International Conference - Rough Sets and Current Trends in Computing, Warsaw 1998, Springer-Verlag Berlin Heidelberg.
5. Z.Pawlak (1991). *Rough Sets. Theoretical Aspects of Reasoning about Data.* Kluwer Academic Publishers, Boston, London, Dordrecht.
6. L.Polkowski, A.Skowron, J.Komorowski (1996). *Approximate case-based reasoning: A rough mereological approach.* In: H.D.Barkhard, M.Lenz (eds.), Fourth German Workshop on Case-Based Reasoning. System Development and Evaluation, Informatik Berichte 55, Humboldt University, Berlin, 144-151.