

Management of Duplicate Data Elements in DPP™ Virtual Associative Memories

by Chris Brandin

Release 1.1

NOTE: In October 2003, Xpiori, LLC acquired NeoCore Holdings, LLC including all technology and patents. Any references to Neo, NeoCore or NeoCore 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

DPP supports three methods for managing duplicate entries in Virtual Associative Memories, each tailored for different performance requirements. This is not to say that one method generally performs better than any other; rather, depending on the application, one method may be more appropriate than the other. All methods can be simultaneously accommodated, so the decision as to which one to use can be made at run-time rather than in advance. There are different circumstances under which one may want to access duplicates.

1. The user wants every instance of a particular data element.
2. The user wants a particular instance of a data element.
3. The user wants the intersection of multiple data element instances. For example, "locate everybody named Smith, who lives on Main Street, who lives in Denver". Each element is likely to have duplicate entries, so the member sets must be intersected.

Each case will benefit from different methods of managing duplicates. Also, all cases can appear in a single application. Thus, it is desirable to simultaneously support multiple duplicate management schemes and be able to select the one most appropriate for a particular query.

Method One – Chained Duplicates

The Associative Memory Controller (AMC) typically stores duplicate entries (that is, the ones associating the same key) in a linked list. Duplicates can be accessed either by successive "read next" commands, or by specifying which duplicate instance is desired. This method is very fast when the user wants to start with the first instance of a duplicate, and sequentially access the rest. Although it is possible to specify a duplicate number when reading, the AMC has to navigate through the duplicate list to locate a particular instance. In applications containing long duplicate lists, performance may suffer.

Method Two – Folded Duplicate Instance Values

Because Icons are fixed-field representations of variable key data, it is possible to append additional data to key data without changing the Icon size. This feature makes it possible to "fold" the duplicate number into the Icon. Basically, this amounts to managing duplicates by making them no longer duplicates. For example, suppose there are 10,000 instances of the name "Smith" in a database of Colorado telephone listings, and we want the 8,768th one. Using Method One, 8,767 entries would have to be navigated before we arrived at the one we want. Method Two offers a better way to manage this:

1. One entry corresponding to "Smith" is created. The association is the count of "Smith" instances.
2. The 10,000 instances of "Smith" are added, but instance numbers are appended first. The first instance is called "Smith_D00000001", the second "Smith_D00000002", and so on. As each instance is added, the count contained in the "Smith" instance is incremented.
3. The 8,768th instance of Smith can be associated directly by specifying "Smith_D00008768".

There are two reasons a “Smith” entry containing the instance count is created: as duplicates are added, the number of instances preceding it may not be known; and, when data set intersections are being performed, a binary search of duplicate instances is required in order to be able converge.

Suppose a DPP Store is created to index a telephone directory. Each listing record consists of: first name, last name, street address, city, state, ZIP code, area code, and telephone number. All data elements are to be indexed, resulting in one DPP virtual associative memory entry for each data element in each record – a “record” being a listing. The association for each data element corresponds to the “record” number to which it belongs. In order to locate all people named “Smith” in Denver, CO, it is necessary to locate all entries that correspond to “first name = Smith”, and “city = Denver”, and “state = CO”. Basically, the desired answer is the set of records that corresponds to the intersection of the members of three sets: “last name = Smith”, “city = Denver”, and “state = CO”. The associations are used to identify the common record numbers. The first intersecting member of these sets is found in the following way:

1. Lookup the instance count for each item. “Smith” has 10,000 entries, “Denver” has 1,000,000, and “CO” has 2,500,000. Start with the item with the lowest duplicate count – “Smith” in this case.
2. Lookup “Smith_D00000001” (the first instance).
3. Divide the instance count for “Denver” by two – yielding 500,000. Lookup “Denver_D00500000”. If the association (record number) is lower than the association from “Smith_D00000001”, then divide by two again and lookup “Denver_D00250000”, if it is higher add the divided value and lookup “Denver_D00750000”. If the two never converge, lookup “Smith_D00000002” and try again. By successively applying this binary reduction method, the first record for “Smith” in “Denver” will be found.
4. Using a similar binary reduction technique, determine whether the data element “state = CO” converges with the record located in step 3.
5. Repeat steps 3-4, except using the running instance counts until all three sets are converged on a record; resulting in the first person named Smith, who lives in Denver, CO.

This method is surprisingly fast – because binary searches require relatively few probes. Another method can be employed that can yield much faster results in many cases.

Method Three – Folded Association Values

Suppose that instead of locating every “Smith” who lives in Denver, CO, we want to locate every “Katzenlieber” who lives there instead. The instance count for “last name = Katzenlieber” would be very low – probably one or two. To support method three, another index entry is added for each data element – similar to the method two entries. In this case, however, the associated record number is appended to the key instead of the duplicate instance number. So, if listing record number 4,963 contained “state = CO”, an entry for “CO_A00004963 would be created (these type of entries need no association, so a separate store may be used to save memory). The resulting lookup process becomes:

1. Lookup the instance count for each item. “Katzenlieber” has 2 entries, “Denver” has 1,000,000, and “CO” has 2,500,000. Start with the item with the lowest duplicate count – “Katzenlieber” in this case.

2. Lookup "Katzenlieber_D00000001" (the first instance). It returns an association (record number) of 4963.
3. Use the association value returned in step 2 and append it to "Denver" creating the key "Denver_A00004963", and attempt a lookup. In this example, the entry will be found.
4. Using the method described in step 3, lookup "CO_A00004963". The entry will be found, locating the first "Katzenlieber" who lives in Denver, CO..

If at least one item has a low duplicate count, this method can be very fast – because the number of probes is bounded by the item with the lowest duplicate count.

Conclusion

The Symbolic Processing and Virtual Associative Memory features of DPP can be used for efficient duplicate item management and very fast data set intersections.

Appendix

Related Papers

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

Brandin, Chris, "DPP™ Memory Management"

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 NeoCore Icon Generator, Associative Processing Unit, and Associative Memory Controller used in Digital Pattern Processing™"

#