

Duplicate Tree Structures in DPP™ Virtual Associative Memories

by Harry Direen, Ph.D.

Release 1.1

NOTE: In October 2003, Xpiori, LLC acquired Xpiori Holdings, LLC including all technology and patents. Any references to Neo, Xpiori or Xpiori Holdings, LLC technology or patents as such are now the property of Xpiori, LLC.

**Xpiori, LLC
2864 S. Circle Dr.
Ste. 1200
Colorado Springs, CO 80906
(719) 527-1315
www.xpiori.com**

© 2007 by Xpiori, LLC. All rights reserved.

Version 1.1

Copyright © Xpiori, LLC All Rights Reserved

Xpiori technology is protected by the following patents:

US Patent #5,742,611 (21 Apr 98)

US Patent #5,942,002 (8 Aug 99)

US Patent #6,157,617 (5 Dec 00)

US Patent #6,167,400 (26 Dec 00)

US Patent #6,324,636 (27 Nov 01)

US Patent #6,493,813 (10 Dec 02)

US Patent #6,792,428 (14 Sept 04)

Other U.S. and international patents pending.

The information in this white paper has been provided by Xpiori, LLC. To the best knowledge of Xpiori, it contains information concerning the current state of information processing technology. Xpiori, LLC disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to products described in this paper, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. No specific reliance should be made on the material provided herein without thorough investigation of the technology and its proposed application to specific circumstances. Product and technology information is subject to change without notice.

Introduction

Duplicate trees are another method for handling duplicate data elements that occur within Virtual Associative Memories. Refer to the paper, "Management of Duplicate data Elements in DPP Virtual Associative Memories" for a discussion of method one. Method Two, Folded Duplicate Instance Values, handles duplicates by appending an instance number to the key data. The new icon representing key data with instance number is unique, and the association for this item is stored based on the unique icon. This method essentially distributes duplicate data items throughout the entire Virtual Associative Memory or the "NeoStore". Due to memory paging issues, distributing the duplicate data throughout the NeoStore can cause memory access penalties when working with a given duplicate list on large NeoStore. Duplicate trees help localize duplicate lists of data in memory thereby minimizing memory paging penalties.

Duplicate trees are built up from small, fixed-size Duplicate Quanta Arrays (DQArray). The array elements are the size of an association, assuming that the size of an association is large enough to hold both a memory pointer plus enough bits to index a DQArray. The DQArrays are allocated from a pool of memory separate from the Virtual Associative Memory or the "NeoStore". Duplicate trees are built one DQArray at a time, as needed to hold all of the duplicate associations. The duplicate associations are stored in numerical order within the duplicate tree. This allows fast binary searches to be performed to find a particular association. All associations within a given duplicate tree are unique.

Because duplicate trees are built from memory pools separate from the NeoStore, additional memory pools can be allocated when required as duplicate trees grow and as new duplicate lists are added.

Duplicate Trees

When there are no duplicates against a given item, the item's association will be stored within a quanta in the NeoStore. Figure 1 shows this situation.

Figure 1

Confirmer	Chain
Flags	Associatio

NeoStore Quanta

When there are one or more duplicates against a given item, a duplicate tree is started. The first duplicate against a given item will cause a DQArray to be allocated from the DQArray pool. The association stored in the NeoStore quanta will be moved to the first location in the DQArray. The new association will become the second item in the DQArray. The NeoStore quanta will now be used to point to the DQArray. The "N" section of the quanta will contain the number of duplicates stored. The "F" section of the quanta will contain the Primary and Allocated flags required by the NeoStore structure along with the number of levels in the duplicate tree. In this case the number of levels will be 1. Figure 2 shows the new arrangement. This is a level 1 duplicate tree. Additional duplicates against this item may be added until the level one DQArray is filled.

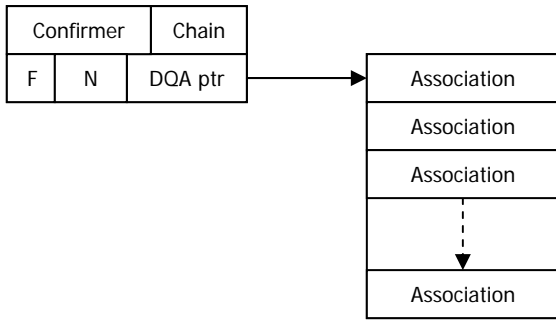


Figure 2
Level 1 Duplicate Tree

When the level 1 DQArray is filled, a level 2 duplicate tree is started. A DQArray is allocated from the DQArray pool. This array is used as pointers to the next level DQArrays (refer to figure 3). The first location in the DQ pointer array will point to the already full DQArray. Another DQArray will be allocated from the DQArray pool and will be used to hold additional associations. The DQArray pointer in the NeoStore quanta will be changed to point to the new DQ pointer array. The level flags in the "F" section of the quanta will be changed to indicate a level 2 tree. As more associations are added, more DQArrays can be added until the DQ pointer array is filled. When this pointer array is filled, another level can be added. Figure 4 shows a level 3 duplicate tree. Higher level trees can be created as required.

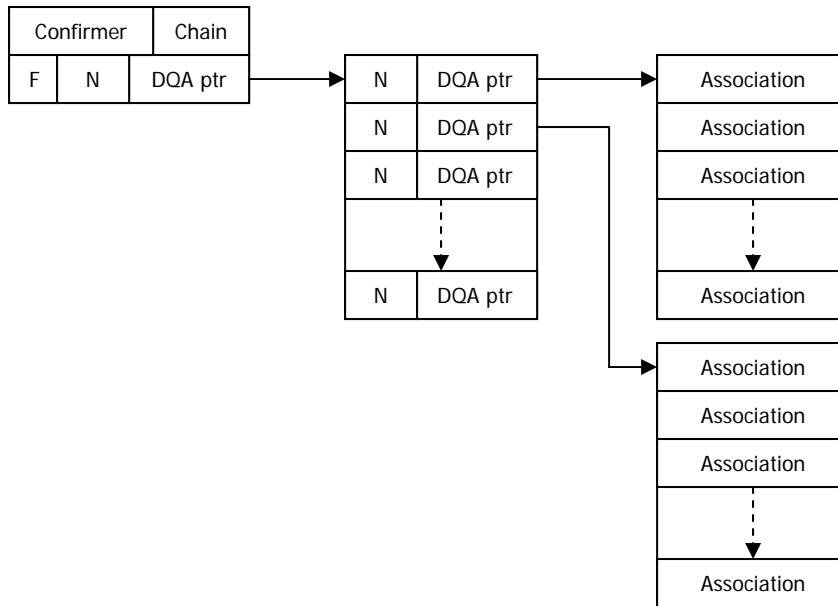


Figure 3
Level 2 duplicate Tree

The "N" in the DQ pointer arrays store the number of valid items in the DQArray pointed to. If the DQArray pointed to contains associations, then N will be the number of associations stored in the array. If the DQArray pointed to is another DQ pointer array, N will contain the number of valid pointers in that array. The number "N" is used for efficient indexing through the various arrays.

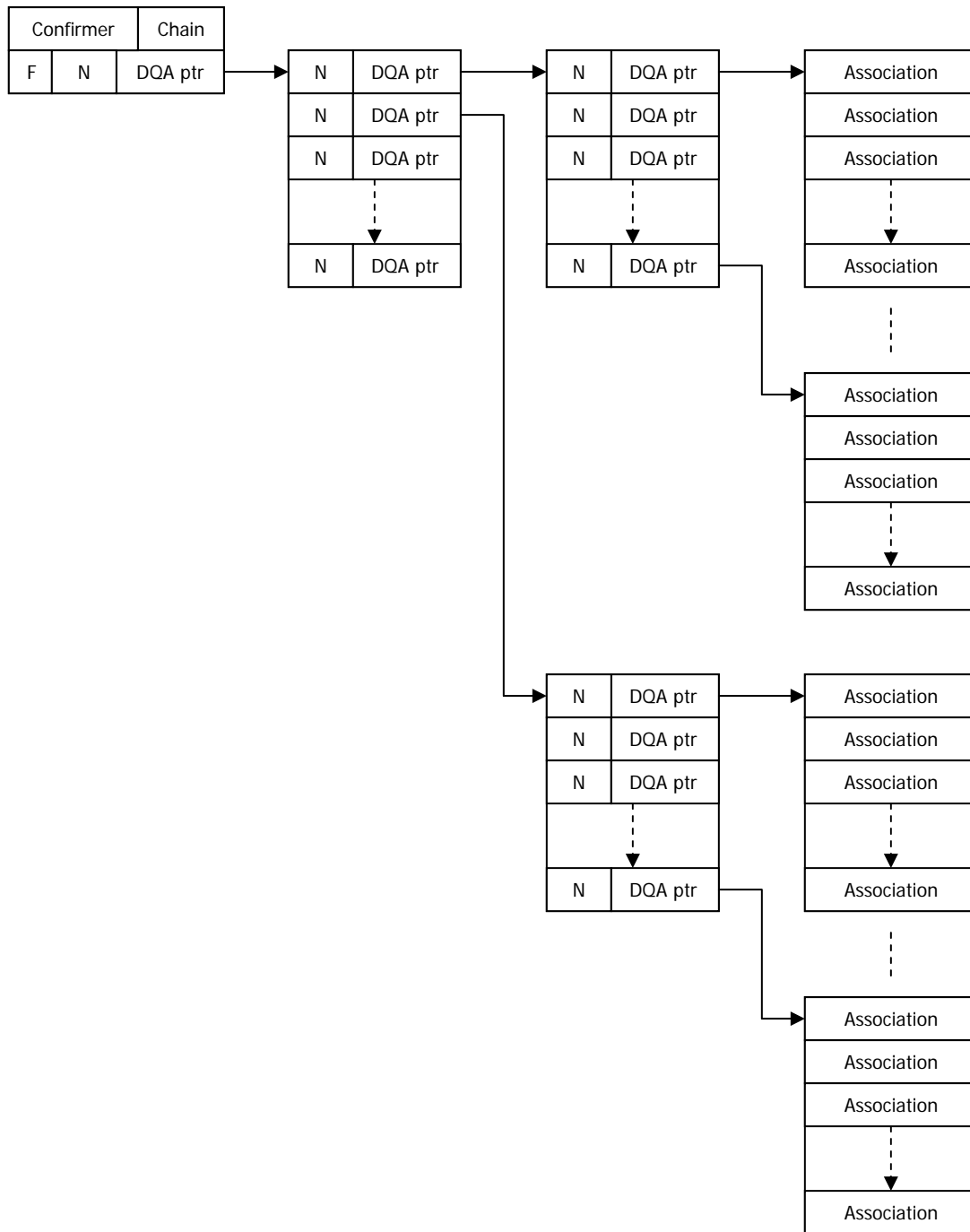


Figure 4
Level 3 Duplicate Tree

As noted above, associations are stored in numerical order in a duplicate tree. This implies that a hole must be opened up in the association list when inserting an association in the middle of a list. Rather than shifting long lists of associations, new empty DQArrays may be added to make room for the new associations. When a new association is added in the middle, the DQArray that Duplicate Tree Structures in DPP™ Virtual Associative Memory
 www.xpiori.com

the association is to be added into is checked for an empty slot. If there is an empty slot, the associations within the array are shifted to make room for the new association. If the current array is full, the array above the current array is checked for room. If there is room there, the associations are shifted in that direction to make room for the new association. If there is no room in the array above, the array below is checked. Once again, if the array below has room, the associations are shifted in that direction to make room for the new association. If the current array and the DQArrays above and below the current array are full, an empty DQArray can be inserted after the current array to add room for the new association. The associations are then shifted into the new array, making room for the new association. When an association is removed, it is removed from the appropriate DQArray. The rest of the associations within the array are shifted such that all empty slots are left at the end of the array. This approach to storing duplicates will leave holes in the duplicate tree structure, but these holes can be handled efficiently because the number of valid associations in the DQArray is stored in the DQ pointer arrays. This method prevents the entire list of duplicates from having to be shifted every time a duplicate is inserted or removed from the middle of the list.

Efficiency of Memory Use

The memory use efficiency for duplicate trees can be defined as:

$$MemEfficiency = \frac{Number\ of\ Associations\ Stored}{Total\ Number\ of\ Array\ Elements\ Used} * 100$$

The memory efficiency will be effected by the size of the arrays used (number of array elements per array) and the number of associations stored in the duplicate tree. Figure 5 shows the memory efficiency for an array size of 16 elements and an array size of 64 elements verses the number of associations in the tree. The solid line is for a 16 element array and the dashed line is for a 64 element array. The graph assumes that there are no holes (storage gaps between associations). The saw tooth nature of the graph is due to new arrays being added and new levels being added as the number of associations stored increases. The 16 element array efficiency levels out at about 93 to 94 percent when the number of stored associations become large. Figure 6 is the same graph with more associations being stored. It is clear from the graphs that 16 element DQArrays are a good trade-off between relatively few duplicates stored and large numbers of duplicates stored.

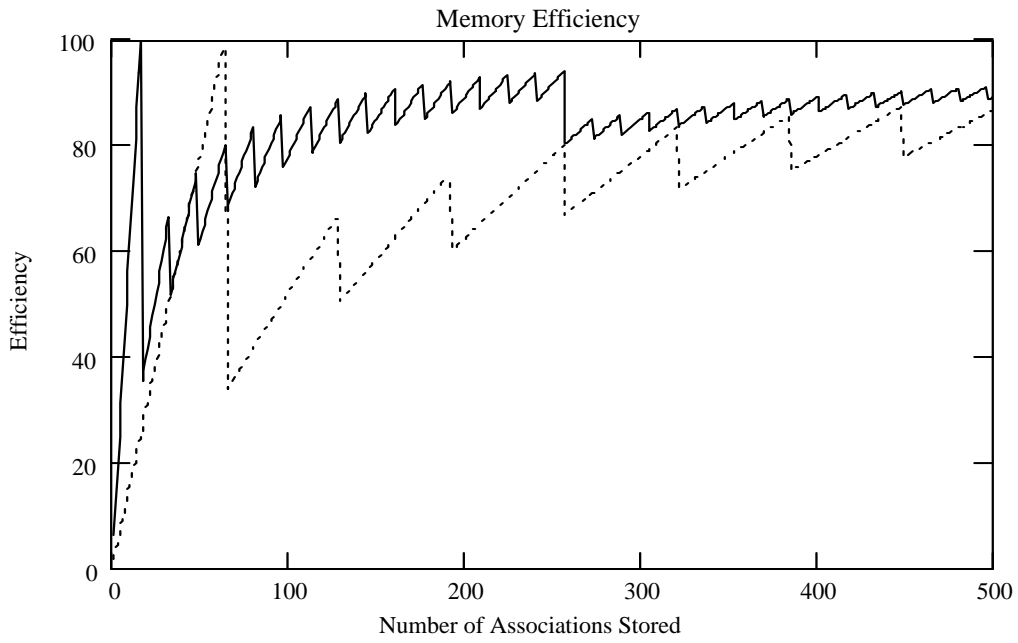


Figure 5

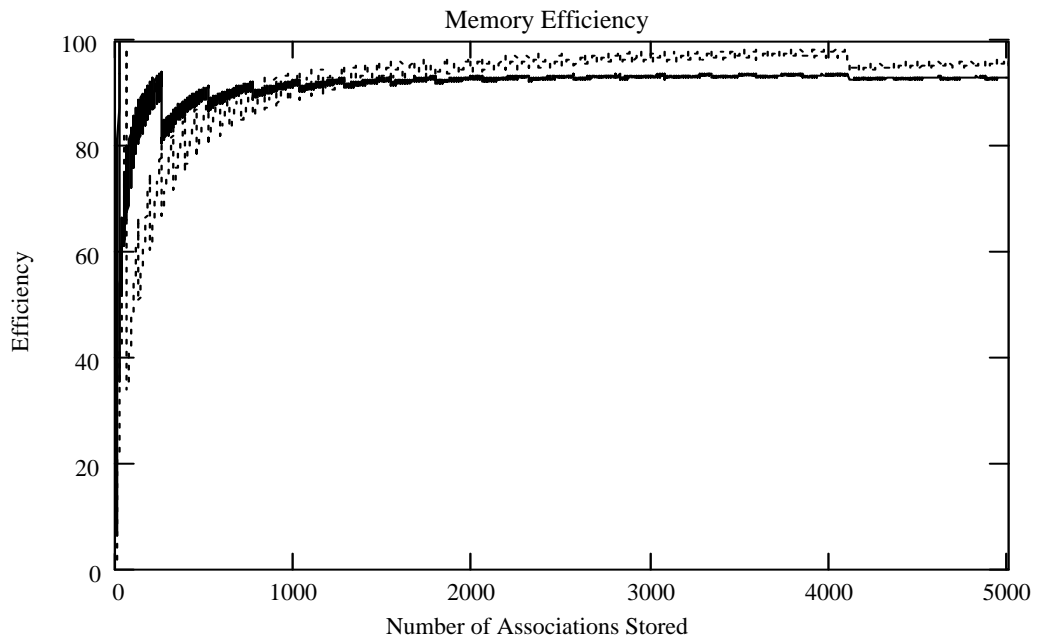


Figure 6

Conclusion

Duplicate trees provide a memory efficient method of handling long chains of duplicate entries in Virtual Associative memories. The tree structure allows fast binary searches of the stored associations.

Appendix

Related Papers

Brandin, Chris, "A Definition of Digital pattern Processing™"

Brandin, Chris, "DPP™ Memory Management"

Brandin, Chris, "Management of Duplicate Data Elements in DPP™ Virtual Associative Memories"

Brandin, Chris, "Three-Dimensional Content Scanning Using DPP™"

Brandin, Chris, "Optimized Coding Methods for Icon Generation and Manipulation In DPP™"

Brandin, Chris, "Behavioral Set and Field Descriptor Implementations in DPP™"

Direen, Harry and Phillips, Keith, "Finite Fields and Properties of the Xpiori Icon Generator, Associative Processing Unit, and Associative Memory Controller used in Digital Pattern Processing™"