

# On Coding Navigation Paths for In-Memory Navigation in Persistent Object Stores

Markus Kirchberg<sup>1</sup>, Alexander Kuckelberg<sup>2</sup>, Klaus-Dieter Schewe<sup>1</sup>, Alexei Tretiakov<sup>1</sup>

<sup>1</sup> Massey University, Department of Information Systems  
Private Bag 11 222, Palmerston North, New Zealand

<sup>2</sup>RWTH Aachen, Chair of Railway Studies and Transport Economics  
Mies-von-der-Rohe-Str. 1, 52056 Aachen, Germany

## Abstract

*We consider matrix index and navigation index approaches to in-memory navigation of persistent object stores. We demonstrate that both approaches can be re-formulated independently from the used coding technique. We expose the limitations of fixed length coding and the inefficiency of simple continued fractions as a variable length coding technique. The conclusion is that alternative variable length coding techniques, possibly based on known compression algorithms, should be explored. The architecture of persistent object stores should permit plugging in and configuring coding algorithms to optimise the operation of a particular system.*

## 1. Introduction

Persistent Object Stores (POSs) provide the core of the physical architecture of a database management system [6]. The term arose in the context of object oriented databases (OODBs), for which it became necessary to consider complex objects with identifiers, values of complex types and references [3, 16]. While the interest in OODBs has faded, persistent object stores are still extremely important, e.g. for realising XML-based databases, for which again complex tree-like values and references have to be supported [1].

The type of access to data in a POS can be classified into three groups:

**direct:** In this case a physical object is retrieved by providing its unique object identifier. This access method is always implicitly present, even in relational databases, where tuple identifiers are used on the physical level.

**associative:** In this case objects are retrieved by providing some key values. This is the most common access method for relational databases, which has led to various methods for hashing and index-based approaches such as B-trees, B<sup>+</sup>-trees, grid-files, etc. For spatial databases these access methods have been generalised to R-trees and R\*-trees [2, 9].

**navigational:** Navigational access is the only access method that does not have an analogue for relational databases. Roughly speaking navigational access retrieves objects by following the references between objects forward and backward. The major focus in POS research has been on navigational access [5, 10, 12, 13, 14, 19, 20, 21].

The approach by Subieta in [19] to navigational access is based on physical data structures on spiders of rings and thus is rather similar to old techniques used in network database systems.

While this is sufficient for databases of limited size, it is not the best access methods, if reference paths tend to become long [11]. The reason is that each navigation step requires access to physical data.

The approach by Kemper and Moerkotte in [10] provides access relations to support more efficient access to objects that can only be reached over a path, whereas Valduriez in [20] uses a separate index structure. While these approaches improve the access performance, they are static in the sense that the paths that are to be supported by the index or the access relation have to be selected in advance causing a serious drawback due to maintaining the additional access structures [11].

The High-Performance Object Store (HIPOS) approach by Rabitti and Zezula in [21] provides a direct encoding of paths using simple continued fractions (SICFs) [15]. However, the graph describing the references between physical objects has to be partitioned into trees and the SICF-encoding only applies to navigation within such a tree. Then an additional look-up table has to be used for navigation along more than one of these trees. Furthermore, the method produces an unnecessary overhead in the case of linear navigation [11]. This deficiency has been removed in the extension of HIPOS to the Matrix-Index Coding approach (MIC) by Kuckelberg in [12], simply by using SICFs for two different purposes at the same time: encoding the successors of a node in a tree as in HIPOS, and encoding linear paths.

Surprisingly, the more recent work on storage of XML documents did not follow any of these lines of research. Instead of this, most research, e.g. the work in favours a translation to a relational database. This may reflect the observation that despite their merits the navigational access methods were not yet completely convincing.

Therefore, we pick up this line of research in this article. In particular, we further analyse the HIPOS and MIC approaches and show first that it is possible to decouple navigational access from the particular coding technique, i.e. SICFs in the case of MIC. We then investigate in more detail the memory space needed for the path encoding. Based on this we show that fixed length encoding is a bad idea, whereas SICFs can be considered as a form of a variable length code. We further analyse this code and show that also SICFs are not the best encoding technique.

In Section 2 we revisit the MIC approach and describe the encoding using SICFs. We show that the actual encoding technique using SICFs is not an intrinsic part of the method, thus we can decouple navigational access from the SICF encoding. In Section 3 we look into more detail into performance parameters and estimate the required memory. Finally, in Section 3.3 we show our results on the inefficiency of the SICF encoding. We summarize our finding and conclusion in Section 4.

## 2. Navigational Access Based on SICFs

We completely abstract from the logical database level of and the transformation of a logical schema into physical storage structures. Thus, we may assume a physical object reference graph to be given. This is in full analogy to the approaches taken in [10, 12, 19, 21].

Formally, we assume a set  $\mathcal{O} = \{O_1, \dots, O_n\}$  of storage objects and a set  $\mathcal{R} = \{R_1, \dots, R_m\}$  of references between them. We write  $R_i = (O_s, O_e)$  for a reference from  $O_s$  to  $O_e$ . Then  $R_i^{-1} = (O_e, O_s)$  is called the *inverse reference* of  $R_i = (O_s, O_e)$ . We do not distinguish be-

tween an object  $O_i$  and its object identifier (OID). Thus,  $\mathcal{G} = (\mathcal{D}, \mathcal{R})$  defines a directed graph, called *object reference graph*. As usual, we may use an *adjacency matrix* to represent an object reference graph.

Without loss of generality we assume  $\mathcal{G}$  to be acyclic. As shown in [21] we may simply cover the whole graph by maximal acyclic subgraphs. Navigation between subgraphs is then supported by a specific table called the *linker*, whereas navigation index and matrix index coding only deal with the acyclic components.

## 2.1. Basic Terminology

Our emphasis here is on *navigational access*. Roughly speaking this means to walk along paths in the reference graph from a starting object to a set of goal objects to be retrieved. We may distinguish between *one step* navigation in the case a goal object can be reached by a single reference or inverse reference or *multi step* navigation if more than one reference (or inverse reference) is needed. We also distinguish between *forward* or *backward* navigation according to the exclusive use of references or inverse references.

In general, navigational access is performed using a sequence of one/multi step, forward/backward accesses.

We need some more terminology [21, 12]:

- The *enclosure* of an object  $O_s$  is the smallest set of objects  $E(O_s) \subseteq \mathcal{D}$  with  $O_s \in E(O_s)$  and containing all objects reachable from  $O_s$  by one step forward access.
- The *inverse enclosure* of an object  $O_s$  is the smallest set of objects  $E^{-1}(O_s) \subseteq \mathcal{D}$  with  $O_s \in E^{-1}(O_s)$  and containing all objects reachable from  $O_s$  by one step backward access.
- An object  $O$  is called *branching object* iff there is more than one reference starting from  $O$ .
- An object  $O$  is called *assembling object* iff there is more than one inverse reference starting from  $O$ .

The core of the navigation problem is now to organize the storage of references in such a way that fast navigation is enabled. Of course, the storage space and the time needed to reorganize the object store in the case of updates also has to be taken into account.

## 2.2. Simple Continued Fractions

Both approaches in [21] and [12] are based on an encoding of paths. For this purpose we identify OIDs with natural numbers, i.e. we assume a one-to-one mapping  $of : OID \rightarrow \mathcal{N} \subset \mathbb{N}$ . Let  $fo : \mathcal{N} \rightarrow OID$  denote the inverse of  $of$ . In Figure 1 we used  $of(O_i) = i$ .

HIPOS and MIC are both based on *simple continued fractions* (SICF), i.e. rational numbers

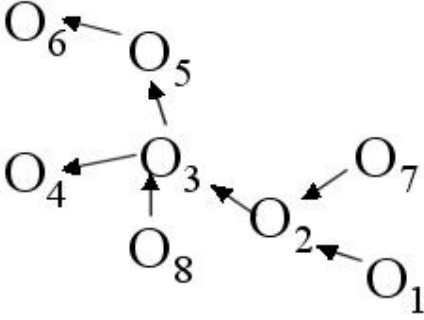


Figure 1: OIDs and references.

of the form

$$\alpha = \frac{N}{D} = \frac{1}{q_1 + \frac{1}{q_2 + \cdots + \frac{1}{q_{m-1} + \frac{1}{q_m}}}}$$

with  $m \in \mathbb{N}$ ,  $q_i \in \mathbb{N}$  for  $i = 1, \dots, m$  and  $q_m \geq 2$ . Thus,  $\alpha$  is uniquely determined by the sequence  $[q_1, q_2, \dots, q_m]$ . If we only consider the subsequence  $[q_i, \dots, q_m]$ , this defines the *partial continued fraction of order  $i$* . We write  $\frac{N_i}{D_i}$  ( $i = 1, \dots, m$ ) for the partial continued fractions of the SICF  $\alpha$ . The following equations are easily verified [15]:

$$\frac{N_1}{D_1} = \frac{N}{D} \quad \frac{N_m}{D_m} = \frac{1}{q_m} \quad \frac{N_i}{D_i} = \frac{1}{q_i + \frac{N_{i+1}}{D_{i+1}}}$$

$$D_{i+1} = N_i \quad q_i = D_i \div N_i \quad N_{i+1} = D_i \bmod N_i$$

Thus, each finite sequence of natural numbers defines a SICF and vice versa. This is fundamental for the HIPOS and the MIC approach.

### 2.3. The MIC Approach

Since a row in the adjacency matrix of the object reference graph represents the enclosure of the corresponding object (and a column represents an inverse enclosure), an idea is to encode rows and columns. If an object  $O$  references  $O_{i_1}, \dots, O_{i_k}$  ( $k \in \mathbb{N}$ ), i.e. the  $of(O)$  row contains “1” exactly for the columns  $of(O_{i_1}), \dots, of(O_{i_k})$ , and  $\sigma$  is any permutation of  $\{1, \dots, k\}$ , we may use the SICF of the sequence  $[of(O_{i_{\sigma(1)}}), \dots, of(O_{i_{\sigma(k)}})]$  for forward navigation. Analogously, the SICF of  $[of(O_{j_{\tau(1)}}), \dots, of(O_{j_{\tau(\ell)}})]$  for objects  $O_{j_1}, \dots, O_{j_\ell}$  ( $\ell \in \mathbb{N}$ ) referencing  $O$  and any permutation  $\tau$  of  $\{1, \dots, \ell\}$  supports backward navigation.

Note that backward navigation in the HIPOS approach through a subtree of the reference graph corresponds to a sequence of objects  $O_{i_0}, \dots, O_{i_m}$  ( $m \geq 2$ ),  $O_{i_0} \notin \{O_{i_1}, \dots, O_{i_m}\}$  referenced by exactly one object. More precisely, we have  $(O_{i_{j+1}}, O_{i_j}) \in \mathfrak{R}$ . In order to preserve the advantages of the HIPOS coding the MIC approach supports *transitive navigation*, i.e. the (backward) code for  $O_{i_0}$  is the SICF of the sequence  $[of(O_{i_1}), \dots, of(O_{i_m})]$ . The situation for forward navigation is analogous using the inverse reference graph.

With each object  $O$  we now associate a code  $Nav_{MIC}(O) = (a, b, c, d)$  accessible via a hash-function *hash*. The pair  $(a, b)$  corresponds to forward navigation;  $(c, d)$  to backward navigation. In order to distinguish transitive navigation from ordinary row/column encoding only the SICFs  $\frac{N}{D}$  for ordinary row (column) encoding are stored as  $a = N, b = D$  for forward navigation (or  $c = N, d = D$  for backward navigation), whereas for transitive navigation we interchange  $N$  and  $D$ . This is possible, as SICFs are always less than 1.

Thus,  $a > b$  indicates that  $\frac{b}{a}$  is the SICF for transitive forward navigation, whereas  $a \leq b$  indicates that  $\frac{a}{b}$  is the SICF for the sequence of all – at least two – referenced objects. For  $a = b = 0$   $O$  does not reference anything.

Analogously,  $c > d$  indicates that  $\frac{d}{c}$  is the SICF for transitive backward navigation, whereas  $c \leq d$  indicates that  $\frac{c}{d}$  is the SICF for the sequence of all – at least two – referencing objects. For  $c = d = 0$   $O$  is not referenced at all.

## 2.4. Decoupling Navigation and Coding Technique

Let us now reformulate matrix index and navigation index approaches, i.e. MIC and HIPOS, independently from the used coding technique.

In the matrix index approach introduced in [12, 11], if the OID navigated from has only a single outgoing reference, the code is expanded to a list of all OIDs that can be reached via single outgoing references only. Thus, for  $O_1$  in figure 1 the corresponding code would expand into a list containing  $O_2$  and  $O_3$  (in that order), while for  $O_2$  the list would contain a single member,  $O_3$ . On the other hand, if the OID navigated from has more than one outgoing reference, the code expands into OIDs for all of its nearest neighbours. Thus, for  $O_3$  in figure 1 the corresponding code would expand into  $O_4$  and  $O_5$ . A single code retrieval may resolve more than one reference traversals, resulting in faster navigation than with join index [12, 11].

In navigation index approach, references are labelled incrementally by independently using the same scheme for each OID with incoming references. In Figure 2, the labels are given as Latin letters. In addition each OID with more than one outgoing reference ("branching" object) and each OID with no outgoing references ("end" object) is given a globally unique "seed" ( $S_3, S_4$  and  $S_6$  in Figure 2). If the OID navigated from has only a single outgoing reference, its code is expanded into a list of labels for references leading from it to a branching or end OID, with the branching or end OID seed appended at the end of the list. Thus, code for  $O_1$  in Figure 2 will expand into the following list:  $[b, a, S_3]$ .

Note, that this automatically gives us access to codes for all intermediate OIDs (e.g., code for  $O_2$  is the one corresponding to  $[a, S_3]$ , while  $O_3$  is coded via a list containing the seed only,  $[S_3]$ ). The inclusion of seeds assures that the codes are globally unique. Now, codes and seeds can be matched to the corresponding OIDs via a separately maintained relational table, the so-called

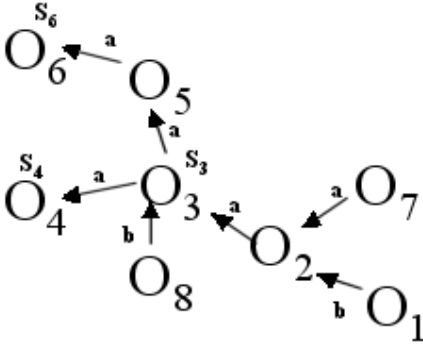


Figure 2: OIDs and references labelled for navigation index approach.

”expander”. Thus, unlike in the join index approach, it is possible to traverse several references and obtain an OID at the end of the navigation path by accessing a relational table only once.

In navigation index approach, navigation from branching OIDs is realized via a reduced join index table, containing only references originating from branching OIDs (the so-called ”linker”).

### 3. Analysis of Code Efficiency

The work in [12] provides a detailed comparison of time and space complexity for matrix index and navigation index approaches, and compares them against pure join index approach in an assumption that all codes occupy the same memory space. Yet, as codes correspond to potentially long sequences, the ability to apply fixed length coding cannot be taken for granted.

#### 3.1. Performance Parameters

Navigation performance depends on various parameters depending on the structure of the reference graph and the specific navigation task. In order to describe the reference graph, especially its degrees of branching and assembling, the following parameters are useful:

- $N$ , the number of objects in the reference graph;
- $NR$ , the number of references in the graph;
- $AR \in [0, 1]$ , the assembling rate, i.e. the probability of an object to be an assembling object;
- $BR \in [0, 1]$ , the branching rate, i.e. the probability of an object to be a branching object;
- $AAR \geq 2$ , the average assembling rate, i.e. the average number of objects referencing an assembling object;
- $ABR \geq 2$ , the average branching reference, i.e. the average number of objects referenced by a branching object.

Moreover, the following parameters are needed to describe the specific navigation:

- $NL$ , the navigation length, i.e. the number of references and inverse references involved in the navigation;

- $FN$ , the forward navigation length, i.e. the number of references involved in the navigation;
- $BN$ , the backward navigation length, i.e. the number of inverse references involved in the navigation;
- $NAO$ , the number of accessed objects, i.e. the number of target objects to be retrieved as the result of the navigation;
- $NOC$ , the number of forward/backward changes within the navigation.

In addition, we use two more parameters, the *object access cost*  $OAC$  which gives the costs to access a single object and the *index access cost*  $IAC$  given by the costs to access a commonly smaller index entry in additional navigation structures. These parameters depend on the used hardware.

Note that according to [10] some of the parameters e.g.  $NR$  can be determined from other graph parameters.

### 3.2. Memory Estimates

Consider now the case of an OID connected to further  $m$  OIDs as in Figure 3, with  $O_m$  being an end or branching OID. Then, the number of possible combinations is easily calculated as  $(N - 1)(N - 2) \cdots (N - m)$ , which for  $N \gg m$  can be approximated as  $N^m$ . The memory required to hold this number of alternatives is approximated as  $m \log_2 N$ . Therefore, even if we limit the length of navigation possible, with matrix index approach, no matter how the sequence is encoded, to accommodate all possibilities within a fixed length code, one would have to reserve for each OID an amount of memory that grows logarithmically with the number of objects in the database.

Now, let us consider the case of navigation index approach. If the only restriction is the number of OIDs connected in a path, than configurations with  $(N - m)/m$  references entering each of the OIDs in the path will be included. Let us fix the partition and only consider the different labelling for references entering each of the OIDs. Then, the number of possibilities is estimated as  $[(N - m)/m]^m$ . This, again, leads to memory required for fixed length coding growing logarithmically with the number of objects in the database, as  $m \log_2 N$  for  $m \ll N$ .

Yet, if in case of navigation index approach along with limiting the number of OID in the path we also limit the number of references entering a vertex by an  $r \ll N$ , the number of possible combinations may be manageable enough to allow fixed length coding. It can be estimated as  $r^m$ , which does not depend on  $N$ . To restrict the maximal number of references entering an OID (the "fan-out") one could introduce intermediate vertices, that do not correspond to OIDs. It appears that it is not possible to impose a similar restriction in case of the matrix index approach.

As we could see in this section, if navigation length is restricted, fixed length coding can be achieved by reserving for each OID the amount of memory sufficient to hold an array of all OIDs in the path ( $m \log_2 N$ ). This is valid for both matrix index and for navigation index approach, and does not depend on the coding function used.

If for navigation index approach in addition to limiting the navigation length the number of references entering an OID is also restricted, a more favourable code length can be achieved.

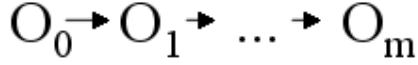


Figure 3: Transitive navigation.

Path length	Max. $q$	$q$ distribution	Optimal fixed length (bits)	SICF (bits)
12	65536	Uniform from 1 to max. $q$	192	336
12	256	Uniform from 1 to max. $q$	96	152
6	65536	Uniform from 1 to max. $q$	96	161
12	65536	95% uniform between 1 and 3, the rest 5% - max. $q$	192	46
12	256	95% uniform between 1 and 3, the rest 5% - max. $q$	96	36
6	65536	95% uniform between 1 and 3, the rest 5% - max. $q$	96	21

Table 1: Memory required for optimal fixed length coding and for SICF. For SICF, the value is an average taken over 400 random realizations.

### 3.3. Using SICF as a variable length code

In [12, 21] the only coding technique considered are SICFs.

As we saw in the preceding section, there is a limit to what can be achieved by using fixed length codes. On the other hand, SICF may still be useful as a variable length coding technique. To verify that, we encoded some sequences of random numbers and verified the memory size required to store the resulting codes. As a comparison, we also noted the memory required for the optimal fixed length coding storage (in which the code size is just enough to hold all possible combinations). The results are summarised in table 3.3.

It is immediately evident that the performance of SICF is dramatically affected by the distribution of numbers in the sequence it encodes. In particular, it is outperformed by optimal fixed length coding when the distribution is uniform, and it performs much better than fixed length coding when the distribution is highly non-uniform. Thus, as a compression algorithm, SICF does not work particularly well, as for a good compression algorithm one would expect the result almost as good as for fixed length coding under any circumstances.

## 4. Conclusion

We demonstrated, that matrix index approach (MIC) and navigation index approach (HIPOS) to in-memory navigation of POS can be reformulated independently from the used coding technique. We found that fixed length coding has inherent limitations, namely, no matter which coding technique is to be used, the code size has to be comparable to the memory size required to store all OIDs in the maximum allowed path. We investigated SICFs as a variable-length coding technique, and found that it performs worse than fixed length coding for uniform distribution of numbers in the encoded sequence.

Our research indicates that alternative coding techniques, possibly taking advantage of compression algorithms, should be considered for matrix index and navigation index approaches to POS navigation. In particular, compression techniques taking into account the patterns formed by OIDs and references in a particular class of applications could be highly effective. Furthermore, it is desirable that POS architecture is formulated in such a way, that coding algorithms for in-memory navigation can be plugged in and configured by system administrators to assure the optimal operation of the system.

## References

- [1] ABITEBOUL, S., BUNEMAN, P., AND SUCIU, D. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
- [2] BECKMANN, N., KRIEGEL, H.-P., SCHNEIDER, R., AND SEEGER, B. The R\*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990* (1990), H. Garcia-Molina and H. V. Jagadish, Eds., ACM Press, pp. 322–331.
- [3] BEERI, C. A formal approach to object-oriented databases. *Data & Knowledge Engineering* 5, 4 (1990), 353–382.
- [4] BOHANNON, P., FREIRE, J., ROY, P., AND SIMÉON, J. From XML-schema to relations: A cost-based approach to XML storage. In *International Conference on Data Engineering* (San José, California, 2002), pp. 64–75.
- [5] CHO, E.-S., AND KIM, H.-J. LOD\*: A C++ extension for OODBMSs with orthogonal persistence to class hierarchies. *Information and Software Technology* 42, 5 (2000), 347–356.
- [6] DELIS, A., KANITKAR, V., AND KOLLIOS, G. Database architectures. In *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, Ed. John Wiley & Sons, 1999.
- [7] DEUTSCH, A., FERNANDEZ, M., AND SUCIU, D. Storing semistructured data with STORED. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pennsylvania, 1999), pp. 431–442.
- [8] FLORESCU, D., AND KOSSMANN, D. Storing and querying XML data using an RDBMS. *Bulletin of the Technical Committee on Data Engineering* (1999), 27–34.
- [9] GUTTMAN, A. R-trees: A dynamic index structure for spatial searching. In *SIGMOD’84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984* (1984), B. Yor-mark, Ed., ACM Press, pp. 47–57.
- [10] KEMPER, A., AND MOERKOTTE, G. Access support in object bases. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990* (1990), H. Garcia-Molina and H. V. Jagadish, Eds., ACM Press, pp. 364–374.
- [11] KUCKELBERG, A. Effiziente navigierende Zugriffe auf persistente Objektspeicher. Master’s thesis, Technical University of Clausthal, Germany, 1997.
- [12] KUCKELBERG, A. The matrix-index coding approach to efficient navigation in persistent object stores. In *Workshop on Foundations of Models and Languages for Data and Objects* (1998), pp. 99–112.

- [13] LISKOV, B., CASTRO, M., AND SHRIRA, L. Providing persistent objects in distributed systems. In *ECOOP'99 – Object-Oriented Programming* (1999), R. Guerraoui, Ed., pp. 230–257.
- [14] MAHESHWARI, U., AND LISKOV, B. Partitioned garbage collection of a large object store. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data* (1997), ACM Press, pp. 313–323.
- [15] OLDS, D. C. *Continued Fractions*. New Mathematical Library. Random House, New York, 1963.
- [16] SCHEWE, K.-D., AND THALHEIM, B. Fundamental concepts of object oriented databases. *Acta Cybernetica 11*, 1-2 (1993), 49–84.
- [17] SCHMIDT, A., KERSTEN, M., WINDHOUSER, M., AND WAAS, F. Efficient relational storage and retrieval of XML documents. In *The World-Wide Web and Databases – Third International Workshop* (Dallas, Texas, 2000), vol. 1997 of *LNCS*, Springer-Verlag, pp. 137–150.
- [18] SHANMUGASUNDARAM, J., TUFTE, K., ZHANG, C., HE, G., DEWITT, D. J., AND NAUGHTON, J. F. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of 25th International Conference on Very Large Data Base* (Edinburgh, Scotland, 1999), pp. 302–314.
- [19] SUBIETA, K. LOQIS: The object oriented database programming system. In *Next generation information system technology*, J. W. Schmidt and A. A. Stogny, Eds., vol. 504 of *LNCS*. Springer-Verlag, Berlin, 1991, pp. 403–421.
- [20] VALDURIEZ, P. Join indices. *TODS 12*, 2 (1987), 218–246.
- [21] ZEZULA, P., AND RABITTI, F. Object store with navigation acceleration. information systems. *Information Systems 18*, 7 (1993), 429–459.