A MULTI-DIMENSIONAL DATA ORGANIZATION THAT ASSISTS IN THE PARSING AND PRODUCTION OF A SENTENCE

W. Faris and K. Cheng

Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

UH-CS-08-04

March 28, 2008

Keywords: Data Structure, Natural Language Processing, Artificial Intelligence, Knowledge Learning System, Dynamic Structure.

Abstract

For knowledge systems that rely on teachings from an outside source to gain its knowledge, proper data organizations are instrumental in managing and applying what has been learned. This paper describes the development of a Multi-Dimensional Data Organization (MDDO). A MDDO stores a collection of knowledge of the same category subdivided into multiple orthogonal dimensions, with each dimension having a number of indices. Each piece of knowledge is identified uniquely by a combination of indices, one index from each dimension. This MDDO has been used to store part of the English grammar, and we describe in this paper how it assists a learning program in both the parsing and the production of an English sentence. We define the functionalities of the data organization, describe its organization, describe how each function is implemented, and analyze their worst-case performance.

A MULTI-DIMENSIONAL DATA ORGANIZATION THAT ASSISTS IN THE PARSING AND PRODUCTION OF A SENTENCE

W. Faris and K. Cheng

Abstract

For knowledge systems that rely on teachings from an outside source to gain its knowledge, proper data organizations are instrumental in managing and applying what has been learned. This paper describes the development of a Multi-Dimensional Data Organization (MDDO). A MDDO stores a collection of knowledge of the same category subdivided into multiple orthogonal dimensions, with each dimension having a number of indices. Each piece of knowledge is identified uniquely by a combination of indices, one index from each dimension. This MDDO has been used to store part of the English grammar, and we describe in this paper how it assists a learning program in both the parsing and the production of an English sentence. We define the functionalities of the data organization, describe its organization, describe how each function is implemented, and analyze their worst-case performance.

Index Terms

Data Structure, Natural Language Processing, Artificial Intelligence, Knowledge Learning System, Dynamic Structure

I. Introduction

In any program system, data organization is required to assist an efficient implementation of the algorithms that solve problems. We describe in this paper a Multi-Dimensional Data Organization (MDDO), which is used in handling the communication problem of a learning program. We have proposed *A Learning Program System* (ALPS) [1] whose goal is to learn knowledge currently learned by human. The focus has been on the development of the memory agent of a multi-agent artificial intelligence program [2] to store knowledge and the relationships among them. Basic capabilities, such as creating a new category, adding objects, attributes, and properties to a category have been provided. We have developed two major knowledge components of categories: hierarchy [3] and definition [4]. Hierarchy allows both generalization and containment relationships among categories to be specified and stored. Definition specifies the necessary and sufficient condition that is used to classify objects as a specific category. Recently, we are developing the communication capability of the learning program so that it can communicate in a natural language, specifically, English. Instead of focusing on the formal language model [5] for English, our approach simply recognizes that grammar terms can structurally be either a sequence or a list of alternatives, together with rules and roles [6]. Our program first learns and stores a subset of the English grammar, and then uses the grammar both to understand and to produce a sentence. The MDDO is developed to store part of this grammar, and to assists in the processing and the production of a sentence. A MDDO stores a collection of knowledge of the same category, where each piece of knowledge is identifed uniquely by a combination of indices from orthogonal dimensions.

Currently our system makes use of MDDO in two ways. First, we use it to store the rules of the grammar under two orthogonal dimensions: usage and application. The usage dimension contains two indices, restriction and choice, and the application dimension has two indices as well: kind and structure. A rule must be classified under one of the four possible index combinations, and rules for each combination are used at a different time during the processing of an English sentence. For example, the "choice/structure" rules for complement are applied before the structure of complement is used. If the main verb is an action verb, then the parsing uses object complement, otherwise, it uses the alternative subject complement. On the other hand, the "restriction/kind" rule for decision question requires that its complete subject not be fulfilled by an interrogative pronoun. The rule is checked after the kind of sentence has been identified as a decision question. Our second use of MDDO involves storing the different categories of each kind of pronoun found in the English language [7]. For instance, personal pronouns can have dimensions such as person, case, and gender. The person dimension comprises the indices of first-person, second-person, and third-person, the case dimension is either plural or singular, and the gender dimension has indices male and female. Therefore, the personal pronoun 'he' is categorized as "third-person/singular/male", and the pronoun for "first-person/plural" is 'we'. The benefit of this organization is obvious when the program wishes to generate a sentence. If the system wishes to refer to the user, the person it is communicating with, and it understands that it is a "second-person" personal pronoun, then 'you' can be chosen easily using the key "second-person".

The rest of the paper is organized as follows. Section 2 discusses the three characteristics for MDDO: the organization capability of the structure, the dynamic nature of the number of dimensions and indices, and the sparse number of entries. It also defines the functionalities of the data organization. Section 3 provides a high level solution to the implementation of the functionalities of this data organization, and Section 4 discusses the space complexity of MDDO and time complexity of its functions. The final section concludes our paper.

II. The MDDO

A MDDO stores a collection of knowledge of the same category. Knowledges in this category are subdivided into multiple orthogonal dimensions, with each dimension having a number of indices. Each piece of knowledge is identified uniquely by a combination of indices of orthogonal dimensions. The intention of creating the MDDO was to give the user an easy to manage data structure for handling the natural language processing problem. This includes being able to organize information in a dimension friendly manner, allocate memory dynamically when needed, and provide a solution to handle a sparse amount of knowledge as the number of dimensions increase.

Organizing information in a dimension friendly manner is a vital capability of the MDDO. This capability is provided because, as evidenced from our examples, knowledge within a category is distinguished by a set of indices that reflect the properties of the knowledge. For example, the properties of the personal pronoun 'I' is first-person, singular, subject, and non-possessive. When knowledge objects are arranged by a series of dimensions, they can be retrieved using the set of indices that identify them. In addition, one may not always be cognizant to an ordering of the indices. For example, someone may refer to the personal pronoun 'him' as "third-person/singular/possessive/object/male" while another person may refer to it as "third-person/possessive/male/singular/object". As a result, our MDDO will accept any ordering of the indices to access the intended knowledge. However, with this flexibility comes the requirement that all indices must be unique. In regards to allocating memory dynamically, the reason is that learning is an incremental process. Knowledge such as English is so complex that it is difficult to learn it all at once. Thus it is learned over a span of multiple teachings. As a result, the MDDO needs to allocate memory dynamically as needed to allow for dimensions, indices, and entries to be added to the struture. Taking the personal pronoun example again, after initializing the MDDO with the dimensions of person and case, a third dimension of gender may be added. Additionally, a third index of neutral can be added after the gender dimension is originally taught as having male and female indices only. The neutral index is needed for the personal pronoun 'it'. As for providing a solution to handle a sparse amount of knowledge, the reason is that as dimensions continue to be added, the increase in specifications reduces the possibility of a knowledge object being able to fill the space. To illustrate this, notice that when adding the gender dimension to personal pronouns, all entries classified as first-person and second-person do not have any gender in the English language. For example, the "second-person/subject/non-possesive/singular" personal pronoun 'you' does not have a gender of male, female, or neutral so no entry is required.

The three main functions that maintain the integrity and functionality of MDDO are add, search, and auto-populate. After creating an object instance of this data organization, the add function allows one to add dimensions, indices, and entries as needed. For each added dimension, at least one index must be provided. Since these dimensions are assumed orthogonal, the name of the new dimension must be new to MDDO. This specification of arguments also covers the case of adding indices to an existing dimension. In this case, if the dimension name is not known by the MDDO, then it will not add the indices. To add an entry to MDDO, its unique combination of indices for the different dimensions is needed as the argument. The same combination of indices will be used as the key to locate this entry. Given a combination of indices in any order, each index from a different dimension, the search function returns its corresponding stored entry if it exists. Note that since it is highly likely to be a sparsely populated data organization, only the indices for those pertinent dimensions are needed. Finally, given an existing MDDO with some initial number of dimensions and entries, when adding a new dimension, the auto-populate function will fill in entries for one of the indices of the new dimension with all of the existing entries. For example, consider the MDDO that manages the forms-of-be verb such as ‘am’, ‘are’ and ‘is’. Suppose it initially has the two dimensions person and case. The person dimension has indices of first-person, second-person, and third-person, while the case dimension has singular and plural, respectively. On a newly added dimension, tense, the auto-populate function can fill the entries for the index present tense with all the original entries. Entries for other tenses such as past tense and future tense can be added using the add function. It is important to note that the auto-populate function should only be used when the existing set of entries can fit under one of the newly added indices; otherwise erroneous entries may be created that may be inappropriately classified. For instance, it would not be accurate to auto-populate the set of personal pronouns {‘I’, ‘we’, ‘you’, ‘they’, ‘he’} into the gender index of male. The reason is that only one pronoun ‘he’ would be correct; the rest are gender neutral, and so they are not correctly organized under the ‘male’ classification. In addition to these basic functions, MDDO also provides the information about what it has stored: including the set of dimensions, the set of indices of any dimension, and the set of all entries given a specified combination of indices.

III. HIGH LEVEL SOLUTIONS OF MDDO

Our solution makes use of a basic data organization called dictionary. A dictionary is a collection of data where each piece of data is identified by a unique key. The three main functions of a dictionary are add, search, and delete. The add function simply adds a new data entry and its key into the dictionary, while delete and search functions remove or find the data entry corresponding to the given key, respectively. The dictionary data organization is a well-researched structure, which may be implemented by an AVL tree or a hash table [8, 9]. Our implementation of the MDDO uses the C++ library template class for dictionary called map.

We use three separate structures to manage the MDDO, each implemented as a map. The first structure is the dimensions list, which simply stores all the dimensions of the MDDO instance using the dimension name as the key. The second structure is the index map that uses an index as the unique key to identify which dimension the index belongs. The third structure, referred to as the MDDO map, uses a combination of indices as the unique key to access the stored knowledge object. Since the MDDO may accept any ordering of the indices, this unique key is created by rearranging the indices according to the alphabetical ordering of their respective dimensions. Given a combination of indices, the dimension of each index is first identified using the index map. Then these dimensions are sorted alphabetically by adding each (dimension, index) pair into a temporary map structure using dimension as the key. We can now traverse the map in sorted order and retrieve the corresponding index of each dimension. By combining these indices in this new order, a final unique key is formed. This unique key is then used to either add or search the entry in the MDDO map. Finally, to auto-populate the entries of one index of a new dimension, this index is appended to the key of each existing entry, thus creating a new index set. Using this new index set, each entry is simply added using the add function of the MDDO map, which by our design automatically creates a unique key for insertion.

IV. ANALYSIS OF MDDO

Let k be the number of dimensions in the MDDO, and h the largest number of indices among all the dimensions. The dimensions list has exactly k entries, and the index map size is O(hk). The MDDO map will have a maximum of h^k entries for keys that have an index for each of the k dimensions. Although our structure will also have entries whose keys have indices for a smaller number of dimensions, the total number of those entries is no more than h^k, so the size of MDDO is O(h^k). However, for all our applications using MDDO, there are a sparse number of entries as the number of dimensions increases. Therefore, we will measure our performance based on n, where n is the total number of entries and is expected to be much smaller than h^k. Although it is a sparse data organization, we still expect that the total number of entries is much larger than the number of dimensions and the maximum of indices among all the dimensions. In other words, n is much greater than h and k, respectively.

As for time performance, since we use the C++ library template class map for all three internal structures, if it is implemented by an AVL tree, the worst-case performance is O(log m) time for all three functions, where m is the total number of entries in the map. If implemented by a hash table, the average case performance for each operation is O(1) when a large number of operations are performed. We will analyze our performance based on the number of times those functions will be called and use the worst-case performance for AVL tree. The worst-case performance to add a new dimension is O(log *k*). To add an index to a dimension requires a search for the dimension in the dimensions list and an add to the index map. The search takes O(log *k*) time, and the add takes O(log *hk*), so the total performance is O(log *k* + log *h*). Given a key with *k* indices, the costs of adding and searching for an entry in the MDDO are the same, since they go through the same procedure of creating a unique key. First, it takes O(*k* (log *k* + log *h*)) time to find all the dimensions of the *k* indices from the index map. Then, since the sorting of the dimensions is accomplished by using *k* add functions to a map, the time required is also O(*k* log *k*). Finally, creating the unique key takes O(*k*) time. As a result, the overall cost of creating a sorted unique key is O(*k* (log *k* + log *h*)) time. After the unique key has been created, the task of adding or searching an entry in the MDDO map of *n* entries both take O(log *n*) time. Consequently, the worst-case time to add or search for an entry is O(*k* log *k* + *k* log *h* + log *n*) time, and it can be simplified as O(log *n*) since *n* is expected to be much greater than *h* and *k*. The cost to auto-populate one entry requires appending the index of the new dimension, which takes O(1) time, and the time to add that entry to the MDDO map is O(log *n*). Since there are a total of *n* entries to auto-populate, the worst-case performance is O(*n* log *n*) time.

V. CONCLUSION

In this paper, we have implemented the data organization MDDO to store knowledge of a category. Knowledge objects within the same category have the characteristics that each can be identified uniquely by a combination of indices from a number of independent dimensions. In addition, the functionalities of the MDDO allow knowledge to be added dynamically to accommodate the incremental learning process. This data structure has been used in handling the communication problem of a learning program. It is used to store part of the grammar, specifically, the rules of the English grammar and pronouns. It also assists in the processing and the production of a sentence. Finally, it may also be used to organize verb forms and tenses in later learning stages.

REFERENCES

[1] K. Cheng, "An Object-Oriented Approach to Machine Learning," Proc. WSES International Conference on Artificial Intelligence, June 2000, pp. 487-492.

[2] S. Russell and P. Norvig, Artificial Intelligence, A Modern Approach, 2nd Ed., Prentice Hall, 2003.

[3] K. Cheng, "The Representation and Inferences of Hierarchies," Proc. IASTED International Conference on Advances in Computer Science and Technology, January 2006, pp. 269-273.

[4] K. Cheng, "Representing Definitions and Its Associated Knowledge in a Learning Program," Proc. International Conference on Artificial Intelligence, June 2007, pp. 71-77.

[5] R.A. Frost, "Realization of Natural Language Interfaces Using Lazy Functional Programming," ACM Computing Surveys, 38,4, Article 11, Dec. 2006.

[6] W. Faris and K. Cheng, "An Object-Oriented Approach in Representing and Parsing the English Grammar," technical report #UH-CS-08-03, Computer Science Department, University of Houston, submitted for publication.

[7] W. A. Sabin, The Gregg Reference Manual, A manual of style, grammar, usage, and formatting, 10th Ed. McGraw-Hill/Irwin, New York, 2005.

[8] E. Horowitz, S. Sahni, and D. Mehta. Fundamentals of Data Structures in C++. Computer Science Press, New York, 1995.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 2nd Ed. The MIT Press, Cambridge, Mass. 2001.