A Proposal for Including Behavior in the Process of Object Similarity Assessment with Examples from Artificial Life

Kamran Karimi [1], Julia A. Johnson [2] and Howard J. Hamilton [1]

[1] Department of Computer Science
University of Regina
Regina, Saskatchewan
Canada S4S 0A2
{karimi,hamilton}@cs.uregina.ca

[2] Département de mathématiques et d'informatique
Université Laurentienne
Sudbury, Ontario
Canada P3E 2C6
julia@cs.laurentian.ca

Abstract. The similarity assessment process often involves measuring the similarity of objects *X* and *Y* in terms of the similarity of corresponding constituents of *X* and *Y*, possibly in a recursive manner. This approach is not useful when the verbatim value of the data is of less interest than what they can potentially "do," or where the objects of interest have incomparable representations. We consider the possibility that objects can have behavior independent of their representation, and so two objects can look similar, but behave differently, or look quite different and behave the same. This is of practical use in fields such as Artificial Life and Automatic Code Generation, where behavior is considered the ultimate determining factor. It is also useful when comparing objects that are represented in different forms and are not directly comparable. We propose to map behavior into data values as a preprocessing step to Rough Set methods. These data values are then treated as normal attributes in the similarity assessment process.

1 Introduction

Data is usually considered simply the raw material to be processed. In this view, one receives the data, possibly from a database, looks at it, maybe modifies it and then returns it to a database if needed. In this view, there is a clear separation between the code that processes the data, and the data that is being processed. However, there are problems when "what the data can do," and not "the way they look," is of real interest.

In this paper we propose allowing objects to have behavior, and show that this opens the door for Rough Set [7] techniques to be applied to fields such as Artificial Life [5]. The method suggested in the paper involves representing behavior as a single data value or a set of data values for input to standard Rough Set methods for classification and decision making. These are then treated as if they are constituent

parts of the objects. This preprocessing step allows us to retain compatibility with traditional Rough Set methods.

Assessing the similarity of two data sets, also commonly referred to as objects, without necessarily having any thing to do with the Object Orientation principles, is an important and common operation. Classification of objects is one example of the usefulness of measuring similarity. A concept is expressed by a set of objects that incorporate that concept. In the presence of uncertainty, Rough Set bounds this target set H by two sets, a lower approximation $\underline{H}$, and an upper approximation $\grave{H}$ such that we have $\underline{H} \subseteq H \subseteq \grave{H}$. The Rough Set theory has found many practical applications. Similarity measures include graph measures of Semantic Relatedness [3] for disambiguation of natural language expressions, Correlation measures [1] for calculating the relatedness between word pairs, and Information Theoretic techniques [2] for measuring object associations.

When using Rough Sets to assess the similarity of two objects, researchers usually focus on the parts that make up those objects. Let $x = o(x_1, x_2, \ldots, x_k)$ denote an object x constructed from sub-objects $x_1, x_2, \ldots, x_k$. The usual approach involves computing a function f to measure the similarity of two objects x and y in terms of the similarity of their components, i.e., similarity$(x, y) = f($similarity$(x_1, y_1), \ldots,$ similarity$(x_2, y_2), \ldots,$ similarity$(x_k, y_k))$. This is a static approach. Because each part of an object can have behavior (like a function, which has a source code, and also a run-time behavior), we wish to include this behavior in the process too, essentially along the lines of Object Oriented programming. This is a dynamic approach.

The rest of this paper is organized as follows. In section 2 Artificial Life is briefly introduced and the reader is told why classification methods that use the verbatim values of an object are not of much use there. An example of when behavior is of the ultimate importance is provided. Section 3 presents a guideline to measure behavior and translate it to a single value, or a set of values, which can then be used by ordinary Rough Set techniques, thus retaining compatibility with existing methods and application software. Section 4 concludes the paper.

2 An Artificial Life Problem

Artificial Life is concerned with the study of systems that behave as if they are alive. In most cases the systems are pieces of software, usually called *creatures*, that *live* in an artificial environment. Each creature can be considered a plan that when executed, affects its environment. A simulator can generate new creatures from scratch randomly, or by applying genetic operations of mutation and crossover to existing creatures. Rules of the environment are enforced on the creatures, and pre-defined fitness measures are used as a guide in creating the next generation. Thousands of generations are tried, and the creatures usually evolve to display certain characteristics that help them survive by conforming to the rules of the environment as much as possible. The rules determine the physics of the artificial world, and dictate how “normal” the creatures will behave when compared to the real world. Considering the random elements present in this process, it is no wonder that *spontaneous* emergence of behavior is one of the key characteristics observed in an

Artificial Life environment. It is usually very hard to predict how the creatures will evolve. One usual behavior is that *herds* of creatures show up. Members of each herd have a lot of resemblance to each other, and differ substantially from members of other herds. Artificial Life techniques have been used to *breed* programs that perform useful functions [4].

In this paper, *behavior* is defined as the side effects of *interpreting* data. This interpretation is domain dependent, and can for example be the same as the execution of code produced by an automatic code generator. The definition of behavior can be generalized to include static data too. If there is an easily detectable relationship between the representational format of an object and the effects of its interpretation in the environment, then there is no need to interpret the object. If the data are not interpreted, then behavior is defined as their verbatim values.

If the simulated environment is non-trivial then there is no direct correspondence between the source code of a creature and its behavior. The reason is that in a non-trivial system, on one hand there is more than one way to cause the same effects, and on the other hand executing seemingly similar, but not identical pieces of code can have very different results. In general, a behavior measurement procedure might have to be told to look for specific patterns of interest in certain locations of the system. This problem is greatly reduced when moving to object oriented programming, where global data is more contained and manageable, and completely disappears in a functional programming environment, where global state does not exist.

Comparing behavior is of paramount importance in fields like Artificial Life. One concrete problem is the classification of creatures produced automatically. Because there are thousands of creatures at any time in an Artificial Life simulator, and their behavior may change from one generation to the next, it is very difficult to do the classification manually. One example problem is the classification of the creatures into hunters and non-hunters. Consider an imaginary artificial world, where plant food is created randomly by the simulator. The simulator ages the creatures at regular intervals, which makes them weaker. When they have passed a threshold of weakness, they die and are converted into plants. Creatures all start as peaceful vegetarians, but after a while some may begin developing the traits associated with hunting, like attacking others. This results in the attacked creatures becoming weaker, and thus dying sooner. Such behavior could develop simply because it may be rewarding for the creatures that display them: As the number of competing creatures reduces, there is more food to eat. After a while, they may learn that it is a good idea to hang around weak animals. Still another trait would be to attack weaker animals and then wait, which would probably be the most rewarding behavior.

In this example, there is no explicit hunting behavior, because all the creatures do is eat plants. However, their behavior in the last case may very well be considered to closely resemble that of hunters. Behaviors like attacking other creatures (which makes them weak), waiting near old creature (because they will die in a short time), and moving fast (to chase other creatures) are some of the condition attributes that can be used to help in the classification of the creatures into hunters and non-hunters. Another attribute, creature size, is of doubtful value, but can be considered if the expert looking at the simulator thinks there is a correlation between it and hunting. Using a Rough Set paradigm, one can come up with Table 1 for the creatures of this artificial world. Note that there are no variables anywhere in the simulator to tell us if

a creature attacks others, or waits near old creatures, or moves fast, and the values should be extracted from the creatures' behavior.

Condition Attributes					Decision Attribute
#	Attacks?	Waits near old creatures?	Moves fast?	Creature size	Hunter?
1	yes	yes	yes	small	yes
2	yes	yes	no	small	yes
3	no	no	yes	big	no
4	no	yes	yes	small	no
5	yes	no	yes	big	yes
6	no	no	no	small	no
7	no	no	no	small	no
8	no	yes	yes	small	yes

Table 1. Condition attributes used to determine if the given animals are hunters

Table 1 uses intuitive notions about how a hunter should behave. For example, it is clear that animals 1 and 2 are smart hunters. They attack others, and wait near old (weak) animals, which increases their chance of finding food in a short time, as old animals die sooner. This does not necessarily mean that they hang around the same creature that they attacked, though. Animal number 5, on the other hand, is a stupid hunter because it does attack others, but being a fast mover, does not wait to use the results of its efforts. Animal 4 can be compared to a vulture. It does not attack others, but does wait near old animals. Animal 8 also acts like a vulture, but is classified as a hunter, which is counter-intuitive. This could be the result of an error on the part of the expert who did the classification.

The above table gives the following indiscernability classes: {1}, {2}, {3}, {4, 8}, {5}, {6, 7}. Following Standard Rough Set techniques gives us $\underline{H}$ = {1, 2, 3} and $\bar{H}$ = {1, 2, 4, 5, 8}. If we change the value of creature size for creature 8 from big to small, then we get the following indiscernability classes: {1}, {2}, {3}, {4}, {5}, {6, 7}, {8}. Deleting the size attribute gives the original indiscernability classes. This hints that creature size is redundant. This is also intuitive, as in nature the physical size does not determine if an animal hunts others.

3 Mapping Behavior

The usual way of comparing two objects is to directly compare the values of their corresponding parts, and then use some statistical or heuristic function to come up with a measure of similarity. This method is used in many applications. Using a compatible way to measure behavior will enable us to continue to use the same programs and methods. This can be achieved by the introduction of a mapping function which takes interpretable data, and produces a value or a set of values. These can then be used in the similarity assessment procedure. In general the result of interpreting the data may depend on the global state, and the interpretation may change this state. More formally, we define a function ¦ such that:

- ¦(σ, α) = {σ, α} where α is a data structure that is not interpreted and σ is the global state. The global state does not change and the return value is α itself.
- ¦(σ, β) = {σ', ς} where β is interpretable data, and ς is a measure of changes resulting from interpreting β. σ is the starting state and σ' is the resulting state.

This ensures that the same method can be applied to objects with and without inherent behavior. It can also be applied when an object has behavior that is to be ignored for some reason, in which case b is treated like a. Mapping behavior back to the form of data values (a or s) makes it unnecessary to introduce new terms and techniques, and allows us to retain compatibility with existing methods.

$\dagger$ is domain dependent and should be defined by the experts of the domain. An example for automatic code generation in the functional programming paradigm is that $\dagger$ is simply the result of executing the function b. For a neural network, $\dagger$ provides the input and allows the network to produce its output.

In Table 1, the value of a condition attribute such as "Attacks other creatures" is obtained by a function $\dagger_1(j_1,b)$, with j_1 being the global state of the simulator at the time the function starts execution, and b being the representation of the creature. The result is a possible change in the simulator (leading to the global state j_2), plus a return value from the set {yes, no}. Similar functions ($\dagger_2$, $\dagger_3$, …) should be used to get the other condition attributes.

4 Conclusion

We have suggested taking a broader look at data in the similarity assessment process. We propose allowing data to have behavior, and using this behavior to measure the similarity of two objects. The main point to consider is that comparing parts of an object based solely on their data values may not reveal the complete picture. When the behavior of an object is more important than its representational format, then the data should be interpreted and the results should be included in the similarity assessment process. The conceptually simple technique of expressing behavior in terms of the results of its execution allows for the easy addition of behavior to existing similarity assessment systems. This makes it possible for standard, well-understood methods to be applied to domains such as Artificial Life, where systematic ways of comparison and classification are lacking.

References

1. Dent, M. and Mercer, R. E., A Comparison of Word Relatedness Measures, *Pacific Association for Computational Linguistics*, pp. 270-275, 1999.
2. Jiang J. J. and Conrath, D. W., From Object Comparison to Sematic Similarity, *Pacific Association for Computational Linguistics*, pp. 256-264, 1999.
3. Johnson. J. A., Semantic Relatedness, *Computers & Mathematics with Applications*, pp. 51-63, 1995.
4. Koza, J. R., Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, *Stanford University Computer Science Department Technical Report STAN-CS-90-1314*, p.117, June 1990.
5. Levy, S., Artificial Life: A Quest for a New Creation, Pantheon Books, 1992.
6. Marven, C. and Ewers, G., *A Simple Approach to Digital Signal processing*, John Wiley & Sons, 1996.
7. Pawlak, Z., Rough Sets: Theoretical Aspects of Reasoning About Data, Kluwer Academic Publications, 1991.