Journal of
Chemical
Information and
Computer Sciences

J. Chem. Inf. Comput. Sci., 1996, 36(1), 46-53, DOI:10.1021/ci9500647

SUPPLEMENTARY MATERIAL

Software Implementation of the Partitioned Data Space

In implementing the partitioning algorithm in computer software, each box in the data space which contains library vectors must be represented by a data structure in a way that allows each vector to be associated with that box and each box to be accessible in some manner by neighboring boxes. The C-language data structure used in this work represents the top level of boxes by a multidimensional integer array called *grid* which is dimensioned to the vector dimensionality used, *n* (e.g., a seven-dimensional array in the work presented). The number of elements required in each dimension of *grid* is dictated by the bin size used and the maximum and minimum values in that dimension across the set of library vectors. This array thus has one element for each box in the partitioned data space. Each entry in *grid* contains the value -1 if no library vectors have been assigned to that box or an integer which points to an entry in a one-dimensional array of linked lists, *lists*, in which each list contains a node for each of the vectors in a given box in the data space.

All deeper levels of boxes are represented by an *n*+1-dimensional array, *multi_grid*, where the first dimension distinguishes the data space being stored since every partitioned box requires a dedicated *n*-dimensional array to represent its partitioning, and the last *n* dimensions correspond to the dimensions of the data space. The number of elements in dimensions 2 through (*n*+1) of *multi_grid* parallels *grid*. In a manner analogous to *grid*, *multi_grid* also points into *lists* for each box in the given data space which contains library vectors. Figure 1 displays a representation of *lists* for the case in which seven levels were used, the top level bin size was 8.0, and the density threshold was 500 vectors. Several linked lists showing a few of the library vectors in given boxes in the data space have been displayed. These will be referenced below as the assignment of a library vector to its

appropriate box is traced.

Two one-dimensional arrays, *list_sum* and *grid_ptr*, are needed to navigate between various levels of the data space. These arrays are parallel to *lists*. Portions of these arrays are displayed in Table 1 along with corresponding elements of *grid* and *multi_grid*, as well as other information necessary for their usage. The first column in the table indicates which data space in *multi_grid* corresponds to each row. The first four rows in the table do not have entries for column 1 because they correspond to boxes in the top level of the data space, all of which are located in *grid*. The subsequent rows correspond to boxes in *multi_grid*, and thus have entries in column 1. Columns 2-8 give the bin numbers of the corresponding dimensions 1-7 for a given box in the data space defined in column 1. The ninth column, labelled *list_ptr*, is the pointer into the data structure, *lists,* from the box in *grid* or *multi_grid* defined by the first eight columns and is the actual number stored in *grid* or *multi_grid*. This can be considered a box number in the data space. *List_sum*, the tenth column, contains the number of library vectors in the corresponding box. This value is used for comparison to the density threshold to determine if a given box should be partitioned into smaller boxes. *Grid_ptr*, the eleventh column, is used to point from a dense box in a given level to the data space in *multi_grid* in which this particular box has been partitioned. Every entry in *grid_ptr* is initialized to 0. After the library vectors have been assigned to the correct lists, the only entry in *grid_ptr* which points to a box in *multi_grid* and has a value of 0 is the entry which points to the zeroth row of *multi_grid*. The last column, labelled *recurse_level*, is included to show the level of the data space corresponding to entries in *list_sum* and *grid_ptr*. The top level of the data space is denoted as level 0.

The library vectors are processed sequentially. The box in the seven-dimensional array, *grid*, to which a given library vector belongs is calculated as a seven-element index in

which the individual elements, b_i, are defined as

$$b_i = (e_i - min_i)/s \quad (1)$$

where e_i is the value of the data vector in dimension i, min_i is the minimum value in that dimension across the library of data vectors, and s is the predefined bin size. Each value of b_i is truncated to an integer and thus serves as an element number in the corresponding dimension of *grid*. For the partitioning structure and database used in this example, the first library vector, labelled vector 0, maps into box [13, 20, 1, 0, 0, 0, 0] of *grid*. If a box has never been accessed before, a new linked list is begun in *lists*, and the entry in *grid* is assigned the value of the pointer into *lists*. This is the case for the first library vector, and a list is begun for the first entry in *lists*. The corresponding value of *list_ptr* is therefore 0, and in Figure 1, it is seen that library vector 0 occupies the first node in the first linked list in *lists*. The value in box [13, 20, 1, 0, 0, 0, 0] of *grid* is changed from -1 to 0. In Figure 1, it can be seen that library vector 2 also begins a new linked list, and box [13, 24, 2, 0, 0, 0, 0] of *grid* is assigned the current value of *list_ptr* which is 1. When a library vector being processed maps into a box in the data space which already contains one or more library vectors, that library vector is simply added to the end of the appropriate linked list in a new node. One can see from Figure 1 that library vectors 1, 3, and 6 are examples of this case.

The assignment of a library vector begins by an assignment to a box in the top level and continues to deeper levels until the vector reaches an assignment to a box at some level which is not dense or until the maximum number of allowed levels has been reached. The vector is added to the proper list at each level corresponding to the box in the particular data space into which the vector maps. Although it is redundant to store the vector in multiple lists, the memory cost is acceptable and this procedure simplifies both building and searching the data structure.

The actual creation of the data structure proceeds one level at a time. All of the library vectors are assigned to the proper box in the top level. Then, each dense box in the top level is partitioned into the second level of boxes, and all the library vectors in the list for that dense box in the top level are assigned to their proper box in the second level. This process is repeated at each level until no boxes are dense or until some maximum level of partitioning has been reached. From eqn. 1, it can be seen that the minimum value of each dimension in the data space is needed to calculate the box in a given data space into which a particular library vector maps. A data structure, *smin*, which is parallel to *multi_grid* holds these minimum values.

The specific example of the assignment of library vector 743 to its proper box in the data structure will be given to help explain Table 1. Through the use of eqn. 1, this vector maps into box [13, 24, 1, 0, 0, 0, 0] of *grid.* Table 1 shows that the list of library vectors for this box is maintained in entry 49 of *lists* (i.e., *list_ptr* = 49). In Figure 1, vector 743 is indicated as the fourth entry in *lists*[49]. After all library vectors have been assigned to linked lists in *lists* for the top level of boxes, the assignment of vectors to the second level of boxes is begun. It is found that *lists*[49] has greater than 500 nodes since *list_sum*[49] is 2680. *Grid_ptr*[49] is assigned the value 31, since 31 denotes the next unused data space in *multi_grid*. Library vector 743 maps into box [1, 1, 1, 0, 0, 0, 0] in data space 31 of *multi_grid* as seen in Table 1. The integer pointer for this box into *lists* is 801, and the number of nodes in the linked list for *lists*[801] is 1752 as seen in *list_sum*[801]. Thus, the box in data space 31 into which library vector 743 maps is dense and must be partitioned. The next unused data space in *multi_grid* is 84, as indicated by *grid_ptr*[801]. Therefore, box [1, 1, 1, 0, 0, 0, 0] of *multi_grid*[31] is partitioned in data space 84 of *multi_grid*. Library vector 743 maps into box [1, 0, 0, 1, 0, 0, 0] of *multi_grid*[84]. One can continue following

the mapping of this library vector into ever smaller boxes in the data space until a non-dense box is reached. The sequence goes from box [1, 0, 0, 1, 0, 0, 0] of *multi_grid*[84] to *lists*[1346], and then through *grid_ptr*[1346] to data space 125 in *multi_grid*. In data space 125, the corresponding box is [0, 1, 1, 0, 0, 0, 0]. The sequence continues to *lists*[1829], through *grid_ptr*[1829], box [1, 1, 1, 0, 0, 0, 0] of *multi_grid*[156], and finally to *lists*[2387]. *Lists*[2387] is non-dense (i.e., *list_sum* = 315), and thus the assignment of library vector 743 is completed. By following this process for each of the vectors in the library, the data structure is built and can then be searched to find the closest library matches for a given test vector.

Implementation of the Database Search

The first step in finding the closest matches for a given test vector is to find the smallest target box in which the test vector lies. This is the smallest box in the data space which contains the test vector and at least p library vectors, where p is the density threshold used to define a box as dense. This is implemented in the software by use of a recursive subroutine which finds the target box to which the test vector belongs at the particular level of the current recursive call. If the target box at that level is dense, the recursive routine is called again to find the target box at one level deeper in the data space. The second step in the search procedure involves a traversal of each of the neighboring boxes of the smallest target box.

The searching of the data space for the closest matches will be illustrated for vector 11 in the test set using the data space partitioning discussed in the previous section. This test vector encodes the chemical environment for one of the carbon atoms in naphthalene as [101.65, 175.39, 12.63, 1.76, 0, 0, 0]. This vector maps into box [13, 24, 1, 0, 0, 0, 0] of *grid* at the top level. It ultimately maps into box [1, 1, 1, 0, 0, 0, 0] of *multi_grid*[156], which is the

same box referenced above in the assignment of library vector 743.

Figure 2 displays how the test vector maps into the ultimate target box for each of the seven dimensions and will be used to illustrate the searching of the neighboring boxes. It is important to note that the dimensions are not independent, but are being considered independently for illustrative purposes. The consequence of all the dimensions being involved simultaneously in the searching process will be given at the end of this example, and it will be shown how the true case differs from the example case.

In Figure 2, each of the dimensions in the data space is represented by a separate box in the figure, and each box is labelled accordingly. The numbers at the left side of each box correspond to the position of the bin along the axis for that dimension. For example, the top level box representing the first dimension ranges from 94.72 to 102.72. This is a distance of 8.00 which corresponds to the bin size at the top level. This particular range corresponds to bin 13 of the first dimension. Each box is divided into four columns of partitioned boxes, each of which represents the data space at a deeper level. At each successive level, each box is subdivided into two smaller boxes. Each box in a pair of boxes at a given level is labelled by 0 or 1 in the upper left hand corner of the box. In addition, each of the boxes in the entire set of boxes for a given level is labelled sequentially in the upper right hand corner. The target box for each level is labelled with a T and relevant neighboring boxes are labelled with an N.

The process by which the smallest target box is found can be illustrated by use of Figure 2. Consider the box representing the first dimension of the vector. As already stated, this box represents bin 13, and one can confirm this by calculating the bin based on the range labels of 94.72 to 102.72. This box meets the density threshold, and thus the target box at the next level is sought. The value for the first dimension, 101.65, falls between 98.72 and

102.72, and thus, the target box at the next level is box 2 (labelled in the upper right hand corner) and is labelled with a T. This box also meets the density threshold, and the mapping continues to the next level. The mapping ends at the fifth overall level with a non-dense box. The lowest-level target box has the sequence number 14 in the figure and corresponds to the boundaries of 101.22 and 101.72. The search begins by comparing the test vector to each of the vectors in this target box.

The process by which neighboring boxes are searched is also illustrated in Figure 2. The boxes labelled with an N indicate the neighboring boxes which must be searched at each level. A one-dimensional box always has two neighbors, and thus, in the deepest level, each of the boxes on either side of the target box is labelled with an N. The condition for searching and finding the correct closest matches is that for a given target box, all neighboring boxes of equal size must be searched. However, as discussed in the main body of the paper, smaller boxes can be searched if they have sufficient vectors.

The algorithm for negotiating the various levels of neighboring boxes is based on establishing boundary values for use in defining a box as a neighbor. When the ultimate target box has been located (the target box at the deepest possible level with sufficient library vectors), a minimum and a maximum boundary value are stored for each dimension. The minimum value is calculated as the minimum boundary of the target box at the given dimension minus the step size of the deepest level of the data space. The maximum value is calculated as the maximum boundary of the target box at the given dimension plus the step size of the deepest level. These values are used to determine whether a given neighboring box in a higher level must be searched. If the maximum or minimum value for a given dimension falls inside the box under consideration, that box must be searched. The minimum boundary value for target box 14 for dimension 1 is set at 101.095, and the maximum

boundary value is set at 101.845.

After the search of the target box and the minimum and maximum boundary values have been set, the searching of neighboring boxes can begin. Box 13 must be searched as it is a neighboring box to the target box and falls within the minimum and maximum boundary values. Following this search, the recursive subroutine returns to the next higher level of the data space. The search of box 7 (parent of box 13) is considered completed upon the return from the lower level, and box 8, which is the neighbor of box 7, is next considered for searching. Since the maximum boundary value falls inside box 8, box 8 must be searched. If box 8 is dense, recursion will take place, and box 15 will be searched instead, provided it contains the minimum number of vectors required for the search of smaller neighboring boxes.

Upon completion of this search, the search of box 8 is considered complete, and the recursion ascends to the next higher level. Box 3 in this level is next considered for searching. Since the minimum boundary value does not fall in box 3, it is not searched. Likewise, when the next higher level is reached, box 1 is considered for searching, but since the minimum boundary value does not fall within this box, it is not searched. The neighboring boxes in the top level are next considered, but are not searched since the minimum and maximum boundary values do not fall within any of these boxes. The assignments of the target boxes and neighboring boxes for the remaining six dimensions are displayed similarly in Figure 2.

The discussion presented above treated each dimension individually. However, in practice, the dimensions are not searched individually. In the partitioning scheme used in this example, a box in seven dimensions actually has 2186 neighbors. There is just one smallest target box in the seven-dimensional data space, uniquely identified by the set of seven bin

numbers derived from the use of eqn. 1. The seven-dimensional coordinates of the occupied neighboring boxes associated with each box are calculated based on the target box coordinates stored in *smin* for each level of the data space, allowing the analysis of the maximum and minimum boundary values described above to be conducted in the actual seven-dimensional space.

As the target and neighboring boxes are encountered, Euclidean distances are computed between the test vector and the library vectors in the selected boxes. A sorted list of the desired p smallest distances and corresponding library numbers is maintained as the search proceeds.

Table 1

Parameters Describing Storage of Selected Boxes in the Partitioned Data Space

data space[a]	Bin Number							list_ptr	list_sum	grid_ptr	recurse _level
	1	2	3	4	5	6	7				
	13	20	1	0	0	0	0	0	41571	0	0
	13	24	2	0	0	0	0	1	602	1	0
	12	30	1	0	0	0	0	2	645	2	0
	13	24	1	0	0	0	0	49	2680	31	0
0	1	1	0	0	0	0	0	521	26214	45	1
31	1	1	1	0	0	0	0	801	1752	84	1
84	1	0	0	1	0	0	0	1346	870	125	2
125	0	1	1	0	0	0	0	1829	715	156	3
156	1	1	1	0	0	0	0	2387	315	0	4
181	1	1	1	0	1	0	0	2895	2	0	6

[a]No entry indicates this box lies in the top level of the partitioned data space.

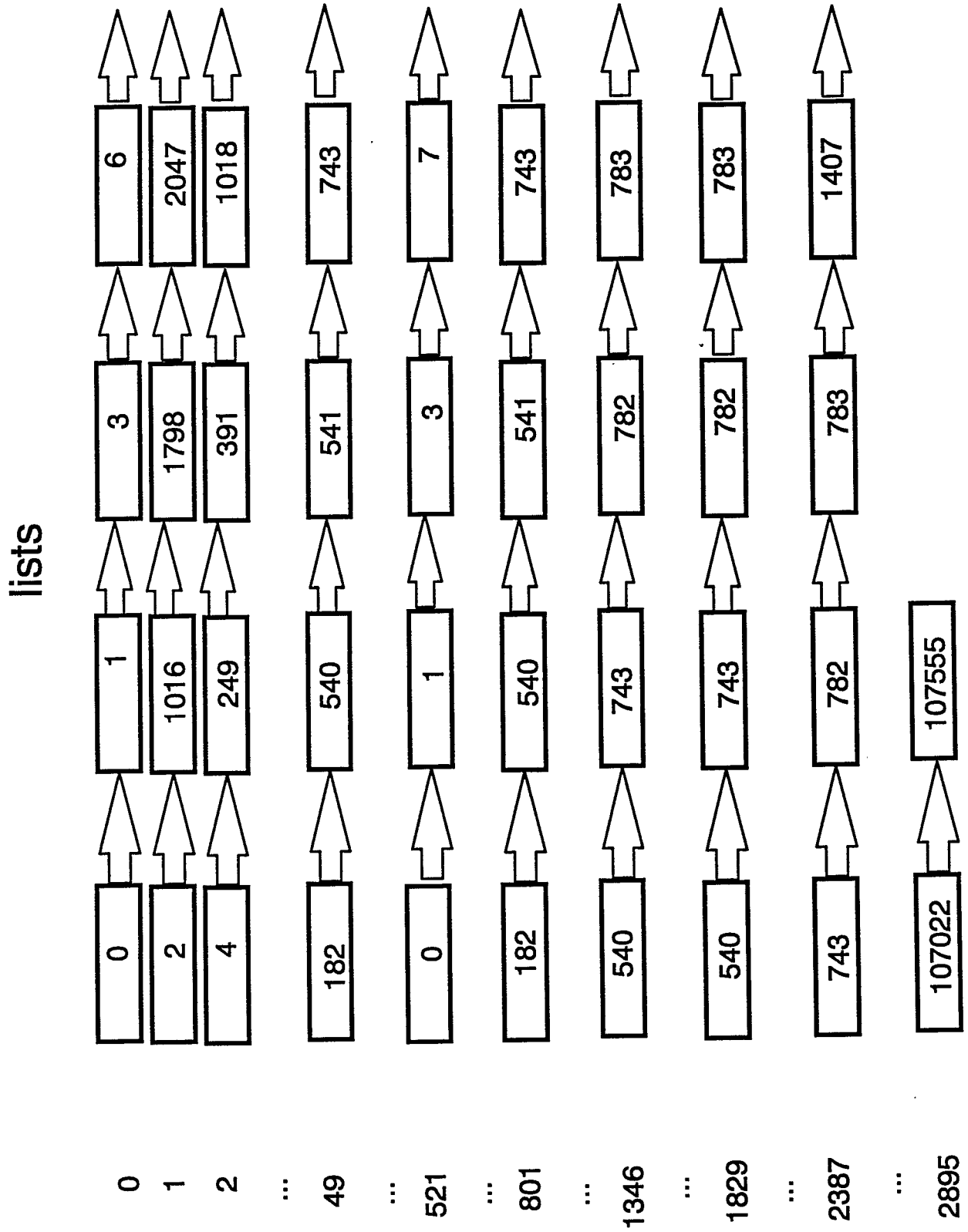
FIGURE CAPTIONS

Figure 1: Illustration of storage in the array of linked lists, *lists*. Entries are included that correspond to the boxes in the data space described in Table 1. The number displayed to the left of each list denotes the box number described by that list.

Figure 2: Illustration of the mapping of target vector 11 into each dimension of the partitioned data space. Each dimension is displayed separately as a box, and the four levels of the data space below the top level are denoted by the columns within each box. At each level, the target vector maps into the box labeled, T. The relevant neighboring boxes for the search are labeled, N.

Figure 1

U-53-12



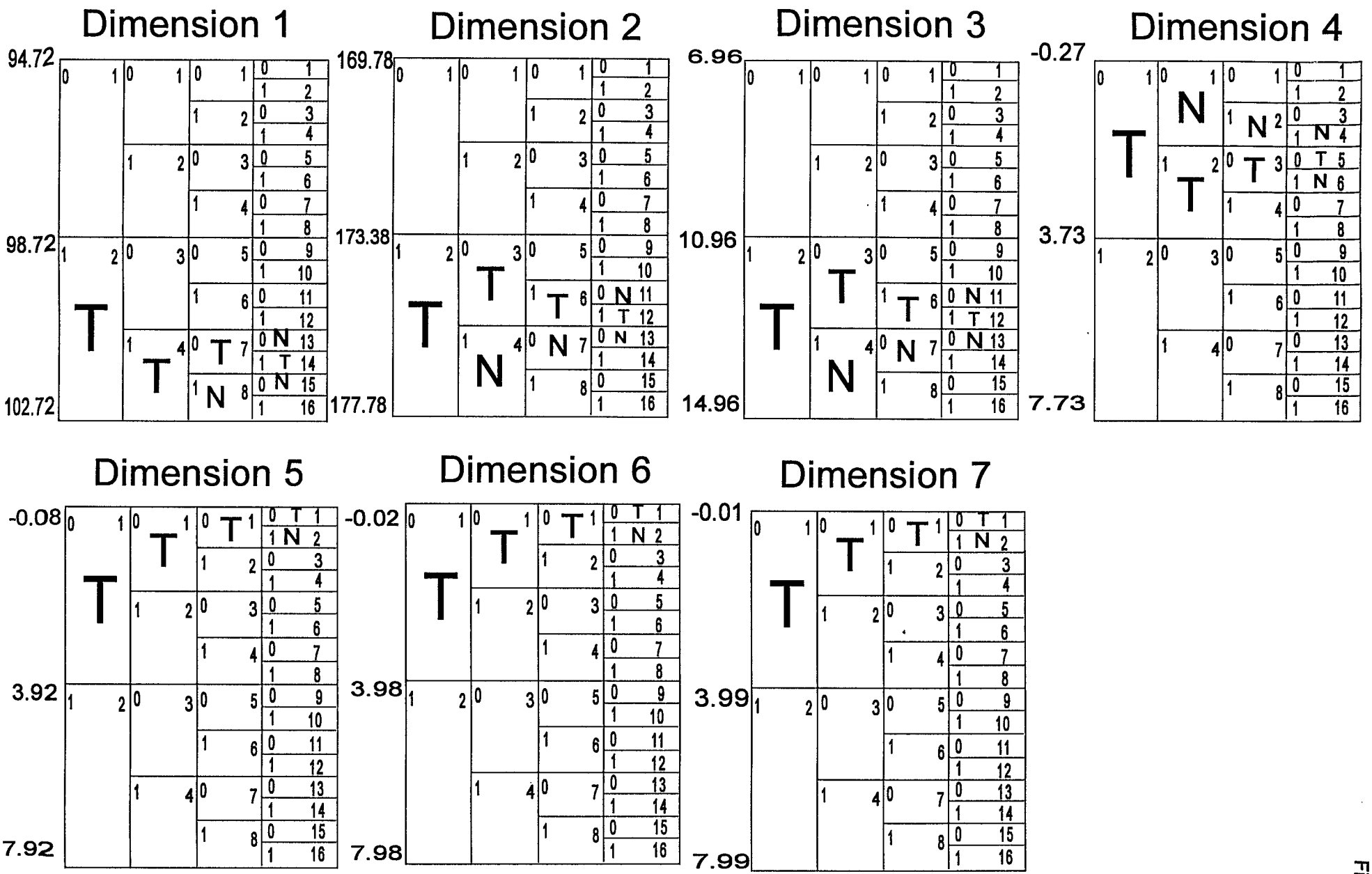


Figure 2
U-53-13