Improving Array Search algorithm Using Associative Memory Neural Network

Emad Issa Abdul Kareem [†] and Aman Jantan [††],

Universiti Sains Malaysia (USM), School of Computer science, Pinang, Malaysia

Abstract

When we face any problem need to solve it with or without computer, we must choose a proper data structure, to represent its data efficiently. This means, we can insert, delete and search any element using chosen data structure efficiently. Array is the most widely used data structure.

Our research aims to combine array with BAM neural network to improve array search algorithm to find any element in the array structure with only one comparison. Thus, we are developing new array structure algorithms (inserts, delete and search)

Experimental test presents promising results by applying all the proposed algorithms. In addition Experimental test proves that the proposed search algorithm is able to find any element in the array structure with only one comparison via using BAM.

Key words:

Data structure algorithms, array data structure, search algorithms, neuralnetwork, associative memory, bidirectional associative memory BAM.

1. Introduction

In solving any problem with or without a computer it is necessary to choose an abstraction of reality, i.e., to define a set of data that is to represent the real situation. This choice must be guided by the problem to be solved.

Then, choose the prepare data structure representation of this information. This choice is guided by the tool that is to solve the problem, i.e., by the facilities offered by the computer.

If we can choose a prepare data structure to represent the data of our problem, this means an efficient operations will be done to deal with this data.

The main operations in any data structure are INSERT any element, DELETE any element and SEARCH for any element.

The task of searching is one of most frequent operations in computer programming. It also provides an ideal ground for application of the data structures so far encountered. There exist several basic variations of the theme of searching, and many different algorithms have been developed on this subject.

Manuscript received May 5, 2008
Manuscript revised May 20, 2008

This work will spend on combine array with BAM neural network to improve array search algorithm, which is initially developed by Kosko (1988, 1992).
This development is a conditional development; in which the condition is to maintain all the features of array structure.

2. Literature review

Typical programming languages such as Pascal, C or Java ext... Provide primitive data types such as integers, real, Boolean values character and string. They allow these to be organized into arrays, where the arrays generally have statically determined size.

2.1Array structure:

Array is probably the most widely used data structure; in some languages it is even the only one available. An array consists of components which are all of the same type, called its base type; it is therefore called a homogeneous structure. The array is a random-access structure, because all components can be selected at random and are equally quickly accessible. In order to denote an individual component, the name of the entire structure is augmented by the index selecting the component. This index is to be an integer between 0 and n-1, where n is the number of elements, the size, of the array [4].

2.1.1Insertion and Deletion of element in an array structure:

When we are storing elements in array and we need to store another element in the array, then we need to INSERT this element in prepare position.
Other cases if you accidentally stored duplicate data, then you need to DELETE the duplicate element.

Depending on [5] to insertion any element in array structure, we need to have two Information: the element to be inserted and the position to which it will be inserted (see figure 1).

For deletion, we only need the position to which it will be deleted (see figure 2).

Insert Algorithm
Input: **A**: array of length n. **X**: value of element to be inserting. **POS**: the position of X in the array A.
Output : **A**: array of length n (after insertion)
Step1 : For I = n-1 to POS A [I]:=A [I-1]. **Step2**:A[I]:=X.

Figure 1: Insert Algorithm

Delete Algorithm
Input: **A**: array of length n. **POS**: the position of X in the array A.
Output : **A**: array of length n (after insertion) **X**: the value of element in position **POS**.
Step1 : For I = POS to n-1 A [I-1] =A [I]. **Step2**: X=A [I-1]. **Step 3**: A [I-1]:=0.

Figure 2: Delete Algorithm

2.1.2 Search Algorithms:

- Basic Sequential search

This very basic algorithm is also known as the linear search or brute force search. It searches for a given element in an array or list by looking through the array sequentially until it finds the element or reaches the end of the structure. Let n denote the size of the array or list on which we search. Let An be a random variable representing the number of comparisons made between keys during a successful search and let An be a random variable for the number of comparisons in an unsuccessful search [7].

[6] Refers, there are another four approaches of sequential search:

1- Self-organizing sequential search: move-to-front method.
2- Self-organizing sequential search: transpose method.
3- Optimal sequential search.
4- Jump search.

-Sorted array search

This type of algorithms is designed to search for an element in an array whose elements are arranged in order.

According to [6] there are three approaches:

1- Binary search.
2- Interpolation search.
3- Interpolation-sequential search.

-Hashing

Hashing or scatter storage algorithms are distinguished by the use of a hash-hashing function. This is a function which takes a key as input and yields an integer in a prescribed range (for example, [0, m— 1]) as a result.

The function is designed so that the integer values it produces are uniformly distributed throughout the range. These integer values are then used as indices for an array of size m called the hashing table. Records are both inserted into and retrieved from the table by using the hashing function to calculate the required indices from the array keys [7].

In this paper we want to combine between array structure and associative memory neural network.

2.2 Associative Memory Neural Nets:

Associative memory neural nets are single-layer nets in which the weights are determined in such a way that the net can store a set of pattern associations. Each association is an input-output vector pair, s:t. If each vector t is the same as the vectors with which it is associated, then the net is called an autoassociative memory. If the t's are different from the s's, the net is called a heteroassociative memory. In each of these cases, the net not only learns the specific pattern pairs that were used for training, but also is able to recall the desired response pattern when given an input stimulus that is similar, but not identical, to the training input [1].

Bidirectional associative memory (BAM) is one of the heteroassociative memory [3].

2.2.1 Bidirectional Associative Memory BAM:

A BAM stores a set of pattern associations by summing bipolar correlation matrices (an n by m outer product matrix for each pattern to be stored). The architecture of the net consists of two layers of neurons (see figure (3)), connected by directional weighted connection paths. The net iterates, sending signals back and forth between the two layers until all neurons reach equilibrium (i.e., until each neuron's activation remains constant for several steps). Bidirectional associative memory neural nets can respond to input to either layer. Because the weights are bidirectional and the algorithm alternates between updating the activations for each layer, we shall refer to the layers as the X-layer and the Y-layer (rather than the input and output layers)[2].

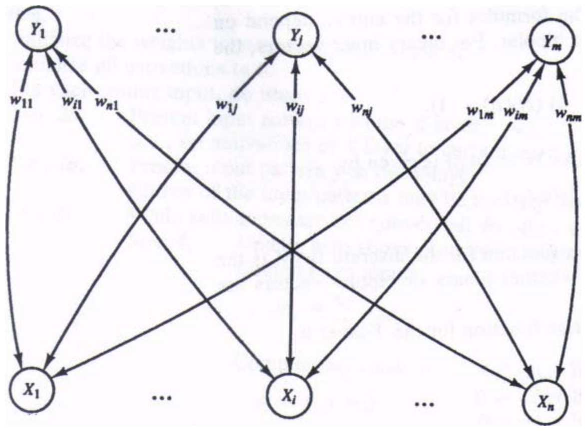

Figure 3: Bidirectional associative memory BAM

3. Array structure using BAM

The proposed array structure has the same structure of the original one plus BAM matrix.
Using BAM neural network we will create associative memory to store each element with its index (x : i. where : x is any element in the array and i is the index of the element x in the array) Thus , we can use this associative memory to retrieve x if the input of the BAM neural network is i and vice versa (we can retrieve i if the input is x) .

We propose three algorithms, the first algorithm (see figure (4)) is to insert any element in the proposed array structure, the second algorithm (see figure (5)) is to delete any element from the proposed array structure and the third algorithm (see figure (6)) is to search about any element in the proposed array structure.

Insert Algorithm
Input: **A**: array of length n. **X**: value of element to be inserting. **I**: index of **X** value. **BAM**: bidirectional associative memory matrix.
Output: **A**: array of length n (after insertion) **BAM**: bidirectional associative memory matrix.(new BAM after insertion)
Step1: convert **X** value and **I** value to binary vector **XV** and **IV**. **Step2**: convert **XV** and **IV** to bipolar using hard limiter function [4]. $Y = \begin{cases} 1 & Y > 0 \\ -1 & Y <= 0 \end{cases}$ **Step3**: insert **X** value in the array in position **I**. A[I]=X **Step4**: Create Wight matrix **W** between **XV** and **IV** W=XV*IV. **Step5**: add new weight matrix **W** to the **BAM** BAM=BAM+W. **Step6**: end.

Figure 4: Insert algorithm.

Delete Algorithm
Input: **A**: array of length n. **I**: index of the element to be deleting. **BAM**: bidirectional associative memory matrix.
Output: **A**: array of length n (after deletion). **X**: the value of the deleting element. **BAM**: bidirectional associative memory matrix.(new BAM after deletion)
Step1: delete the element from array in position **I**. X=A [I]. **Step2**: convert **X** value and **I** value to binary vector **XV** and **IV**. **Step3**: convert **XV** and **IV** to bipolar using hard limiter function [4]. $Y = \begin{cases} 1 & Y > 0 \\ -1 & Y <= 0 \end{cases}$ **Step4**: Create Wight matrix **W** between **XV** and **IV** W=XV*IV. **Step5**: delete new weight matrix **W** from the **BAM** BAM=BAM-W. **Step6**: end.

Figure 5: Delete algorithm.

Search Algorithm
Input: **A**: array of length n. **X**: the element to be search about it. **BAM**: bidirectional associative memory matrix.
Output: **found**: is a Boolean variable, **found**=true if **X** is found in the array else **found**=false if not found. **I**: the index of the value in the array.
Step1: convert **X** value to binary vector **XV**. **Step2**: convert **XV** to bipolar using hard limiter function [4].

```
$Y = \begin{cases} 1 & Y > 0 \\ -1 & Y <= 0 \end{cases}$

Step3: create the assumed index AI
       AI=XV*BAM.
Step4: convert assumed index AI from bipolar to binary vector
Step5: convert assumed index AI from binary to integer I.
Step6: use the assumed index I to access the element in the array and Compare it with X
If A [I] =X then
            begin
                Found=true
                return (I)
            end
            Else found= false.
Step7: end.
```

Figure 6: search algorithm.

4. Testing and discussion:

Empirically, we have simple exercise to illustrate the improving of the array search algorithm via using BAM neural network with array structure.
In this exercise, we have an array of character with length 5.

We want to store (c, g, f, w, and z) in the array (see figure (7)).

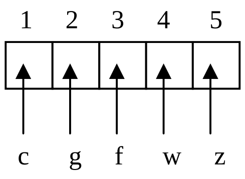

Figure (7): array of character (Array of char) with length 5.

Depending on the suggested insert algorithm to insert any element in the array (see figure (4)), we have to convert all the data and its position (index) in the array to binary and then to bipolar (see table (1)).

Table (1): table of convert the data and its poison to binary then to bipolar

Data	Binary	Bipolar	Index	Binary	Bipolar
c	00011	-1-1-111	1	001	-1-11
g	00111	-1-1111	2	010	-11-1
f	00110	-1-111-1	3	011	-111
w	11000	11-1-1-1	4	100	1-1-1
z	11010	11-11-1	5	101	1-11

All programming languages are dealing with the characters depending on its position in English alphabet. Thus, in table (1) we convert all the character depending on its position in the English alphabet, this means (a will be 1, b will be 2 … and z will be 26).

Depending on the suggested insert algorithm, we have to create the bidirectional associative memory BAM to all the elements which are want to store them in array as follow:

To store c → -1-11 to store g → -11-1

$$\begin{bmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} +$$

To store f → -111 to store w → 1-1-1

$$\begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} +$$

To store z → 1-11 **BAM to store all**

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & -3 & -1 \\ 5 & -3 & -1 \\ -3 & 5 & -1 \\ -3 & 1 & 3 \\ -3 & 1 & -1 \end{bmatrix}$$

We can use the BAM matrix to find any element in this array structure.
For example we want to find w in this array and return Boolean value (found) and its position if it is in the array and (not found) if it is not in the array.

Depending on the suggested search algorithm(see figure (6)), we just need to multiply the w after converting it to bipolar with BAM matrix, to return a position, and then we have to compare the stored value in this position with searching value w.

If they are equal, it will imply that w is already found in this position, whenever not equal this means w not found, as follows:

w after conver

BAM

$$\overset{\downarrow}{[1\,1\,-1\,-1\,-1]} * \overset{\downarrow}{\begin{bmatrix} 5 & -3 & -1 \\ 5 & -3 & -1 \\ -3 & 5 & -1 \\ -3 & 1 & 3 \\ -3 & 1 & -1 \end{bmatrix}} = [19,-13,-3]$$

After applied hard limiter function:
[19,-13,-3]→ [1,-1,-1]→ 4 is the position in the array, after comparing the stored value in this position w with searching value w ,we can find there are equal, this means w is already found in the array in position 4.

But for example, when we want to search about (b) in the same array:

B → 00010 → -1-1-11-1

b after conver

BAM

$$\overset{\downarrow}{[-1\,-1\,-1\,1\,-1]} * \overset{\downarrow}{\begin{bmatrix} 5 & -3 & -1 \\ 5 & -3 & -1 \\ -3 & 5 & -1 \\ -3 & 1 & 3 \\ -3 & 1 & -1 \end{bmatrix}} = [-7,1,9]$$

After applied hard limiter function:
[-7, 1, 9]→ [-1,1,1]→ 3 is the position in the array, after compare the stored value f in this position of array with searching value b, we can find there are not equal, this means b is not found in the array at all.

Depending on these tests:

1- No need to sort the data of the array structure before using the proposed search algorithm because its technique do not depend on sorted data (for example binary search algorithm need to sort the data of array structure before use it).

2- Ones can repeat the same steps using other types of array (integer, string ext…).

3-With this proposed array structure, we can use all the algorithms of sorting, traversing ext….

4- In case of recurrence of any element in more than one location in the array, and we want to know the number of element frequency with its positions, ones can repeats all steps of proposed search algorithm after implementing minus operation in each iteration between the stored matrix of the searched element and BAM matrix, storing new BAM matrix in new BAM matrix in order to avoid destruction the origin BAM.

5- Many programming languages have its own function to convert char, string, integer…ext to convert char, string integer ext… to binary, thus no need to built any array convert function.

6-Ones can use the proposed algorithms with any array which it is already built before; via creating BAM to all the elements and its positions.

5 Conclusions

Compared with all the search algorithms used, the proposed algorithm is more efficient because it is just need one comparison to find the element in sorted or unsorted array structure.

Even when the size of BAM increasing depending on the length of the array; search algorithm is still an efficient algorithm and it is still need one comparison to find the element.

References

[1] Emad Issa Abdul Kaream. "Hopfield Neural Network Using Genetic Algorithm", M.Sc. thesis, High studies institute for computer and information, Baghdad, Iraq, (2001).

[2] Emad Issa Abdul Kaream. "Alternative Hopfiled Neural Network With Multi-Connect Architecture." journal of College of Education, Computer Department, Al-mustansiryah University, Baghdad, Iraq, (2004).

[3] Laurence Fausett Ed., "Fundamental of Neural Networks, Architectures, Algorithms and Applications". Prentice-Hall, (1994).

[4] N Wirth, "Algorithms & data structures ", Book, Prentice-Hall, Inc. Upper Saddle River, NJ, USA 1985.

[5] Rajeev Raman ,Venkatesh Raman and Srinivasa Rao," Algo rithms and Data Structures",Book, Springer Berlin / eidelberg,2001.

[6] G.H. Gonnet and ETH, Zurich,"Handbook of Algorithms and Data Structures In Pascal and C ", ADDISON-ESLEYPUBLISHINGCOMPANY, Second Edition, 1991.

[7] D.S.Malik "data structures using C++", Couse Technology ,Inc ,first edition ,2003.

Emad Issa is a lecturer in department of computer science in Al-Mustansiriyah University, Baghdad, Iraq, received the B.Sc. in computer science from Baghdad University, Baghdad, Iraq in 1994, and in obtained his MSc. In AI from Commission for Computers and Informatics, Informatics Institute for postgraduate studies, Baghdad, Iraq, during 1998-2001. Now he is PhD. Student in Universiti Sains Malaysia (USM). His research interests are in the fields of AI ,

Image processing , Pattern recognition and development techniques.

Aman Jantan is a senior lecturer at the school of computer Scince / Universiti Sains Malaysia (USM) , received the BComp.Sc , M.Sc. degree in computer science from the Universiti 1993 and 1996 obtained his PhD in 2002 from the same university. His research interests are in the fields of AI, computer and network security, E-commerce/web intelligence, compilers design and development techniques.