ESEL/2: A Program for Selecting the Most Representative Training Events for Inductive Learning

by

Stephen A. Crann

January 1983
ESEL/2: A Program for Selecting the Most Representative Training Events for Inductive Learning

by

Stephen A. Cramm

January 1983

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ISG 83-1
File No. UIUCDCS-F-83-901

This work was supported, in part, by a National Science Foundation Grant No. MCS-82-05166, and in part, by an Office of Naval Research Grant No. N00014-82-K-0186.
CONTENTS

ACKNOWLEDGMENTS

1. ESEL2 Methodology
   1.1 YL Events and Descriptors
   1.2 Outstanding Representatives
      1.2.1 Distances Between Events
      1.2.2 Continuous v.s. Quantized Measure
   1.3 ESEL2 Algorithm
      1.3.1 Input Initial Data
      1.3.2 Computation of Structure Distances
      1.3.3 Reduction of Events

2. ESEL2 Algorithm Implementation
   2.1 Input Initial Data Procedure
      2.1.1 Input Relational Table Order
      2.1.2 Undefined Names and Continued Tables
   2.2 Computation of Structure Distances
      2.3 Reduction of Events Procedure
      2.3.1 Sorting of Events
      2.3.2 Computations with Parameters
      2.3.3 Partitioning
      2.3.4 Reduction, Near Neighbors and Alternates
      2.3.5 Output of Results
   2.4 Computing System Memory Remaining

3. KPK-BTM ESEL2/GEM Experiment
   3.1 KPK-BTM Data
   3.2 ESEL2/GEM Experimental KPK-BTM Data
   3.3 ESEL2 Runs
   3.4 Gem Runs

4. ESEL2 User Guide
   4.1 ESEL2 Input--EVENTS
      4.1.1 ESEL2 Requirements for Relational Tables
      4.1.2 ESEL2 Parameters
      4.1.3 ESEL2 Parameter Restrictions
      4.1.4 ESEL2 Parameter Default Values, Recommended Values
      4.1.5 Structure Table Format
   4.2 ESEL2 Output--ESELOUT
   4.3 ESEL2 Output--REPORT
   4.4 ESEL2 Temporary Files
ACKNOWLEDGMENTS

Mike Stauffer gave me a great deal of assistance and help. Robert Stepp provided me with needed files, information, and suggestions. He also supplied formulas for the structured and linear variables, as well as the time to explain them to me. Additionally, he taught me a variety of miscellaneous items, all of which contributed greatly to the final program and report.

Partial support of this work was provided by the National Science Foundation, Grant No. MCS-82-05166 and the Office of Naval Research, Grant No. N00014-82-K-0186.
This document explains the methodology and the implementation of the program ESEL2, a successor to the original ESEL program. ESEL2 supports a variety of improvements over ESEL and has the additional ability to input and output data in the form of relational tables, the method of data I/O used by Gem, Generalization of Example by Machine (AQ12). This document replaces any previous ESEL2 documents.

1. ESEL2 Methodology

The purpose of ESEL2 is to select the most representative subset of events from a large number of \( VL_1 \) events, in support of programs such as Gem. The necessity of an ESEL program is evident by the long run times required of Gem when processing a large number of events; on the order of 200 or more.

The theoretical background for ESEL2 can be found in "On the Selection of Representative Samples from Large Relational Tables for Inductive Inference" by Ryszard S. Michalski [Michalski 75], or in the program description of ESEL/AQ11 [Michalski, Larson 78]. I will, therefore, summarize only those sections which are important for understanding ESEL2 implementation.

1.1 \( VL_1 \) Events and Descriptors

The \( VL_1 \) events mentioned previously, are represented by descriptors (variables) which have a specific value for a specific event, as well as a domain in which these values must lie.
The program ESEL2 recognizes four types of descriptors or variables, only two of which are presently recognized by Gem. The two recognized by Gem are nominal and linear:

**Nominal Descriptors** comprise sets whose domains have no order. Examples of nominal descriptors are one's hair color and the model of a car.

**Linear Descriptors** comprise sets whose domains are linearly ordered. Examples are a person's age and the year a car was manufactured.

The two additional descriptors which ESEL2 can recognize are the type structured-nominal and the type structured-linear:

**Structured-nominal Descriptors** comprise sets whose domains consist of nominal values within a hierarchy. These domains are referred to as partially ordered sets or posets. They are referred to as "structured descriptors" in [Michalski 75]. An example of a structured-nominal descriptor is a person's position in a hierarchy.

**Structured-linear Descriptors** comprise sets whose domains are tree structures. The leaf values of each tree are linearly ordered. The branch node values are computed from the leaves; they are the average of their son values. An example of a structured-linear descriptor is the distance of the United States from the equator. The cities are leaves, the counties, states and regions are higher.
1.2 Outstanding Representatives

ESEL2, like its predecessor, implements an algorithm called "Outstanding Representatives" (see [Michalski 75]), or OR in order to reduce the number of \( V_{1} \) events.

Basically, the OR method reduces the input event set by choosing events which are most distant from one another. This will result in a representative subset which will delineate the outer boundaries of the original set. Thus, if the original event set (class of events) describes a circle, the chosen subset will line the perimeter of the circle.

One drawback to this method is that it is extremely sensitive to erroneous events which differ significantly from the main area of input events. In order to offset this undesirable quality of OR, ESEL2 can specify that each representative event chosen have a number of near neighbors. If an event does not, it is not chosen and another is found.

1.2.1 Distances Between Events

As indicated above, the OR method must be able to determine the distances between two events. In ESEL2, this is done by a method R. S. Michalski refers to as continuous measure. Continuous measure is simply the summation of the distances between the values of each descriptor (variable) multiplied by a weight (cost) given to that descriptor.

\[
d_c(e_1, e_2) = \sum_{i=1}^{n} w_i d(x^1_i, x^2_i),
\]

where \( w_i \) is a weight (cost) associated with the descriptor (variable) \( x_i \).
In the above formula, \( e_1 \) represents one event, \( e_2 \) the other. \( x_i \) is a variable (descriptor) represented in the events and \( x^1_i \) is the value of that variable in event one. \( n \) is the total number of variables (descriptors) and \( d(x^1_i, x^2_i) \) is the distance between the values of variable \( x_i \).

Each type of variable (descriptor) has its own method for the computation of distances. The first three methods presented below are from [Michalski 75]. The last was presented to the author by Robert Stepp:

**Linear Variable Distance**:

\[
d(x^1_i, x^2_i) = \frac{\text{abs}(x^1_i - x^2_i)}{d_i}, \quad 1 \leq i \leq n_1
\]

where the domain of each linear variable is represented by the set \( \{1, 2, 3, \ldots, d_i - 1\} \), \( d_i \) is the cardinality (level) of the domain of \( x_i \), and \( n_1 \) is the number of linear variables.

**Nominal Variable Distance**:

\[
d(x^1_i, x^2_i) = \begin{cases} 1, & \text{if } x^1_i \neq x^2_i \\ 0, & \text{otherwise} \end{cases}, \quad 1 \leq i \leq n_n
\]

where \( n_n \) is the number of nominal variables.

**Structured-nominal Variable Distance**:

\[
d(x^1_i, x^2_i) = \frac{\text{NB}}{MNB}, \quad 1 \leq i \leq n_{sn}
\]

where NB is the shortest path length (number of branches) linking \( x^1_i \) with \( x^2_i \), MNB is the greatest NB in the structure, and \( n_{sn} \) is the number of structured-nominal variables.
Structured-linear Variable Distance:

\[ d(x^1_i, x^2_i) = \frac{\text{abs}(\overline{AL}^1_i - \overline{AL}^2_i)}{\text{MAL}_i}, \ 1 \leq i \leq n_{sl} \]

where \( \overline{AL}^1_i \) is the average value of all the descendental sons of the variable value \( AL^1_i \), \( \text{MAL}_i \) is the cardinality of the set of leaf values (level), and \( n_{sl} \) is the number of variables.

1.2.2 Continious v.s. Quantized Measure

While continious measure has already been presented as the algorithm used in ESEL2, some explanation behind its choice should be presented. In Michalski’s "On the Selection of Representative Samples from Large Relational Tables for Inductive Inference" (Michalski 75), the author presents two possible algorithms, one of these is continious measure, the other is called quantized measure.

Quantized measure is computationally simpler than continious measure as it eliminates the need for a multiplication operation. It does, however, require \( p \) and \( p \) thresholds (see Michalski 75 for proper explanation) be supplied. This additional input requirement is the reason for the use of continious measure. Additionally, the Gem relational tables contain a column for specification of a weight (called cost) value for each variable. Thus, the use of quantized measure would require a number of additional numbers be input while continious requires none.

1.3 ESEL2 Algorithm

The ESEL2 algorithm can be divided in three parts: Input, which consists of reading input file and converting the data
into a form which is easily handled by the other two sections; computation of structure distances, which involves building the structures and storing the distances between values or newly computed values; and reduction of the event class, which involves choosing the events which best represent the class and then outputing them. The specific input requirements, output format, and algorithm implementation will be discussed later. This is simply designed to give a general overview of the ESEL2 operating procedure.

1.3.1 Input Initial Data

1. Locate and read the next table title and name.

2. If the table title is "DOMAINTYPES" then read all domaintypes names and associated information. Store the information in the domain storage area.

3. If the title is "NAMES" then read the values and names. Store them in a linked list hash table set up for each domain in its storage area.

4. If the title is "VARIABLES" then read all the variable names and associated information. Store the information in the variable storage area. If a domaintype name is specified, copy the information from the domain storage to the variable storage area.

5. If the title is "STRUCTURE" then read the names and subnames and convert to their corresponding values or simply read their values and subvalues. Store the information on a temporary file for later computation.

6. If the title is "EVENTS" then read the parameter line (if it exists), variable line, and variable names and/or values. Convert the names to corresponding numbers and store the information on a temporary file for later computation. Find the next table. If it has the same name as the last Events table, repeat section 6 except storing the information on a different file (this means the table has been continued). If the table is an Events table but not a continued one, then goto the beginning of section 6. If the table is not
an events table, then if the input file is empty, terminate the procedure. Otherwise continue with 7 inorder to report an error (no tables should come after Events tables).

7. If the table title was not recognized and the input file is not empty, then skip the table. This is done to allow a user to run Gem data on ESEL2. Gem recognizes more tables than ESEL2, thus ESEL2 must skip a table it does not recognize. Additionally, a status report is written to the output file report to inform the user of the skip. This is done incase of a table title misspelling.

8. Execute 1 until the input file is empty. If it is, return all domain storage area to the heap in order to give mom memory to the next procedures.

1.3.2 Computation of Structure Distances

1. Until the file containing the structures is empty, continue executing this procedure.

2. Read the structure from the file and build the structure.

3. If the structure built is for a structured-nominal variable, then set up the distance storage area for this variable and initialize each of the distances to a high, unobtainable distance ($\infty$). Structured-nominal variables require that the distance between each node (value) is stored. The present node in 3.1 will be the zero node.

3.1 The present node is considered visited. Each node which is related (adjacent) to the present node shall be called to execute 3.2 if it is not presently visited.

3.2 If the distance between the present and related node is not one, set it to one. Now check each distance between other nodes and the present node. If it exists (i.e. distance $\infty$) then the distance between the "other" nodes and the related node shall be set to the distance between the "other" nodes and the present node plus one if the former is greater than the latter. If atleast one change is made in this section of code, then the section 3.1 is called recursively with the related becoming the present. If no changes are made, then this portion of the code terminates (returning control to previous 3.1).

3.3 Upon completion of 3.1 control is transfered here. The storage area is searched for the largest distance. Each distance is then divided by the largest distance.
4. If the variable is a structured-linear type, then set up the alternative distance storage area (an array of alternative variable values used for distance determination). Find root nodes and use 4.1 to compute distances.

4.1 For each son of the node, use 4.2 to determine the value. Sum all the son values and divide by the number of sons. This gives the average son value. It is the value of the node.

4.2 If the value has previously been determined, return that value to 4.1. If the node (that is the son of the node from 4.1) is a leaf, return the value of the leaf. Otherwise call 4.1 (recursive call) with the new node as the previous 4.1's node's son.

4.3 When 4.1 terminates, control is transferred here. Assign the new "distance" variable values to the created variable storage area.

The algorithms outlined above (#3 & #4) are quite complex due to recursion. The main recursive sections occur between subsections 1 & 2 in both #3 & #4. The recursion in algorithm #3 is especially complex due to the iterative loop in subsection 3.2. These algorithms are not, however, important for the average user of ESEL2. They are printed here only for those interested in examining the source code of ESEL2.

1.3.3 Reduction of Events

1. Determine the distance to zero for all events, sorting them in increasing order. The distance to zero is defined as \( d(e_i, e_0) \) for the \( i \)th event. The event \( e_0 \) is an imaginary event with each variable value set to zero. This procedure is very complex due to the high number of events which ESEL2 can handle. Therefore, the events are sorted on three temporary files. Additionally, the procedure determines the maximum and minimum zero distance events.

2. The actual distance between the maximum and minimum zero distance events is determined and divided by the number of events in the class less one. This is the average distance between events in this event class, as long as the maximum and minimum zero distance events are the maximum and minimum events.
3. The input parameters are used to compute the number of intervals (partitions), the number of near neighbors, the maximum number of reduced events, and the near neighbor distance.

4. The event class is divided into partitions based on the number of intervals computed in part 3.

5. A partition is read from the temporary storage file into main memory where the distance between each event in the partition is computed and stored.

5.1 The two events with the largest distance between them are chosen, then tested for enough near neighbors. If neither event is qualified, the two events are marked as error events (as are any near neighbors) and this step is repeated until at least one valid representative is found, or no events remain to be tested (i.e. all events are either error events or selected).

5.2 If at least one representative is found, the following selection process is repeated until enough events have been selected from this partition, or no events remain to be tested:

\[
\frac{s-1}{\pi} d(e_s, e_j) = \max_{e \in E_i} \frac{s-1}{\pi} d(e, e_j);
\]

where \(e_s\) is the potential number \(s\) representative event, and \(e_j\), \(1 \leq j \leq s-1\), are the previously selected representative events and \(E_i\) is the \(i\)th partition event set. If \(e_s\) has enough near neighbors, it is officially added to the selected subset. If not, it along with all its near neighbors are marked as error events. The reason that an error event's near neighbors are marked as errors is that none of the events in that group of near neighbors will qualify once one fails.

6. If the number of near neighbors required is at least one, a system of selecting alternative representative events is used. This involves selecting one additional alternative per interval, until the number of alternatives plus reduced events equals the total requested number of reduced events. If this is never reached, due to near neighbor restrictions, the alternatives are used along with the chosen events. Once this value is reached, the alternatives, starting with the last chosen, are replaced by the representative events, until the reduction process ends.

7. The input file is echoed until the event class is found, the parameter line ignored, and the appropriate reduced event line numbers are output. All non-chosen event line numbers are ignored. A blank line is echoed when complete.

8. Step 1 is repeated until no more event classes exist.
2. ESEL2 Algorithm Implementation

Although the ESEL2 algorithm has been presented, the effects of the specific implementation on input, computation, output and run time can be very interesting and important.

2.1 Input Initial Data Procedure

This algorithm was designed by the author to read the input file, translate the information into data which can be easily handled by the other procedures (i.e. convert character keywords to integer keywords) and then store that information on system files for later retrieval. In order to do this, ESEL2 uses three temporary integer files: Strfile, which stores the structure information for structured variables; efile, which stores the event information for each class of events; ofefile, which stores any event tables which are continuations of previously read event tables. The use of these files, as well as other data structures allows the user to exercise a great amount of freedom when entering information.

2.1.1 Input Relational Table Order

One freedom allowed by the ESEL2 input algorithm is the order in which tables are read from the input file. While Gem specifies a number of rules of order, ESEL2 has only two rules:

1. All names within a table must be defined previously.
2. Only Event tables may follow Event tables.

The effects of these rules will be discussed next.
The effect of rule one is clear. Tables can be entered in any order as long as the names within the table have been defined. Thus, a variable structure table cannot come before the variable table which defines the variable the structure is associated with. However, the Names tables, which are associated with a domaintype, may be placed any place after the domaintype table as long as it is before any structure or event table which uses its names and corresponding values.

The effect of rule two is clear, however, the reasons for it are probably not. In different terms, rule two indicates that any table, which is not an Event table, which follows an Event table will not be recognized, no matter what type. This is necessary because of the input procedure and the continued table option. Additionally, all ESEL2 tables can and should be defined prior to the event tables. Thus, this rule should have no effect on ESEL2 users. It also allows users of Gem to leave their Inhypo directly prior to the corresponding Event tables, since ESEL2 does not recognize Inhypo tables anyways.

2.1.2 Undefined Names and Continued Tables

If, however, a table is read which contains an undefined name, a report of the error will be printed in the Report file. Additionally, the user will be notified of the error on-line, and the program will halt execution.

The final freedom offered by ESEL2 input is the continued table. This feature allows the tables Domaintypes, Variables, and Events to be continued. While the events table can have only
one continuation, variable and domaintype tables may have any number of continuations. The parameter line in the Events table must only be specified in the first occurrence of the table. Further requirements and specifications may be found in Gem documentation.

2.2 Computation of Structure Distances Procedure

As explained earlier, the two main algorithms of this procedure are recursive, one with an iterative loop within the recursive structure. Both algorithms were designed by the author, although the actual formulas were designed by R. S. Michalski and R. Stepp, as explained in section 1.2.1.

The first algorithm, for computation of distance between structured-nominal variable values, can accept any undirected graph as input. The structured-linear procedure must, due to its purpose, be a tree structure and thus contain no circuits. If a structured-linear variable was to specify a graph with circuits, the required leaves would not all exist and the distances determined would be based on the order of node computation, and have little to do with the actual structure.

One requirement of both graphs is that the values (input, not computed) of their nodes must contain the integers zero through the number of nodes less one. In other words, if a graph contained 25 nodes, they must be called (directly or through the use of predefined names) 0, 1, 2, ..., 23, 24.

Finally, the maximum number of relations (pointers to adjacent nodes) a node may have is the maximum domain size of a variable (presently 59). This allows graphs of any
structure to be entered. This includes multigraphs. Multigraphs will, however, cause unnecessary computation since the branch-value from one node to another is always one. Additionally, structures which have nodes which point to themselves will be accepted, however cause erroneous results in the computational phase of this procedure.

2.3 Reduction of Events Procedure

This procedure is clearly the most complex, containing algorithms for sorting event classes, determining partitions, determining near neighbor requirements, selecting the reduced event set as well as determining each chosen event's validity, and finally reporting the reduction to the user. This section of the report will describe these procedures more clearly as well as presenting necessary implications of the implementations.

2.3.1 Sorting of Events in Event Class

This procedure is very involved. It must read the events from the files efile and ofefile, compute their distance to zero, then sort them in ascending order. This work is added to by the requirement that ESEL2 handle 30,000 events. Obviously this can not all be done in main memory. Therefore, this procedure uses three temporary event record files to assist in the sorting.

The procedure for the sorting of the events using the three files (called ecfone, ecftwo, and ecfthr) was written by the author with the example of a similar algorithm written by E. Riengold. It uses a concept called polyphase merge, and
the reader may find additional information in literature con­
erning external sorting, sorting from tape units or sequencial
files, and quick-insertion sorts. The algorithm used by this
procedure is not as efficient as others available in literature,
however, its original design has given it many of the virtues
of the other routines without the overhead required by them.

The algorithm yields so-called runs of sorted events
on two of the temporary files named earlier (ecfone and ecftwo).
The runs are distributed on the files in an attempt to achieve
a perfect initial distribution for a polyphase merge. These
distributions are listed in the table below:

<table>
<thead>
<tr>
<th>number of file merges required</th>
<th>file run count(1)</th>
<th>file run count(2)</th>
<th>file run count(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>k+1</td>
<td>#runs1_k +</td>
<td>#runs1_k ;</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>#runs2_k !</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where "#runs1_k" stands for the number of runs in file (1)
when k merges were required. Below is an example of a poly-
phase merge from a perfect distribution:

<table>
<thead>
<tr>
<th>Merge #</th>
<th>File(1)</th>
<th>File(2)</th>
<th>File(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13/01</td>
<td>08/01</td>
<td>00/00</td>
</tr>
<tr>
<td>1</td>
<td>05/01</td>
<td>00/00</td>
<td>08/02</td>
</tr>
<tr>
<td>2</td>
<td>00/00</td>
<td>05/03</td>
<td>03/02</td>
</tr>
<tr>
<td>3</td>
<td>03/05</td>
<td>02/03</td>
<td>00/00</td>
</tr>
<tr>
<td>4</td>
<td>01/05</td>
<td>00/00</td>
<td>02/08</td>
</tr>
<tr>
<td>5</td>
<td>00/00</td>
<td>01/13</td>
<td>01/08</td>
</tr>
<tr>
<td>6</td>
<td>01/21</td>
<td>00/00</td>
<td>00/00</td>
</tr>
</tbody>
</table>

where "01/21" stands for 1 run from 21 original runs. This merge
also prevents more than one file from being empty which would
require additional code and time to copy some of the runs onto one of the empty files in order to begin merging files again. Additionally, if the procedure does not yield a perfect number of initial runs, the files are padded with dummy runs until a perfect distribution is achieved. These dummy runs are not actually processed but are "place holders."

In order to produce these runs, a list of events is maintained. The length of this list is discussed in section 2.4 of this document. The list is sorted by a modified version of the quick sort and insertion sort, commonly referred to as the quick-insertion sort.

The procedure then dumps 47% of the list onto the appropriate file. The vacated 47% is filled and the list examined. Those events with a distance to zero less than the last event stored in the present run (on the file) are placed in the front of the list. These become the beginning of the next run. The events in the present run portion of the list are then sorted and again 47% of the list length is dumped. This is continued until the present run portion of the list becomes less than 55% of the list. At this point, the remaining present run events are dumped and the next run is begun with the events in the front of the list.

The procedure above is repeated until no more events exist. Then, any remaining events are dumped into the final run. The files are padded (if appropriate) and the polyphase merge is begun.

The reason for the 47% dumps is to increase the run
lengths in the files. If the list was simply sorted and dumped, the run length would simply be the list length. However, using partial dumps of 47%, the run lengths in the average case increases to 135% of the list length. In the worst case the run length is again only the list length, but in the best case only one run is formed.

The process of using partial dumps does require some extra sort time. However, this sort time is small as the second, third, etc., sorts are already partially sorted. Additionally, the time saved by reducing the number of merges more than compensates for the time lost sorting as input and output to external files always consumes a large amount of time.

2.3.2 Computations With Parameters

Each of the four parameters input by ESEL2 are modified, or at least checked, before the values they determine are used by the reduction process: The parameter "SELECTION" must be less than 300, greater than zero, and at least 1% of the total event class; the parameter "PARTITION" must be less than or equal to 100, greater than zero, and at least 1/(percent of events to be selected); the parameter "INACCURACY" must be less than 100, greater than or equal to zero, and indicate the approximate percent of an event class which may be errors or non-uniform; and the parameter "DISTANCE" must be greater than zero or less than 1000 in order to modify the near neighbor distance, or zero which indicates that the near neighbor distance is to be the average distance between events within a partition.
While some of the parameter restrictions are obvious, others are not. However, the best way to clear any confusion might be to simply explain the formulas parameters are used in. This will also enable the user to make better use of these parameters.

The parameter "SELECTION" is simply the number of events which may be collected in the reduced set. Since the maximum partition size is 100, the minimum number of events which may be selected is one per 100, or 1%. However, if the user specifies a partition size of 10 events, but requests a selection number less than 10% of the event class, the number of events to be selected per partition will be less than one. In this case, the partition size is modified so that the number of partitions equals the number of events which will be selected, as specified by the selection parameter.

The parameter "PARTITION" specifies the maximum number of events which will be considered in an interval (partition); it is, in other words, the partition size parameter. This parameter does not directly determine the partition size. It determines the number of partitions or intervals which will be examined. As noted above, this number must be less than or equal to the number of events to be selected, and is adjusted to meet this requirement regardless of the parameter. This means that if a user specifies that 1% of the event class is to be selected, there will be partition sizes of 100 events, and thus, the number of partitions will equal 1% of the events in the event class. If a run of ESEL2 leaves an event class
with a number of intervals equal to zero (not allowed during input), the number of events in the event class was greater than 30000. Since this would cause the minimum of the selection size to be greater than the limit of 300 events, the number of intervals is set to zero purposely, so that no events are chosen from that event class. Additionally this allows the program to continue executing properly so that other event class reductions may be continued.

The parameter "INACCURACY" is a user's estimate of the percent of the data which is non-uniform or simply errors. This parameter determines the number of near neighbors a potential selected event must have to be considered a valid representative. The relationship between the number of near neighbors and "INACCURACY" is:

\[
\text{Near Neighbors} = \frac{\text{(# Events in Class)} \times \text{("INACCURACY"/100)}}{\text{(# Selected Events)}}
\]

Thus, the number of near neighbors equals the number of class events per the maximum number of events to be reduced, times the percent of the event class which is non-uniform. If, for example, an event class contained 100 events and 10 of these were going to be selected as representatives, the following "INACCURACY" parameter values would require the indicated number of near neighbors:

<table>
<thead>
<tr>
<th>&quot;INACCURACY&quot;</th>
<th>Near Neighbors/Event</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>22.222...</td>
</tr>
<tr>
<td>60</td>
<td>6</td>
<td>44.444...</td>
</tr>
<tr>
<td>80</td>
<td>8</td>
<td>66.666...</td>
</tr>
<tr>
<td>90</td>
<td>9</td>
<td>88.888...</td>
</tr>
<tr>
<td>99</td>
<td>10</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>111.111...</td>
</tr>
</tbody>
</table>
The percent column represents the percent of neighbors which must be neighbors, according to the formula. The reason that, at a parameter value of 90, the percent of near neighbors ≈ 100 is because 90% of the event class, including the events to be chosen, are non-uniform. Thus, to be sure an event is uniform, it must have 9 near neighbors. In other words, 9 near neighbors indicates that 10 events are close together and thus one of those may be chosen as a representative event.

Finally, the parameter "DISTANCE" is the percent of the predefined near neighbor distance which this event class will require to indicate an event is a near neighbor. This predefined distance is the average distance between events, as computed according to the formula outlined in section 1.3.3 part 2. If, however, the parameter is zero, the near neighbor distance used is the average distance between events in the partition. The difference between these two measures are most important with nominal variables.

The average distance between events is computed by taking the maximum and minimum distance to zero events, computing their distance, and dividing it by the number of events in the class less one. The average distance between events in the partition is computed by summing the distances and dividing by the number of events in the partition less one.

The reason this difference is important with nominal variables is that the maximum and minimum distance to zero events are usually not the maximum and minimum events in the
2.3.3 Partitioning

The partitioning of the event class is accomplished on an incremental basis. While the number of partitions is set (by the parameters), the number of events in the partition and the number of events to be selected from the partition are modified on a continuous basis. The formulas that determine these values for each partition are:

\[
\text{Partition size} = \frac{\text{Number of events yet to be partitioned}}{\text{(total number of partitions - partition number + 1)}};
\]

\[
\text{Partition selection} = \frac{\text{Number of events yet to be selected}}{\text{(total number of partitions - partition number + 1)}};
\]

Now, suppose a user requests a partition size of 10 events, and a selection of 5 events from an event class of 34 events. The partitioning is as follows: Partition one, nine class events and a selection of one event; partition two, eight class events and a selection of one event; partition three, nine class events and a selection of two events; partition four, eight class events and a selection of one event. If the near neighbor requirements
prevent any (or the proper number) of events from being selected in an interval, the continuous modification of the partition selection number allows the events to be selected in later intervals. Additionally, the continuous modifications insures approximately equal partition and selection sizes, a goal during OR reduction.

2.3.4 Reduction, Near Neighbors, and Alternates

The actual reduction process begins by reading the partition events off the temporary file and computing the distances between them. These distances are computed and stored because of the need to continually determine the distances between events for neighbor determination and representative event selection.

A status array, containing the present status of all events in the partition is maintained for the purpose of determining near neighbors. The status values are initialized to "available" but can change to "delegate", "near neighbor", or "error event." Only events which are "available" or "near neighbors" may be chosen as prospective delegates, or representatives. Additionally, if an event of the type "near neighbor" is tested for inclusion into the representative subset, the near neighbor requirement is increased by one, as a new near neighbor must be found first to take its place.

Once the algorithm in section 1.3.3 part 5 chooses an "available" or "near neighbor" event as a prospective representative, the recursive near neighbor determination algorithm is called. This algorithm was written by the
author to determine if an event had the required number of near neighbors. This is done by finding all available events which are within the near neighbor distance of one another, or other events. Once enough available nodes are found, the procedure is terminated. If enough near neighbors are not found, all events in that group are marked as "error events." This is done so that events in that group are not tested as none would have enough near neighbors to qualify as delegates. Events with "delegate" status are also marked as errors. This does not matter, however, because the number of the event is stored in the reduction array already.

Chains of islands form a good analogy with the status array and near neighbor algorithm. If an island is within one day's travel (by row boat) of another, these are considered near islands. If an island is chosen by a tribe as a place to live, it is considered their home. Finally, island groups which do not have enough near islands, as determined by the chief, are called evil islands, and as such can not be the home of the tribe.

However, if the requirement for near islands, or near neighbors, and/or the near neighbor distance are too restrictive the selection process has two devices which insure that the appropriate number of events are selected. The first is the continually modified partition selection number, discussed in section 2.3.3, and the other is alternate events, which are chosen when ever the near neighbor requirement is greater than zero.
Basically, the reduction process computes one additional representative per interval, until the representative subset is filled. Then any additional representatives which are chosen from new partitions replace the alternates.

The reason this is done is to help insure that the entire number of requested representatives are chosen. A high number of near neighbors or, more importantly, a very small near neighbor distance could prevent representatives from begin chosen. If the following partitions can not find enough representatives, the alternates are used. This situation could be quite common, with an event class of a few nominal variables mixed with a majority of linear variables, if the average distance between events in the entire class is used as the near neighbor distance, as opposed to the average within a partition (see 2.3.3).

2.3.5 Output of Results

The first step in this procedure is to sort the reduction subset according to input line numbers. The input file is then echoed until the event class is found. Next, the parameter line (if it exists) is ignored and the variable line echoed. Then, line by line, the input is read and echoed only if it corresponds to the next line number in the reduced subset. Once the last line is input, a blank line is echoed to indicate that the reduced table is completed.

If a table is continued, the above procedure is repeated for the next table in the input. Additionally, if sequential line numbers were used by the class, these are modified so that they remain sequential within the reduced event class.
2.4 Computing System Memory Remaining

This procedure is one that has not been mentioned previously. It is used to determine the amount of system memory which is available for sorting the events in increasing order of distance to zero. This is done to allow users to enter any number of structured-nominal or linear variables, and at the same time be able to sort the data.

Structured-nominal variables require distance arrays which give the distance between any two points. Structured-linear variables require an additional array of variable values to allow distance computation between points (values). This memory comes at the expense of memory required to sort the events; One structured-nominal variable with values 0-58 requires 1711 words of memory for storage.

Therefore, in order to prevent a violation of workspace specifications, the above named routine counts the amount of memory consumed by each structured variable: 59 words for structured-linear and up to 1711 words for structured-nominal, which has its memory allocated in 59 word units. It then uses a constant specified in the program which indicates the maximum number of 59 word storage units the specified RFL will allow, (the set values are 415 for an RFL under 120000B words) in order to determine the length of the sorting list which may be used. Should this number drop below 100 (i.e. 100 events in the list) a warning is printed. Since many users are allowed more memory, the source code contains instructions for increasing the max value.
3. **KPK-BTM ESEL/GEM Experiment**

In order to test the usefulness of ESEL2, a large data set was obtained which described the chess end game in which White has a king and pawn, Black has only a king, and it is Black's move. This game is referred to as King-Pawn-King, Black-To-Move. The purpose of the experiment was to determine the total run times of ESEL2 plus Gem on the data set, then compare this amount to the time required to run the data set on Gem alone.

3.1 **KPK-BTM Data**

The data set used for this experiment was invented by Tim Niblett and Alen Shapiro at the Machine Intelligence Research Unit of the University of Edinburgh. These researchers determined that a set of 31 features (variables) could be combined into 2411 feature vectors (events) in order to describe all of the 83,622 legal KPK-BTM positions. Additionally, these feature vectors (events) may be divided into two sets of board configurations, Drawn and Lost. These two Classes describe the game outcome with respect to the Black player.

The features or variables are nominal and take on the values true (1) or false (0). In the total event set of 2411 events, 1366 represent Lost class events and 1045 represent Drawn class events. Further information on KPK-BTM and Niblett & Shapiro's data set may be found in "A Comparative Study of Two Inductive Learning Systems," written by Paul O'Rorke at the University of Illinois Computer Science Department.
Included in his study is a list of the 31 features and their meanings (see appendix A).

3.2 ESEL2/Gem Experimental KPK-BTM Data

The data used for this experiment was not the entire set created by Niblett & Shapiro. It included 1901 events which were divided into 698 Drawn class events and 1203 Lost class events. The data was entered in a form compatible to AQ11 by Jeff Jackson and termed "KPKBTM CHESS DB FROM SCRATCH FOR 2CND CLASS." The author modified the data into a form acceptable to Gem, and thus ESEL2. It is presently stored in a file called "KPKESEL." The reason the entire data set was not used was that it was not readily available in AQ format. The reduced set will, however, affect the experiment little.

3.3 ESEL2 Runs

A series of runs using the KPKESEL data was made on ESEL2. The first set of runs used a partition size of 10 events to select 70 Drawn class events (slightly more than 10%) and 120 Lost class events (slightly less than 10%). The inaccuracy parameter was varied between 0 and 20. The second set of runs selected the same number of events and varied the inaccuracy as the last. However, the second set of runs requested a partition size of 100 events. The final set of runs repeated the first two sets with the exception that the distance parameter was set to zero.
The above table summarizes the ESEL2 runs on the data set. The run times were not included, as some of the runs did not produce any output (runs 1, 2, 4, and 5). This was due to the nominal variable effect on average distance between class events (see section 2.3.2). The other runs could be combined into two groups based on their output and run times which were identical (the run times weren't exactly identical, but very close). The two groups are the 100 event partitions and 10 event partitions; runs 6, 10, 11 and 12 and runs 3, 7, 8, and 9 respectively.

The run times for the 100 partition size class were approximately 148 CPSeconds. The run times for the 10 partition size class of runs was approximately 19 CPSeconds. The deviations within the runs was not greater than .9 CPSeconds. All runs used exactly the same amount of memory, 1123058 words. The reason for this is that the major use of memory was sorting the event lists, which required the same amount of memory each time. Additionally, had the list length of the sorts been increased, the run times could have been reduced slightly.
3.4 Gem Runs

Three separate runs using different data sets were made with Gem, using all of Gem's default parameters. The three runs used the original data set of 1901 events, the 10 partition data set with 190 events and the 100 partition data set with the same number of events. The results of these runs were quite impressive.

The 1901 event data set run on Gem took 282 CPSeconds and used 116472B words of central memory. The 10 partition data set run consumed only 32 CPSeconds and used only 51441B words of C1. Similarly, the 100 partition data set took 51441B words of C1, however, it required a slightly longer run time which totaled 44 CPSeconds.

These results are combined with the ESEL2 results and listed in the table below:

<table>
<thead>
<tr>
<th>Run number</th>
<th>DATA #events</th>
<th>ESEL2 Run time</th>
<th>#events</th>
<th>Gem Run time</th>
<th>Covers</th>
<th>TOTAL Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1901</td>
<td>---</td>
<td>1901</td>
<td>282</td>
<td>42</td>
<td>282 CPS</td>
</tr>
<tr>
<td>2</td>
<td>1901</td>
<td>19</td>
<td>190</td>
<td>32</td>
<td>15</td>
<td>51 CPS</td>
</tr>
<tr>
<td>3</td>
<td>1901</td>
<td>148</td>
<td>190</td>
<td>44</td>
<td>19</td>
<td>192 CPS</td>
</tr>
</tbody>
</table>

The results show a significant improvement in the amount of run time required to process the events. The covers generated, however, were not quite as good when the events were processed first by ESEL2; however this is to be expected. Additionally, the covers generated by the 100 event partition reduced set were significantly better than the 10 event partition covers, but at a cost nearly four times as great. Therefore, the results seem to indicate that ESEL2 can significantly cut Gem computational time by choosing the most representative events.
4. ESEL2 User Guide

Input and output to ESEL2 is done through system files. These files are named EVENTS, ESELOUT, REPORT or are simply the files which are specified in the execution statement in the same order as above.

4.1 ESEL2 Input--EVENTS

ESEL2 input is entered in the system file EVENTS, or the file first specified in the ESEL2 execute statement. The format of the file EVENTS is identical to the input file to Gem with only minor changes. Therefore, this guide will only discuss their differences. If a user does not know the format of Gem input relational tables, the Gem user guide should be examined.

ESEL2 recognizes 5 table types. These are: domaintypes, used for the definition of domaintype names; names, used for the definition of domain names; variables, used for the description of variables; structure, used for the definition of a structured variables structure; events, used to hold the class events and define their variable values. Gem does not presently recognize the structure table as structured variables are not presently available on Gem. Additionally, any tables encountered by ESEL2 which are not of the above five types are simply echoed. A report of the skip is included in the output file REPORT (see 4.3 for additional information on REPORT). This feature allows a Gem data set to be entered into an
ESEL2 program with little or no changes, and then run immediately on Gem after output by ESEL2. The report of the skip is made incase a table title was misspelled, and therefore skipped by accident.

4.1.1 **ESEL2 Requirements for Relational Tables**

The list presented below is a list of additional requirements or modifications ESEL2 requires for Gem relational tables. Again, the specific format can be found in the Gem user's guide.

1. **ESEL2 does not recognize exclamation-points as blanks.** This is done in Gem to accomodate tables output by QUIN.

2. **ESEL2 does not recognize placeholders which represent default values in table columns.** If a column exists, ESEL2 expects the proper parameter be entered, even if it is the default value.

3. **ESEL2 does not have default variable names.**

4. **Structured variables have a fixed structure and the definition of it, in the structure table, must follow the definition of the variable name in the variable table.**

5. **Sequential event line numbers may be used by the user.** These will, however, be modified when the reduced set is output onto the ESELOUT file.

6. **Variable values specified in the Events table must be singular; no multiple values are allowed.**

7. **The ESEL2 parameters (see sections 2.3.2 and 4.1.2) are input on a single line, which is initiated with a star ("*").** The line must come after the event class table title line.
(before the variable order line). Each parameter has a default value, therefore the line, or parts thereof, need not be entered.

Examples of various tables with parameter lines follows:

```
classone-EVENTS
* PARTITION=100 INACCURACY=30 SELECTION=190 DISTANCE=0
  # variable1 variable2 variable3 ...
  1 value  value  value  ...
  2 value  value  value  ...
  3    :    :    :
  ...

classtwo-EVENTS
  # variable1 variable2 variable3 ...
  1 value  value  value  ...
  2 value  value  value  ...
  3    :    :    :
  ...

classthree-EVENTS
* INACCURACY=25 SELECTION=90
  variable1 variable2 variable3 ...
  value  value  value  ...
  value  value  value  ...
  :    :    :
  ...

classthree-EVENTS
  variable(n) variable(n+1) ...
  value  value  ...
  value  value  ...
  :    :    :
  ...

classfour-EVENTS
* DISTANCE=50 INACCURACY=40 SELECTION=290
  variable1 variable2 variable3 ...
  value  value  value  ...
  value  value  value  ...
  :    :    :
  ...
```

In the above four class event tables (table for classthree is continued), only the classtwo event table uses all default values (notice there is no star since there is no parameter line). The other tables vary between setting all four parameters to
setting two (any number can be set). Also, notice that in the continued table, the parameter line is in the first occurrence of the table, and is not repeated with the second. This is an additional restriction.

8. In addition to the variable types NOM and LIN, ESEL2 recognizes the types SNA and SLN, representing structured-nominal and structured-linear respectively.

9. The weight term in the OR method is simply the Gem term cost.

4.1.2 ESEL2 Parameters

The placement of the ESEL2 parameter line has already been discussed, however, the specific parameters have not. There are four parameters which ESEL2 recognizes. They are PARTITION, SELECTION, INACCURACY, and DISTANCE. The computations which are done with these parameters are discussed in section 2.3.2, therefore a brief description of each will only be given here.

The parameter PARTITION indicates the maximum partition size, in terms of the number of events to be considered in that partition. The number of intervals, which is actually determined by the parameter, must not fall below the number of events to be selected. If it does, the number of intervals is set at that value, regardless of the parameter value.

The parameter SELECTION indicates the maximum number of events which are to be selected from the event class, and termed representatives. The maximum number of selected events is 300 and the amount may never fall below 1/3 of the total.
number of events in the event class. This adds an additional restriction to the event class table. No more than 30,000 events may be input. If more are input, the number of intervals is set to zero, and that table does not have any representatives chosen.

The parameter INACCURACY indicates the percent of an event class which a user believes is non-uniform, or simply error events. This parameter determines the number of near neighbors which the selection process will require for an event to be chosen as a representative. If, for example, a user entered an inaccuracy of 20 (20%), ESEL2 would require that over 20% of an event's neighbors be near neighbors.

The parameter DISTANCE indicates the distance required for an event to be considered a near neighbor (the maximum distance, that is). The parameter represents the percent of a pre-determined distance which will be required. That distance is the average distance between events in the event class. Thus, if 200 is entered (200%), the near neighbor distance is twice the average distance between events in the event class. If 50 is entered, it is the average distance between events in the event class. This pre-determined value is very inaccurate for nominal variables. Therefore, if a large percentage of event variables are nominal, the user may enter a zero for the distance parameter. This will cause the average distance between events in the partition to be used, at the expense of a uniform requirement.
This average, that of events within the partition, is totally accurate, however, not as desirable as a constant requirement which the other value is. The problem with the other value is that it can be much less than the actual average, thus allow no events to be chosen. For further discussion on this, see section 2.3.2. Also, classone Events table enters a zero value in section 4.1.1.

4.1.3 ESEL2 Parameter Restrictions

Each of the parameters mentioned above has high and low limits as well as additional restrictions. When ever any of these restrictions are violated, ESEL2 automatically adjusts them so that they satisfy the violated requirement. Below is a table of the minimums, maximums, and other restrictions which apply to the parameters:

<table>
<thead>
<tr>
<th>Name of Partition</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
<th>Additional Restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTITION</td>
<td>1</td>
<td>100</td>
<td>Atleast one event must be selected per partition</td>
</tr>
<tr>
<td>SELECTION</td>
<td>1</td>
<td>300</td>
<td>Atleast 1% of the event class must be selected</td>
</tr>
<tr>
<td>INACCURACY</td>
<td>0</td>
<td>99</td>
<td>If zero is entered, no near neighbors are required</td>
</tr>
<tr>
<td>DISTANCE</td>
<td>0,1</td>
<td>999</td>
<td>If a zero is entered, the partition average is used</td>
</tr>
</tbody>
</table>

4.1.4 ESEL2 Parameter Default Values, Recommended Values

The default values for the parameters PARTITION, SELECTION, INACCURACY, and DISTANCE are 10, 1, 10, and 100 respectively. These values have been selected for their effects of the "average" user's data set.
The partition size default value of 10 is chosen so that the reduction operation will run relatively fast. Since the distance between each event must be computed, the smaller the partition, the quicker the process. For example, 10 partitions of 10 events require 450 distances be evaluated and stored. One partition of 100 events requires 4950 distances be computed and stored. However, this speed comes at the expense of accuracy. The more partitions a class is divided up into, the less accurate the OR method. Additionally, partitions tend to break up groups of near neighbors. This could cause certain valid events to be marked as invalid. Therefore, based on the above information, the recommended partition sizes ranges between 10 and 100.

The selection parameter has a default value of one. This is done because unless an event class has under 100 events, this value will be modified to 1/3 of the event class. It is not recommended, however, to leave this parameter without a value, depending on the default value of 1/3, unless the user is not concerned with the actual number of events selected.

The inaccuracy parameter has a default value of 10. This indicates that a value of 10% of the neighbors an event has must be near neighbors. This value should handle most data sets fairly well, preventing the events with few or no near neighbors from being selected. However, certain data sets might require a very high number of near neighbors to prevent error events from being selected. Additionally, a better representative sample could be found with a higher
value, even if the non-uniformity of an event class is low. The recommendation for the use of the inaccuracy parameter is to run a number of identical data sets through ESEL2 while using a variety of different inaccuracy values, then compare the results.

The distance parameter has a default value of 100. This means that 100% of the pre-determined near neighbor distance will be used to determine near neighbors. If all the variables in an event class are linear, the pre-determined value will be accurate, so modification of the parameter is left to the discretion of the user. However, if a small number of the variables are nominal, then the lowering of that value is recommended. If the majority, or even a relatively high percentage of the variables are nominal, the author recommends use of the "zero" option.

4.1.5 Structure Table Format

Since Gem does not accept structured variables, a format must be specified. The format described was recommended by R. Stepp. Additional information on the restrictions of structured variables and exactly what structures are may be found in sections 1.1, 1.2.1, 1.3.2, 2.1.1, 2.2 of this document and related literature.

The first line of the table is the title line. It gives the table title, "STRUCTURE", and the table name separated by a hyphen. The table name corresponds to a variable name, which was previously defined in a variable table.
The table name (variable name) precedes the title.

The next line specifies the type of values entered. They may be either the actual values or name representatives which were previously defined in a Names table. This line also specifies the father to son relationship, which is the method the table uses to construct the structure. The two lines which follow are the only allowable lines for this table: "Name Subname" or "Value Subvalue".

There are no default lines for the table, and there must be as many lines as there are relations in the structure. Again, the two sets of columns must come in the order specified (i.e., Subname cannot precede Name). Finally, there is no restrictions on the order in which the values are entered.

Listed below are some example tables and their completed structures:

<table>
<thead>
<tr>
<th>variablename1-STRUCTURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>root</td>
</tr>
<tr>
<td>root</td>
</tr>
<tr>
<td>root</td>
</tr>
<tr>
<td>root</td>
</tr>
<tr>
<td>son1</td>
</tr>
<tr>
<td>son1</td>
</tr>
<tr>
<td>son2</td>
</tr>
<tr>
<td>son3</td>
</tr>
<tr>
<td>son4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variablename2-STRUCTURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>
The information is restated in the format of the Gem user guide:

```
<table>
<thead>
<tr>
<th>variablename-STRUCTURE</th>
<th>Optional Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE</td>
<td>SUBVALUE</td>
</tr>
</tbody>
</table>

-or-

| variablename-STRUCTURE | NAIE           | SUBNAIE        |
```

No default lines. Must be as many lines as there are relations (branches) in this variable structure. Both columns must be present (either version) and in this order. (If used, this table must come after the VARIABLE table).

1. **VALUE/NAME**: Internal value or name representative of it which represents the father in this relation. (Integer: \( =0, =\text{Levels} - 1 \)).

2. **SUBVALUE/SUBNAME**: Same as above except the son in the indicated relationship.

4.2 **ESEL2 Output-ESELOUT**

The output data from ESEL2 is stored in the second file in the execution statement, or simply the file named ESELOUT if none is specified. The file contains all the tables input. The only modifications are:

1. Only the representative events are listed in the class event table.

2. The sequential event numbers are modified if present.

3. The ESEL2 parameter line is removed.

If no events were selected from a class, the table title and variable order line are copied, followed by a blank line. Since Gem will not accept this, the user should check to see how many events were chosen by the program.
4.3 ESEL2 Output--REPORT

All information concerning reductions, skipped tables and ESEL2 termination due to errors which were detected upon input of data are written in the third file specified, or simply the file REPORT if none is specified. This file should be checked after termination to insure that the proper reduction of events took place. This file can also be quite valuable if ESEL2 is halted by an error.

4.4 ESEL2 Temporary Files

The following temporary files are used by ESEL2: EFILE, OFEFILE, STRFILE, ECFONE, ECFTWO, ECFTHR. The first three files are integer files and used for storing the event, continued event and structure tables in their raw (integer) form, respectively. The last three are special record type files which are used to sort the events prior to partitioning. In no cases should the user specify these files, as they must be specially created if they are specified. They are mentioned here only for the purpose of informing the user that they will be used during execution, and therefore no local files should be created by the user with the same names.
This report describes a method and a program, ESEL2, for selecting the most representative events from a large set of events characterizing one or more different decision classes. A typical use of ESEL2 is to select most representative training examples for an inductive learning program, such as AQ11 or GEM. The basic idea for the selection method, called OR ("outstanding representatives"), is to choose from each event class those events that are most different from each other but, at the same time, not isolated.

17. Key Words and Document Analysis. 17a. Descriptors

- training examples selection
- inductive learning
- machine learning
- artificial intelligence
- pattern recognition