December 1993

Efficient Lattice Operations Using N-grid Encoding

Wen-Gong Shieh
Chinese Culture University

Efficient Lattice Operations Using N-grid Encoding

Wen-Gong Shieh

Department of Information Management
Chinese Culture University
55 Hwa Kang Rd.,
Yang Ming Shan,
Taipei, Taiwan, R.O.C.
Tel: (02)861-0511 Ext. 539

Abstract

To support semantic structures organized as partial orders (e.g. inheritance hierarchies or class hierarchies), a compilation-time partial order encoding technique, called *compact encoding with modulation (CEM for short)*, has previously been developed to enable efficient computations of greatest lower bound (e.g. greatest common subclass) and least upper bound (e.g. least common superclass) at run time. Since most practical semantic structures can be organized as partial orders of small number of dimensions, this paper examines a different partial order encoding technique, called N-grid encoding. For partial orders of small dimensionality, the N-grid encoding is shown to outperform CEM for most practical semantic structures of extremely large size. Besides, the codes of a partial order using N-grid encoding can be modified efficiently if the partial order is changed dynamically at run time, provided that the changes are top-down refinements. That is, an element in the partial order is expanded to another partially ordered set of elements. This is another advantage of the N-grid encoding.

I. Introduction

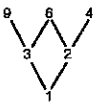
This paper is motivated by the work of H. Ait-Kaci et.al. [Ait89] where they observe that efficient run time computations of greatest lower bound (GLB) and least upper bound (LUB) in a partial order are important to the run time performance of a system with semantic structures of extremely large size. For example, finding GLB and LUB corresponds to finding the greatest common subclass and the least common superclass in a class hierarchy of an object-oriented system. They propose a compilation-time partial order encoding technique using compact binary code words to support efficient run time computations. To facilitate a trade-off between time and space for large partial orders, a code modulation (grouping) technique is introduced into their encoding. We call such an encoding *compact encoding with modulation (CEM for short)* throughout this paper. The code modulation technique reduces the code size of a partial order from $O(N^2)$ (using only compact binary code words) to O(N log N) (using CEM), provided that the partial order can be perfectly modulated [Ait89]. This, of course, has a run time penalty [Ait89].

In this paper, based on our observation that *most semantic structures have very small number of dimensions in terms of partial orders*, we append necessary data structures and algorithms to the partial order encoding technique in [Shie90], called N-grid encoding, to be another alternative to support efficient lattice operations. It should be noted that we need an alternative because "perfect" modulation may not be possible in CEM [Ait89]. For low-dimension partial orders, especially two-dimensional, the N-grid encoding is shown to outperform CEM for most practical semantic structures of extremely large size. Besides, the codes of a partial order using N-grid encoding can be modified efficiently if the partial order is changed dynamically at run time, provided that the changes are top-down refinements. That is, an element in the partial order is expanded to another partially ordered set of elements. This is another advantage of the N-grid encoding.

1.1 Assumptions and Background

Without loss of generality, we assume that all partial orders involved in our encoding have unique GLB and LUB for any two elements (i.e. we consider only lattices). For formal justification of this assumption, see [Ait89].

A **partial order** R on a non-empty set of elements S is a reflexive, transitive, and anti-symmetric relationship defined on elements of S. For example, consider S={1,2,3,4,6,9}, and the relationship D, where for m, n $\in$ S, m D n, if m divides n, then D is a partial order on S. Usually, the relationship is represented as m$\leq$n and the partial order is represented by an undirected graph called a **Hasse diagram** as shown in the following example for the partial order D.



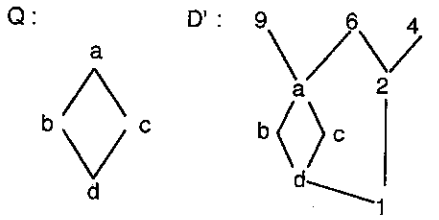
Elements m and n of S are **comparable** if either $m \leq n$ or $n \leq m$, otherwise, they are not comparable. In the above example, 1 and 3, as well as 1 and 9 are comparable; but 2 and 3 are not. A member $l \in S$ ($u \in S$) is a **lower (upper) bound** of a subset E of S, iff $l \leq x$ ($x \leq u$) for all $x \in E$. For example, in the above Hasse diagram, element 6 is a upper bound of elements 2 and 3. Similarly, element 1 is a lower bound of {2,3}. The partial order R on S is called a **lattice**, iff for every pair a, b $\in$ S, a greatest lower bound (GLB) and a least upper bound (LUB) exists for {a,b}.

A partial order T is a **total order** if every pair of elements in T are comparable. A partial order may be converted (nonuniquely) into a total order using topological sort. For example, T_1= (1, 3, 9, 2, 6, 4) and T_2= (1, 2, 4, 3, 6, 9) are two possible total orders for the partial order D given earlier. Note that some new relationships among the elements of S are created while obtaining the total order. The original partial order may be reconstructed using a **realizer**, which is defined as the set of total orders whose intersection yields the original partial order [Dush41]. The cardinality of the smallest realizer for a partial order R is known as the **dimension**, dim(R), of the partial order. In the above example, the realizer for the partial order D is $\{T_1, T_2\}$ and the dimension of D is 2. In this paper, the notation **n-POS** is used to refer to an n-dimension partial order.

A partial order P on G can be **refined** by replacing an element $u \in G$ by another partial order Q on H, leaving a new partial order R on $\{G-u\} \cup H$. We call such a change to a partial order a ***top-down refinement*** on the partial order. The refinement is defined as follows [Sand88].

$$
\begin{aligned}
R \;=\; & \{(x,x') \mid (x,x') \in P \text{ for all } x,x' \in G-\{u\}\} \\
\cup\; & \{(x,y) \mid (x,u) \in P \text{ for all } x \in G-\{u\},\ y \in H\} \\
\cup\; & \{(y,x) \mid (u,x) \in P \text{ for all } x \in G-\{u\},\ y \in H\} \\
\cup\; & \{(y,y') \mid (y,y') \in Q \text{ for all } (y,y') \in H\}
\end{aligned}
$$

For the above refinement, it is known that dim(R)= max{dim(P), dim(Q)} [Hiru51]. An example of refinement of the previous partial order D using the partial order Q is shown below. Note that the element 3 in D is expanded to (or replaced by) the partial order Q, resulting the new partial order D'.



1.2. A Brief Review of the N-grid Encoding

The N-grid encoding in the N-grid model represents an n-POS by an n-dimension vector space (known as an **N-grid** [Shie90]). Each element of an n-grid will be represented by an n-tuple of integers: $x = (x_1, x_2, ..., x_n)$. For any two elements $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, $x \le y$ (y dominates x) iff $x_1 \le y_1$, $x_2 \le y_2$, ..., $x_n \le y_n$.

The following example illustrates the n-grid concept using a 2-dimension partial order (2-POS) and a 2-grid representation (see Figure 1). In Figure 1, the members of P are A, B, C, D, E and their 2-tuples are A=(1,1), B=(2,4), C=(3,3), D=(4,2), and E=(5,5). P is a 2-POS, since the smallest realizer of P consists of two total orders, T_1 = (A,B,C,D,E) and T_2 = (A,D,C,B,E). The vector representation is obtained using T_1 for selecting the first integer and T_2 for the second integer of the 2-tuple. Given a size d realizer of a d-dimension partial order, the time complexity of encoding the partial order on an N-grid (N=d) is in O(nd) where n is the number of elements in the partial order [Shie90]. For detailed algorithms regarding N-grid encoding, see [Shie90].

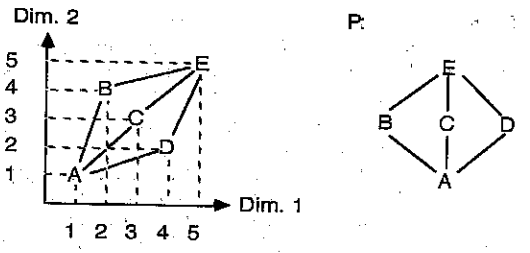


Figure 1. P on a 2-grid.

II. GLB and LUB Using N-grid Encoding

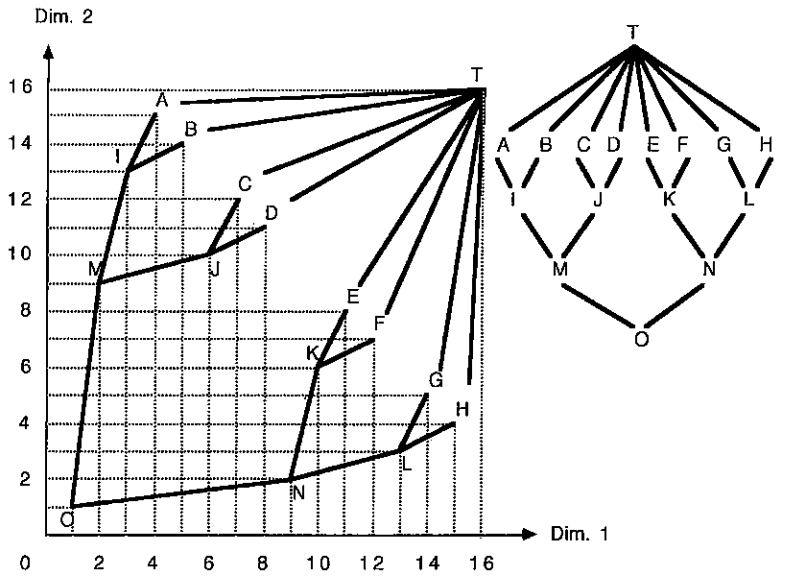
For simplicity, we will first use two-dimension partial orders in this section to demonstrate necessary data structures and algorithms for GLB and LUB operations using N-grid (i.e. 2-grid) encoding. We assume in the following discussions that the 2-grid encoding of each element in each partial order is given.

2.1. Data Structures and Algorithms for 2-POS

For example, to support efficient GLB and LUB operations, the 2-POS in Figure 2 uses three arrays: Dim2, LOW1 and UP1. These arrays are created as follows. First, sort all elements in the 2-POS into a list in ascending order based on their dimension 1 entries. We will use E(i) to denote the i-th element in the sorted list. Entries of dimension 1 are then used as indices of the three arrays. Each array element Dim2[i] stores the dimension 2 entry of E(i). Each array element LOW1[i] stores some value j, a pointer, such that E(j) is a lower bound of E(i), and E(j) has the largest *dimension 1* entry among all E(i)'s lower bounds. If E(i) has no lower bound, then LOW1[i] stores zero. Similarly, Each array element UP1[i] stores some value j, a pointer, such that E(j) is an upper bound of E(i) and E(j) has the smallest *dimension 1* entry among all E(i)'s upper bounds. If E(i) has no upper bound, then UP1[i] stores zero. The following algorithms create the UP1 array and the LOW1 array. Since each element E(i) requires exactly one PUSH and one POP in each of the following two algorithms, the time complexity of each algorithm is in O(n). (Note that the TOP function returns the top element in the stack.)

```
Create-UP1 Algorithm:
Input:  Array Dim2 and n (the number of elements).
Output: Array UP1.
begin
        PUSH(1);
        for i:=2 to n do
                while   STACK ≠ empty
                        and Dim2[TOP]<Dim2[i] do
                                UP1[TOP] := i;
                                POP(STACK)
                endwhile;
                PUSH(i)
        endfor;
        while STACK ≠ empty do
                UP1[TOP] := 0;
                POP(STACK)
        endwhile
end;
```

```
Create-LOW1 Algorithm:
Input:  Array Dim2 and n (the number of elements).
Output: Array LOW1.
begin
        PUSH(n);
        for i:=n-1 downto 1 do
                while   STACK ≠ empty
                        and Dim2[TOP]>Dim2[i] do
                                LOW1[TOP] := i;
                                POP(STACK)
                endwhile;
                PUSH(i)
        endfor;
        while STACK ≠ empty do
                LOW1[TOP] := 0;
                POP(STACK)
        endwhile
end;
```



UP1	2	3	4	16	16	7	16	16	10	11	16	16	14	16	16	0	pointers to upper bound
LOW1	0	1	2	3	3	2	6	6	1	9	10	10	9	13	13	15	pointers to lower bound
Dim2	1	9	13	15	14	10	12	11	2	6	8	7	3	5	4	16	entries of Dim. 2
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	entries of Dim. 1
	O	M	I	A	B	J	C	D	N	K	E	F	L	G	H	T	elements

Figure 2. Data Structures for Finding GLB and LUB.

Observe that, in the above 2-grid (Figure 2), E is in the left upper corner of G. Based on the definition of N-grid encoding (i.e. for any two elements $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, $x \le y$ (y dominates x) iff $x_1 \le y_1$, $x_2 \le y_2$, ..., $x_n \le y_n$), if we scan from right to left (following dimension 1) starting at E, and stop at the first element with dimension 2 entry less than that of G (i.e. the element N), we will always find a common lower bound of G and E (because E's dimension 1 entry is smaller than that of G, and G's dimension 2 entry is smaller than that of E). Since G and E is assumed to have unique GLB, and N is the first element on the left of E such that its dimension 2 entry is smaller than G's dimension 2 entry, N must be the GLB of G and E. That is, the dimension 1 entries of all other common lower bounds of G and E must be less than that of N due to the scanning order. Besides N, if any other common lower bound of G and E had dimension 2 entry greater than N's dimension 2 entry, it would be incomparable with N, and thus G and E would not have unique GLB, which would be a violation of the assumption. However, using this idea, to find GLB of D and E, we need to scan D,C,J,B,A,I,M, and O, almost half of the whole partial order. The purpose of the above data structure LOW1 is to skip those elements between D and O that have dimension 2 entries higher than that of D. This guarantees that each element scanned is a lower bound of D. Thus, following the pointers in LOW1, we need to scan only D,J,M and O. Such a scanning path is guaranteed to be part of a vertical path in the Hasse diagram of the partial order because the successor of each element in the scanning path is a lower bound of the element. Since LUB has symmetric behavior on the N-grid (i.e. 2-grid in the example) using UP1, we can now summarize the algorithms for GLB and LUB operations using N-grid encoding for 2-POS as follows.

GLB Algorithm:
Input: X and Y encoded as (x_1,x_2) and (y_1,y_2) in a 2-grid respectively .
Output: Z=GLB(X,Y) encoded as (z_1,z_2) in a 2-grid.
```
begin
        i := x1;
        if x1<y1 then
                while Dim2[i] > y2 do i:= LOW1[i] endwhile
        else    i := y1;
                while Dim2[i] > x2 do i:= LOW1[i] endwhile
        endif;
        return  (i,Dim2[i])
end;
```

LUB Algorithm:
Input: X and Y encoded as (x_1,x_2) and (y_1,y_2) in a 2-grid respectively .
Output: Z=LUB(X,Y) encoded as (z_1,z_2) in a 2-grid.
```
begin
        i := x1;
        if x1<y1 then
                while Dim2[i] < y2 do i:= UP1[i] endwhile
        else    i := y1;
                while Dim2[i] < x2 do i:= UP1[i] endwhile
        endif;
        return  (i,Dim2[i])
end;
```

2.2. Dynamic Top-down Refinement on Partial Orders

In this section, we demonstrate the necessary changes to the data structures using N-grid encoding at run time if top-down refinements on partial orders are performed. Again, we only show examples using 2-dimension partial orders. In Figure 3, a partial order P is encoded on an N-grid (N=2) with reserved quota. For example, the reserved quota of element C is 10 through 14 in dimension 1, and 8 through 12 in dimension 2. Observe that each element whose 2-tuple is of the form: (d1,d2), 10≤d1≤14, 8≤d2≤12, using 2-grid encoding, has the same relationship with A,B,D,E as C does. Expanding element C into another partial order Q (i.e. refinement of C in P into Q) is possible if Q can be encoded using the reserved quota of C. That is, each element of Q can be encoded by a distinct 2-tuple of the form: (d1,d2), 10≤d1≤14, 8≤d2≤12. It should be noted that such a refinement will not change the 2-tuples of other elements (i.e. the 2-tuples of A,B,D, and E remain the same).

As an example, the partial order P in Figure 3 is refined by expanding the element C into a partial order Q as shown in Figure 4. Figure 5 demonstrates the necessary changes to the data structures before and after the refinement of C in P into Q. Since the algorithms for making these changes are easy to construct, we omit the algorithms in this paper.

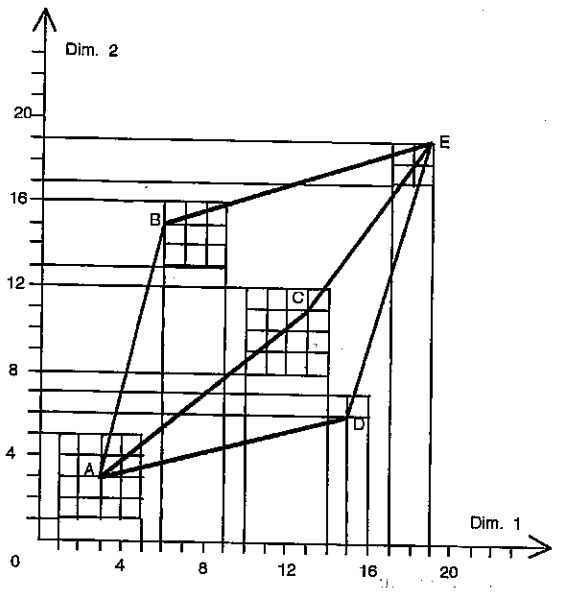


Figure 3. P in a 2-grid with reserved quota.

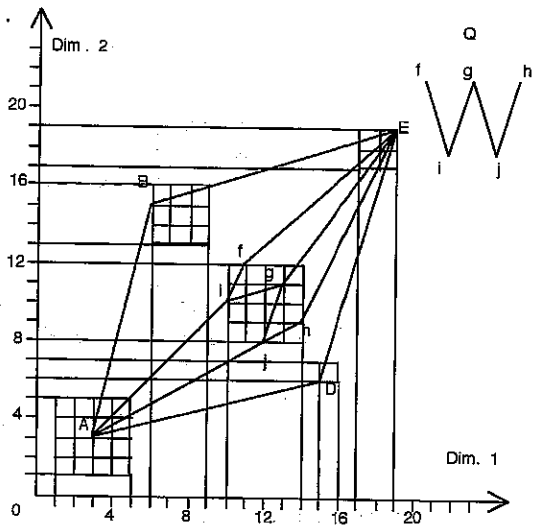


Figure 4. Refinement of C in P into Q.

Data Structures of P before Refinement:

UP1			6			19							19	19				0		pointers to upper bound
LOW1			0			3							3	3				15		pointers to lower bound
Dim2			3			15							11	6				19		entries of Dim. 2
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	entries of Dim. 1
			A			B							C		D			E		elements

Data Structures of Q

UP1	11	0	13	0	0
LOW1	0	10	0	12	12
Dim2	10	12	8	11	9
Index	10	11	12	13	14
	i	f	j	g	h

Data Structures of P after Refinement:

UP1			6			19				11	***19***	13	***19***	***19***	19			0		pointers to upper bound
LOW1			0			3				***3***	10	***3***	12	12	3			15		pointers to lower bound
Dim2			3			15				10	12	8	11	9	6			19		entries of Dim. 2
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	entries of Dim. 1
			A			B				i	f	j	g	h	D			E		elements

Figure 5. Data Structures -- Refinement of C in P into Q.

The data structures for partial order Q is given based on the reserved quota and the assumption that only the elements of Q are considered. After the refinement is performed, two steps are taken to change the data structures of P. Step 1, the contents of the data structures with indices 10 through 14 (note that these indices corresponds to the reserved quota of C) are modified. That is, copy the contents of data structures of Q into corresponding positions in the data structures of P. Step 2, the 0's in the newly copied portion of the data structures of P must be modified. That is, UP1[11], UP1[13], UP1[14], LOW1[10], and LOW1[12] must be changed. We use ***Italic and bold*** format to highlight the new values in those positions. For 0's in UP1, we modify them into 19's, where 19 is the original value in UP1[13] before the refinement. Note that 13 is the index corresponding to the expended element C. Similarly, for 0's in LOW1, we change them into 3's where 3 is the original value in LOW1[13]. These two steps complete the necessary changes to the data structures of P after the refinement.

2.3. Evaluation

Let's consider first the total number of bits required to encode a partial order. We have the space required for encoding as follows.

	Lowest dimension	Highest dimension
#bits(N-grid encoding):	$n\lceil \log_2 n \rceil$	$(n^2/2)\lceil \log_2 n \rceil$
#bits(compact encoding):	$n(n-1)$	$n\lceil \log_2 n \rceil$

Next, let's consider the space requirement of CEM and N-grid encoding to support efficient lattice operations. With modulation, the total number of bits required for CEM is at least $n(\log_e n)n^{(1/\log_e n)}$ (the lower bound exists in partial orders that are perfectly modulated) [Ait89]. With the data structures needed to support GLB and LUB operations, based on Hirugachi's theorem [Hiru55,Boga73], the total number of bits required is in $O(n^2 \log n)$ for N-grid encoding, which is worse than that of CEM. However, when the dimension d is small and fixed, especially when d=2 (i.e. two-dimension), the N-grid encoding (where N=d=2) is compatible with CEM (i.e. in O(n log n)).

Now, consider the time required to find GLB between elements X and Y. The time required for compact encoding without modulation is proportional to the length of the compact code. With modulation (assuming that the partial order is perfectly modulated), the time complexity using CEM is in O(log n) [Ait89].

The time required for GLB using N-grid encoding is constant in case of comparable X and Y when the dimension of the encoded partial order is fixed (e.g. 2 in a 2-POS). In case of incomparable X and Y, the time required is at most proportional to the longest vertical path in the Hasse diagram of the partial order. The longest vertical path in the Hasse diagram is $\log_2 n$ if the Hasse diagram is a complete binary tree, and is about $F^{-1}(n)$, where F(n)=n!, if the Hasse diagram is a tree of exponential nature (the branching factor at depth D was D+1). Therefore, the performance for incomparable X and Y is better than O(log n) in this case.

As mentioned in [Ait89], in practice, the Hasse diagrams of the partial orders we have encountered are generally trees of exponential nature with extra links among nodes. Therefore, the N-grid encoding outperforms CEM in most practical semantic structures of extremely large size for GLB operations (needs only about $F^{-1}(n)$ steps, where F(n)=n!). Besides, since the performance for both GLB and LUB are the same using N-grid encoding, and GLB operations are more efficient than LUB operations using CEM [Ait89], we can conclude that N-grid encoding outperforms CEM in most practical semantic structures of extremely large size for both GLB and LUB operations using compatible space, compared to the time and space used in CEM.

III. Conclusions

In conclusion, we have presented a compilation-time partial order encoding technique, the N-grid encoding, and its necessary data structures and algorithms to support efficient run time GLB and LUB operations. The size of the space used in the encoding and data structures is acceptable and compatible with that of CEM, especially for two- (or low-) dimension partial orders (i.e. lattices).

The major advantages of our approach, compared to CEM, include (1) using N-grid encoding, the GLB and LUB operations are more efficient than using CEM for most practical semantic structures of extremely large size, (2) the space requirement is compatible with that of CEM (and is better than that of CEM in the case of low-dimension partial orders that are not perfectly modulated), and (3) efficient run time changes to partial orders and its corresponding data structures are possible using N-grid encoding. Since some partial orders cannot be perfectly modulated, we believe that our approach provides a valuable alternative, other than CEM, to support efficient lattice operations using acceptable space. Besides, generalized algorithms (not presented in this paper) also suggests an efficient way of information organization and retrieval.

IV. REFERENCES

[Ait89] H. Ait-Kaci, R. Boyer, P. Lincoln and R. Nasr. "Efficient Implementation of Lattice Operations", *ACM Transactions on Programming Languages and Systems*, vol. 11, No. 1, January 1989, pp. 115-146.

[Boga73] K. Bogart and W. Trotter. "Maximal dimensional partially ordered sets", *I, Discrete Math. 5* (1973), pp. 21-32.

[Dush41] B. Dushnik and E.W. Miller. "Partially ordered sets", *Amer. J. Math. 63*, (1941), pp. 600-610.

[Hiru51] T. Hirugachi. *On the dimension of partially ordered sets.* Science Rep. Kanazawa Univ. Vol. 1, 1951, pp. 77-94.

[Hiru55] T. Hirugachi. *On the dimension of orders.* Science Rep. Kanazawa Univ. Vol. 4, 1955, pp. 1-20.

[Sand88] Ravinderpal S. Sandhu. "The NTree: A Two Dimension Partial Order for Protection Groups", *ACM Transactions on Computer Systems*, Vol. 6, No. 2, May 1988, pp.197-222.

[Shie90] W.-G. Shieh, B.P. Weems and K. Kavi. "An N-grid Model for Group Authorization", *Sixth Annual Computer Security Applications Conference*, Tucson, Arizona, Dec. 3-7, 1990.