Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp333-341)

Information Retrieval Using Analogical Reasoning

KHALIL SHIHAB AND HAIDER RAMADHAN
Department of Computer Science, Box 36, Sultan Qaboos University, Al-Khod, 123 Oman,

Abstract: - This work describes an integrated tool based on analogical reasoning for retrieval and presentation of the data in a summarized form acceptable to computer managers and to the people interested in computer performance. The underlying technique uses exact and fuzzy reasoning to retrieve computer systems that reveal similar performance characteristics. We tested the tool on two different domains. The first is a simple domain consisting of taxonomic properties of some classes of animals characterized by qualitative and nominal data, while the second domain is a collection of examples from different runs of a computer performance modeling package. These examples are allowed to have qualitative and quantitative properties.

Key Words: analogical reasoning, knowledge-base system, heuristics, fuzzy reasoning

1. Introduction

The analogical-reasoning method which is described here is based on the idea of considering the results from a modeling package as a new example to be matched against existing examples that are collected from several runs of the package for different computer configurations [3]. This is because we expect that not all expert interpretations of the meanings of a model's outputs can be matched with rules that have general validity [4]. An alternative source of expertise is an extensive memory E of special examples and their meaning. Faced with a new example N, it may be possible to estimate a meaning for N by assuming that some suitable description of N relates to an equivalently-phrased description of a component Ei of E in the same way that the meaning of N relates to the meaning of Ei.
Using fuzzy indexing and retrieval allows attributes that are characterized by quantitative values to be converted into fuzzy sets to simplify comparison. For example, the utilization of CPU can be converted into categorical scale (e.g. high, low, moderate). Also, fuzzy sets allow multiple indexing of an object on a single value with different degrees of membership. For example, if the CPU utilization is 60, this can be classified as high with 0.4 and moderate with 0.7, where 0.4 and 0.7 are the degrees that the CPU utilization is classified as high or moderate respectively. Further more, fuzzy sets allow term modifiers to be used to further increase the flexibility in object retrieval.

2. The Use of Fuzzy Sets

In fuzzy sets an object may partially belong to a set, so the set must be represented by a continuous membership function that maps the domain of the set to an interval of [0,1]. For example, the following functions (1-3) and Figure 1 show the membership functions of high, moderate and low utilization [7, 8].

1. $\mu_{high}(x)=(x-x_1)/d_o$ if $x_1 \le x \le x_2$, 0 if $x <x_1$, and 1 if $x>x_2$

2. $\mu_{moderate}(x)=(x-x_1)/0.5d_o$ if $x_1 \le x \le$ *midpoint*,
   $\mu_{moderate}(x)=(x_2-x)/0.5d_o$ if *midpoint* $\le x \le x_2$, $\mu_{moderate}(x)=0$ otherwise

3. $\mu_{low}(x)=(x_2-x)/d_o$ if $x_1 \le x \le x_2$, 0 if $x >x_2$, and 1 if $x< x_1$

Where x_1, x_2, d_o, and *midpoint* are as follows:

$$X_1 = \begin{cases} X_{CPU} = 30 \\ X_{I/O} = 20 \end{cases} \qquad X_2 = \begin{cases} X_{CPU} = 70 \\ X_{I/O} = 40 \end{cases}$$

$$d_o = \begin{cases} d_{CPU} = 40 \\ d_{I/O} = 20 \end{cases}$$

$$midpt = \begin{cases} midpt_{CPU} = 50 \\ midpt_{I/O} = 30 \end{cases}$$

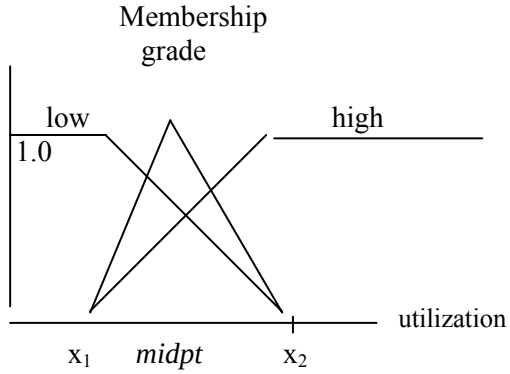

Figure 1: The membership function of high, moderate and low utilization

3. Indexing and Retrieval

Fuzzy indexing and retrieval are useful in domains where objects have quantitative attributes. For objects with qualitative attributes only, indexing can be performed on attributes directly. For example, system performance can be classified as excellent, good, average, or poor (four classes). We can easily index systems by their quality of performance.

3.1. Fuzzification Process

The fuzzification process includes the following steps:

1. When an object is encountered, qualitative attributes are identified.
2. For each quantitative attribute, proper classes are determined based on practical needs.
3. The membership function of each class and its associated α-cuts are determined.
4. Numerical values of each object are converted into proper classes for indexing.

To illustrate the fuzzification process, a running example is used. The context is a Unix system with one multiprocessor CPU and a disk subsystem, which consists of two disks. The pay load consists of three different types: a database service, an interactive service and a backup service. The database service is of a transaction processing type (tp), the interactive workload is of a terminal type (tm), and the backup service is of a batch type (ba). Table 1 shows the current measurements (using an accounting routine and/or a modeling package) collected for this performance problem.

Attribute	***Value***
Number of Devices	3
Workload	multicasts
CPU Utilization	0.65
Dick-1 Utilization	0.58
Disk-2 Utilization	0.06
Memory Occupation	0.14
Type of Largest CPU Process	ba
Type Largest IO Process	tm
Type Largest Memory	ba
Queue Length	2

Table 1: Measurements of a Computer System

Using a frame-based representation, Figure 2 shows a problem instance that is transformed from Table 1.

```
(Problem-Instance
   (kind-of (computer-system multicasts
       CPU-problem I/O-problem))
       (unusual-property (if-needed obtain-unusual))
       (CPU (high/0.88, modrate/0.25, low/0.13))
       (IO-subsystem-uti (unbalanced))
       (bottleneck-recourse (CPU))
       (bottleneck-process (ba))
       (queue-length (if-needed 2))
       (analysis (if-needed generate-analysis))
       (resolution ()))
```

Figure 2: A frame based problem instance

In the transformation of the measurement data in Table 1, the fuzzifier handles the quantitative values that need to be converted into qualitative data. Usually, computer managers classify the CPU utilization into three classes: high, moderate, and low. Using the membership functions, given above, the fuzzifier converts the CPU utilization value 0.65 into membership grades of the respective classes: 0.88 for high, 0.25 for moderate and 0.13 for low. However, if the α-cut is set to 0.5, then the CPU utilization is classified as high/0.88 only.

3.2. Object Representation

Once a problem instance is indexed, four additional attributes are added before it becomes an object to be stored in the database. These additional attributes are: the object number, the *unusual* or the *interesting* property, the problem analysis, and the problem resolution. The data for the first three of these attributes are obtained automatically by using the attachment functions, whereas a computer performance expert adds the value for the last attribute, namely the problem resolution.

The thrashing (when the utilization of all devices is low and the queue length>0) in the model output is an *interesting* property. The *migration* of the cause of a bottleneck from a hardware component, say disk 1, to another, say disk 2, is also an *interesting* feature. Figure 3 depicts these points.

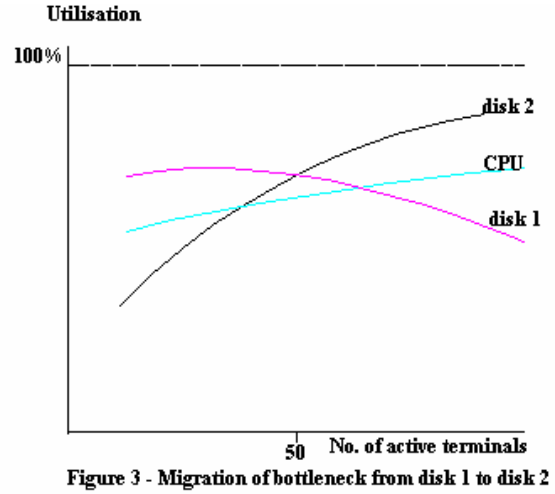

Figure 3 - Migration of bottleneck from disk 1 to disk 2

Instance analysis is generated during object acquisition process. The system uses some fuzzy rules (see section 3.3) in order to automatically generate a performance analysis for the problem instance. For this running problem instance, the generated analysis is as follows.

((stop or replace some batch jobs or programs that need
high CPU time with other jobs or programs that need nothing but I/O)
(Rewrite the programs in more efficient ways)
(Redistribute the jobs to run in another period)
(Balance the load on I/O store devices))

The problem is then referred to a performance expert to identify the major bottlenecks along with their resolutions. Compared with the analysis reached by the system, the expert adds the problem resolution. For this case, the statement "*stop the batch process*" is considered as the solution and therefore added as a value for the resolution attribute. After the problem is solved, the attributes unusual-property, queue-length and analysis become redundant and should, therefore, be eliminated. Other attributes may become irrelevant and their values should be replaced with the wild card character "*". At the final stage an object number is assigned and the object is added to the database.

```
(Example 1
  (a-kind-of (computer-system multi-classes CPU-problem I/O-problem))
     (CPU-uti (high/0.88))
     (IO-subsystem-uti (*))
     (bottleneck-recourse (CPU))
     (bottleneck-process (ba))
     (resolution (Stop the batch process)))
```

Following the same procedures, Table 2 displays several objects in the database.

No.	Bottleneck Process	Bottleneck Resource	Fuzzy CPU Uti	Fuzzy I/O Uti	Memory Wait	Unusual Property	Resolution
1	ba	CPU	high/0.62 modrate/0.25 low/0.13	*	*	*	Stop the batch process
2	*	CPU	high/0.54 modrate/0.54 low/0.45	*	*	*	Optimize the large programs
3	tm	CPU, IO	high/0.3 moderate/0.6 low/0.7	high/0.66	*	*	Reduce the number of active terminals.
4	*	Memory	moderate/0.45	*	high/0.72	thrashing	Upgrade memory
5	tp	IO	*	high/0.54 moderate/0.58	*	*	Add a faster IO device.
6	*	CPU	high/0.57, moderate/0.46 low/0.42	*	*	*	Optimize programs that required high CPU demand.
7	*	CPU	high/0.92 modrate/0.15	*	*	*	Replace CPU
8	*	IO	*	unbalanced	*	*	Balance I/O loads
9	*	CPU	high/0.67, moderate/0.3 low/0.09	unbalanced	*	*	Recode the programs. Redistribute the jobs. Balance I/O load.
10	*	*	high/0.5 modrate/1 low/0.5	modrate/0.3	modrate/ 0.3	*	Satisfactory performance

Table 2: Examples (objects) in a sample database

3.3 Object Retrieval

Given the examples (objects) in Table 2, suppose we want to evaluate the performance of a computer configuration that is described by the following table.

Attribute	***Value***
Number of Devices	3
Workload	multi-classes
CPU Utilization	0.54
Dick-1 Utilization	0.32
Disk-2 Utilization	0.35
Memory Occupation	0.27
Type of Largest CPU Process	ba
Type Largest IO Process	tm
Type Largest Memory	ba
Queue Length	1

Table 3: A computer configuration

After transformation of data in table 2 using the membership functions defined previously, the following problem instance is produced.

(Problem-Instance
(kind-of (computer-system multi-classes CPU-problem))
(unusual-property (if-needed obtain-unusual))
(CPU (high/0.59, modrate/0.8, low/0.4))
(IO-subsystem-uti (*)
(bottleneck-recourse (CPU))
(bottleneck-process (*))
(queue-length (if-needed 1))
(analysis (if-needed generate-analysis))
(resolution ()))

Figure 4: A problem instance

Based on the matching attributes of the problem instance, the object retrieval can easily select the object 1, 2, 6 and 7 from the database to be used as bases for performance evaluation of this new problem instance. There are several ways of finding the most similar object. In this work, we use the following algorithm (similarity measure) [9, 10].

1. The similarity measure, d_q, for an attribute to set to 0 if the attribute's value for the object is equal to the attribute of the problem instance; d_q is set to 0.5 if the attribute's value for the object is a wildcard (i.e. '*'). Otherwise the measure for the attribute is set to 1.
2. The similarity measure for fuzzy attributes is calculated as follows:
   $d_i = \sum_j abs\,(x_{ijk} - x_{ijn})$; where x_{ijk} and x_{ijn} are the grades of attribute *i*, class *j*, for objects *k* and *n* respectively.
3. The similarity measure for the object is the sum of the results obtained from (1) and (2).
   $d_c = d_q + d_i$

Table 4 displays the results of applying this algorithm to the problem instance in Fig 4 and the objects in Table 2. Object 6 is, therefore, the most similar object to the problem instance.

Object	**Fuzzy CPU Utilization**	**Distance**
1	high/0.62, modrate/0.25, low/0.13	0.85
2	high/0.54. modrate/0.9, low/0.45	0.2
6	high/0.57, modrate/0.85/low/0.42	0.09

7	high/0.92, modtare/0.15, low/0	1.38
Problem Instance	high/0.59, modrate/0.8, low/0.4	0

Table 4: Distances between the problem instance of Fig 4 and the candidate objects

4. System Implementation

The system consists of three modules which are: information-transfer module (ITM), object-search module (OSM), and object learning module (OLM). The user may name a destination and a source. In the following sentence: "X is like Y"; X is the destination and Y is the source, which is may or may not given.

4.1. The Information-Transfer Module (ITM)

The goal of this module can be described as follows.

Given:

(i) A statement that identifies the destination and the source objects.

(ii) A statement that identifies the class of the object Y, such as "Y is Z"; where Z is the class of the object Y.

Construct:

(i) An initial-transfer frame (ITF) that contains the properties of the source objects Y. The interesting properties (important and unusual properties) should be given a higher weight.

(ii) A final-transfer frame (FTF) that contains, in addition to the specification of the ITF, the properties of the objects which are listed in the a-kind-of slot of the frame that represents the object Y.

If the a-kind-of relationship has a long list of objects, then a subset of that list is selected. This subset is ended with an object that is allocated at the basic level. The basic level in the hierarchical organization of objects is considered as a level that contains objects after which no further interesting information can be added [5, 6, and 7].

The process administered by ITM depends on fuzzy reasoning; see section 3, and the following heuristics for constructing a transfer frame from a given object:

(a) Select properties that reveal unusual properties of the object. For example, the property *does-not-fly* is an unusual feature for a bird while the *outliers* in the model output are unusual properties of a computer system, see Figures 5 and 6. The *migration* of the cause of a bottleneck from a hardware component, say disk 1, to another, say disk 2, is also an unusual property...

(b) Select important properties. For example, the properties *long-legs* and *long-neck* are important properties of the animal *giraffe*.

(c) Select properties which are not found in the closely related objects.

(d) Select properties that may be given by a user through an interactive dialogue.

(e) Select all slots and values.

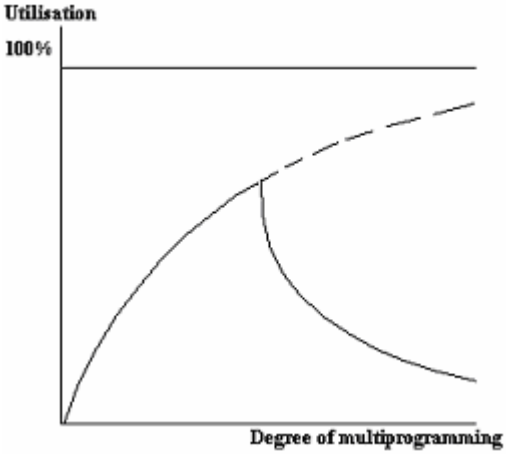

Figure 5: Unusual feature (thrashing)

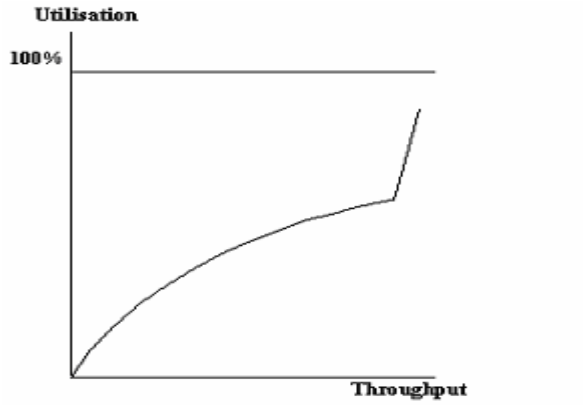

Figure 6: Unusual feature (extreme point)

4.2. Object-Search Module (OSM)

The purpose of this module is to find an object among the offspring of the class of the destination object. Thus, OSM is a search procedure that starts the search from the class of the destination object which is allocated at the same level as the class of the source object.

Computing the target frame requires OSM should pass through the following steps.

(i) Match the properties of the final-transfer frame (FTF) against the properties of each frame at the lowest level of the hierarchical structure (the last offspring of the class of the destination object).

(ii) Compute the distance measure (dm) of the matching results for each slot as follows:

$$dm = \sum P_{Si} \; ; \quad i = 1, 2, \ldots, n$$

$$P_{Si} = P/T; \text{ where}$$

P_{Si} is the matching results for each slot, where s is the slot number,

P is the number of successful property matches, and

T is the total number of the properties of a slot of the target and the FTF.

(iii) Compute the final distance measure by adding the distance measure resulted from fuzzy reasoning, see section 3.

(iv) Repeat (i) to (iv) for each frame in the lowest level of the class of the target frame.

(v) Sort the resulting object-distance pairs according to their measurements.

(vi) Select the object that has the highest measurements.

(vii) Announce that the selected object is the target object.

(viii) Repeat (vi) and (vii) if the user is not satisfied with the result until a satisfactory object can be reached or until no object can be found.

(ix) Call the object learning module (OLM), see section 4.3, to construct a rule when the result is satisfactory and the associated distance measure is high.

4.3. Object Learning Module (OLM)

An important function of OLM is learning new examples. If OLM cannot find a known object matching (fully or partially) a new example, it will prompt the user on this example. It may be considered as an important example to be added to the knowledge base. At this stage, the process of inserting a new example is carried out by the user.

5. Case Study: Computer Performance

The following case study illustrates how the reasoning by analogy technique can be applied to a classification of examples taken from several runs of a computer performance modeling package. The input data to each run is a hypothetical configuration of a computer system. It consists of a workload model and a system model. Defining a computer system model is typically straightforward, since the computer resources correspond directly to the queuing centers. However, to define a workload model, the workload intensity has to be established. This intensity has two aspects: selecting an appropriate workload

type (transaction, batch, or terminal) and setting the appropriate workload intensity parameters for that type. A workload type may consist of multiple classes.

The outputs of a computer package are the values of a computer performance indexes (throughput, response time, utilization, etc.).

The modeling package used for this purpose is a queuing model with multiple classes [3].

As previously mentioned, each example (which is an output of the modeling package) is represented as a frame which refers to its associated input. At present, 30 different examples have been classified and implemented for testing the system. By analogy to the classification scheme of the animals that has been used in this work, a simple method has been adopted for the classification of these examples. This classification is based on whether the system can be described by one or more of the following states: CPU-bound, I/O-bound, over utilized, underutilized, unbalanced, unacceptable response time, acceptable performance, etc. Some of these collected examples are shown in table 2.

The procedural attachment "generate-analysis" to the above frames uses the following template in order to produce a summary analysis for each frame (example) [5].

(analysis
(The important lesson from this run is:
(<$d_{1,1}$><$d_{1,2}$> . . . <$d_{1,n}$>)); diagnostic clauses level 1
(Possible courses of action of action to resolve this problem are:
(<$a_{1,1}$><$a_{1,2}$> . . . <$a_{1,n}$>)) ; actions level 1
{((In addition, it is advisable to
(<$a_{2,1}$><$a_{2,2}$> . . . <$a_{2,n}$>)) or ; actions level 2
(In addition, it is possible to get even better performance if you
(<$a_{3,1}$><$a_{3,2}$> . . . <$a_{3,n}$>))) ; actions level 3
(Because
(<$d_{2,1}$><$d_{2,2}$> .. $d_{2,n}$>)));diagnostic clauses level 2

where $d_{i,1}$, $d_{i,2}$, . . ., $d_{i,n}$ are diagnostic clauses, and $a_{i,1}$, $a_{i,2}$, . . ., $a_{i,n}$ are their associated actions. They are arranged according to their importance level. For instance, the clauses at level 1 are more important than the clauses at level 2.

((Example-3
(a-kind-of (value computer_system))
(has-prop (value CPU disk-1 disk-2 disk-3))
(unusual-prop (if-needed obtain-unusual))
(important-prop (if-needed obtain-important))
(throughput (value 1200 TX/HR))
(uti-CPU (value 0.70))
(uti-disk-1 (value 0.34))
(uti-disk-2 (value 0.12))
(uti-disk-3 (value 0.20))
(analysis (if-needed generate-analysis)))

Figure 8 - Frames representing some examples of computer performance.

Suppose that Example-3 is entered as a new input to the system. Following the reasoning process described in section 3 and section 6, the system selects object 2 from table 2 as the best candidate that matches with Example-3. Consequently, the following analysis will be generated and displayed to the user.

(analysis-2 (The important lesson from this run is that: (The system is CPU bound))
(Possible courses of action to resolve this problem are:
(You are advised to replace some jobs or programs that need high CPU time with other
jobs or programs that need nothing but I/O)
(Rewrite the programs in more efficient ways)
(Redistribute the jobs to run in another period))
(In addition it is possible to get better performance if you:
(Balance the load on I/O store devices))
(Because (The load on I/O store devices is unbalanced)))

6. Conclusion

This work provided further support for the idea of having a database of historical objects along with their analysis, assessment of others are achieved using analogical reasoning. The results given above have been produced utilizing knowledge-based approach, which is implemented by using object oriented methodology. This work aims to demonstrate the use of analogical reasoning for studying computer performance problems, which si ill-structured problem. This and similar types of automation have been considered essential for solving these problems. Analogical reasoning systems have the advantages of being simple and modular. In particular, adding or removing a new object does not affect the available objects in the database. The modularity of these systems enables performance management automation that is custom tailored by end-users based on their own experience.

References

[1] Corcoran, R., *Theory of Mind in Schizophrenia.* In : Penn, D. and Corrigan, P. (Eds.) Social Cognition in Schizophrenia (pp 149-174), Evolutionary Psychology, Volume 3. 2005.

[2] Campbell J. A and K. I. Shihab, *Applications of Artificial Intelligence in Computer Performance Modeling,* Proc. Int., AMSE Conference, Monastir, 1985, 25-28.

[3] Buni, B. and K. I. Shihab, *A Computer-aided Software Tools for Performance Evaluation and Measurements*, Int. Workshop on Expert Systems and their Applications, France, 1990, pp. 105-116.

[4] Charniak, E., at al, "Artificial Intelligence Programming", Lawrence Erlbaum, Hillsdale, N. J, 1987.

[5] Bouchon-Meunier, B. and L. Valverde, *A fuzzy approach to analogical reasoning*, Int. J. Soft Comput. Vol. 3 1999, pp. 141-147.

[6] Winston, P. H., *Artificial Intelligence*, Second Edition, Addison-Wesley, 1984.

[7] D. Dubois, H. Prade (Eds.), *Fundamentals of Fuzzy Sets*, the Handbooks of Fuzzy Sets Series, Vol. I, Kluwer Academic Publishers, Boston, 2000.

[8] Forbus, Kenneth D., Dedre Gentner, & K. Law, *MAC/FAC: A Model of Similarity-Based Retrieval*, Cognitive Science 19:2, 1995, pp. 141-205.

[9] Forbus, Kenneth D., T. Mostek, & R. Ferguson, *An analogy ontology for integrating analogical processing and first-principles reasoning,* Proc. IAAI-02, 2002, pp. 878-885.

[10] Grob, J. and Wood, S., *Modeling Typical Alphabetic Analogical Reasoning*, in Proceedings of the European Conference of the Cognitive Science Society, Osnabruck, 2003, pp. 157-162.