LAH 389

Cytometry 16:357–363 (1994)

TECHNICAL NOTE

Pattern Sorting: A Computer-Controlled Multidimensional Sorting Method Using K-D Trees[1–3]

M. Bigos,[4] D.R. Parks, W.A. Moore, L.A. Herzenberg, and L.A. Herzenberg
Department of Genetics, Stanford University School of Medicine, Stanford, California

Received for publication July 30, 1993; accepted March 6, 1994

Multidimensional binary trees provide a memory efficient and general method for computing sorting decisions in real time for a flow cytometer. Their fundamental advantage over conventional lookup table sorting techniques is that sort criteria in the full N-dimensional data space which cannot be described by projections onto two-dimensional parameter planes can be effectively implemented. This becomes particularly relevant when multidimensional analysis methods such as principal components or clustering are employed. We describe a prototype implementation of this method and point out other possible implementations. © 1994 Wiley-Liss, Inc.

Key terms: Algorithms, flow cytometry, multivariate analysis, pattern recognition, real-time systems

Many investigators (2,10), including ourselves, are actively involved in developing a new class of analysis methods for determining cell populations in flow cytometric data. A feature of these methods is that the full dimensionality of the flow data is used at once to determine relevant cell populations. Two examples of such methods are cluster analysis and principal component analysis. Once populations are analyzed, further biological studies may well require the physical sorting out of such populations. However, there is no guarantee that sort regions defined on populations determined by these methods can be described adequately by intersections of regions in the one- or two-dimensional marginal distributions of the data. For example, in principal component analysis a bivariate plot of the first two principal components may involve both a rotation and translation of the original data. The conventional methodology of implementing computer-controlled cell sorting using one- and two-dimensional lookup tables ("bit mapped" sorting) can therefore fail to represent adequately a sort region defined, e.g., as a polygon on a plot of the first two principal components of a data set.

Higher dimensional lookup tables are possible, but they need prodigious amounts of memory to maintain high resolution. For example, if the flow data have 8 bit resolution (256 measurement channels per parameter), an 8 parameter lookup table will require approximately 10^{19} bytes of memory, a number not technologically feasible at this time. Using lower resolution flow data can reduce the amount of memory needed. For example, 8 parameters of 5 bit flow data (32 measurement channels per parameter) could be sorted using a lookup table 1 gigabyte in size. However, such low resolution would probably not be acceptable for many flow applications. Thus, to sort cells at full data resolution according to generalized analysis requires a different approach.

PRINCIPLE OF PATTERN SORTING

We call our approach to generalized sorting *pattern sorting* because it can implement any pattern of sort

[1]This research was supported by NIH grants CA-42509 and LM-04836.

[2]The methods and application described herein are part of U.S. Patent 4,989,539.

[3]This work was presented at ISAC XVI Congress, Colorado Springs, CO, March 20–26, 1993.

[4]Address reprint requests to M. Bigos, Department of Genetics, Beckman Center B007, Stanford University School of Medicine, Stanford, CA 94305–5125.

decisions in N-dimensional data space, no matter how the pattern is generated. The method begins with a training set of data which is analyzed by any methodology to specify sort criteria for each cell. This is usually done in terms of sorting regions defined geometrically (gating), but any method which assigns a unique sort decision to each cell is acceptable. The classified N-dimensional training set is next split into hypercubes, each containing only one cell of the data set. This results in many small hypercubes in areas of dense data and large ones in areas of sparse data. Each hypercube is then assigned the sort decision of the cell of the training set it encloses. Finally a computer program (the sort program) is generated which determines, for each cell subsequently run through the flow cytometer, which cube it is a member of. This program then returns that cube's sort decision to the cytometer for implementing the physical separation process.

The execution speed of the sort program, as well as its data acquisition path, are the time-critical aspects of pattern sorting. For a jet-in-air sorter with a typical jet velocity of 10 m/s, and a break-off point 6 mm below the laser interrogation spot, the charging pulse for a particular cell occurs approximately 600 μs after the measurement is made. Although instrument design may impose other constraints, the sorting process itself requires that all analog and digital processing of the event data required to generate a sort decision must complete before this time expires. Since the time intervals between cells are, at best, random within an exponential distribution (5,6), we must allow for pileup of events. For example, a data transfer and sort decision calculation time of 100 μs would permit event rates approaching 10,000 per second without frequent 600 μs overruns. Both software design and computer hardware configurations affect the ability to achieve this 100 μs throughput for event processing. As described below, we achieve this on slow and obsolete computer hardware for our prototyping system, thus showing the robustness of the software approach we developed. Different hardware technologies for data transfer and microprocessor design would result in a system that is significantly faster than the one we demonstrate. However, a properly implemented bit-mapped sorting scheme, using equivalent hardware and doing the lookup tables in parallel, will always be faster than the approach described here.

Our approach relies heavily on k-d trees. They are used to decompose the training set data into the N-dimensional hypercubes as indicated above. The resulting tree is then compiled into an assembly language program which is loaded into a dedicated processor and run. As mentioned, this methodology does not, in and of itself, provide a speed increase over bit-mapped sorting. Its usefulness lies in its generalized approach to sorting classified populations.

Thus, the steps in pattern sorting are summarized as follows: 1) collect and analyze a data set to determine sort criteria; 2) construct a balanced k-d tree on this data set using the results of that analysis; 3) prune the k-d tree; 4) generate a sort program from the pruned tree; and 5) load and run the sort program. These steps are described in the next sections.

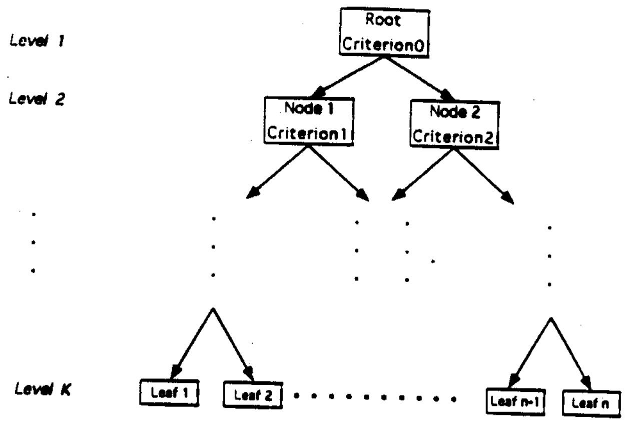


FIG. 1. Constituent parts of a k-d tree. A k-d tree consists of nodes with splitting criteria and paths connecting them. The top node of the tree is called the Root; here it is partitioned by Criterion 0. When traversing the tree, the split criteria at the current node is evaluated to determine whether the next node to visit is the left child or the right child. In this tree, the left and right children of the Root are Nodes 1 and 2, and they contain Criterion 1 and Criterion 2, respectively. The last node in a path is called a Leaf. The depth of the tree is the number of levels between the Root and the Leaves. For a balanced tree, all Leaves will be on the same level. The almost-balanced trees constructed for pattern sorting have all Leaves within two levels.

PROCESS OF PATTERN SORTING

Collect and Analyze a Data Set to Determine Sort Criteria

Pattern sorting does not require a particular method be used to determine sort criteria. What is important is that the analysis end up by classifying each event (cell) in a data set from the sample to be sorted. Typical classifications are LEFT, RIGHT, and NOSORT; however, our method is general enough to apply to instruments with more than two sort streams or for applications other than flow cytometry. This classified data set, for reference, is called the training set.

Because pattern sorting will approximate the sort regions by a composite of hypercubes, it is important to have a large enough training set to delineate these regions adequately. Figure 4 (see later) demonstrates this clearly. Where the events are dense (the bottom of the sort region), the generated hypercubes approximate the specified sort boundary closely; where they are sparse (the top of the sort region), the approximation is coarse. For most sorting applications this will not matter because the exact sort boundary tends to be arbitrary in regions of low cell event frequency. If it is absolutely necessary to exclude cells that fall in a certain region, then the final hypercube boundaries will

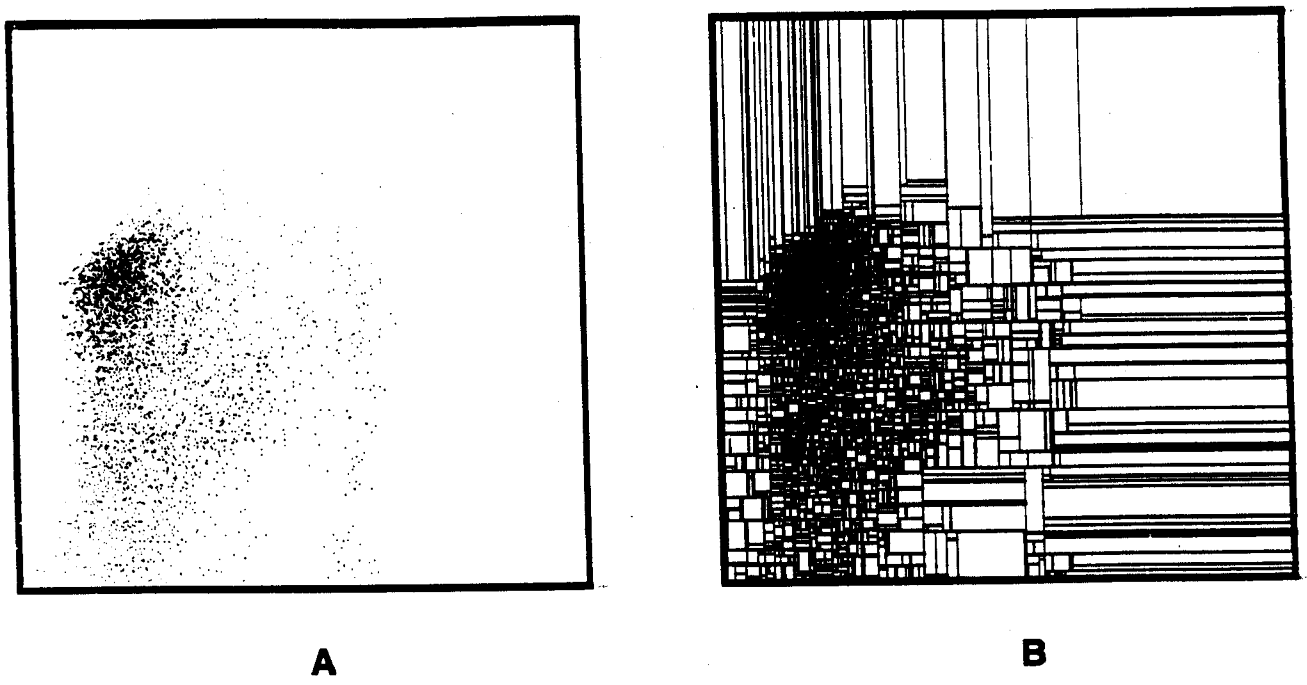


FIG. 2. Example of a k-d tree partitioning. A: A two-dimensional dot plot with 5,000 events of FACS data. B: Using the methods described in this note, this data yield a k-d tree whose 5,000 leaves are shown.

need to be adjusted, an easy extension to our process that we do not address in this note.

Moreover, the higher the dimensionality of the data space, the larger a training set needed to adequately decompose it. We have not yet developed analytic criteria for this. Anecdotal investigations indicate that while three-dimensional distributions are adequately described using 10,000 or 20,000 event training sets, 8 parameter distributions may require 100,000 events or more.

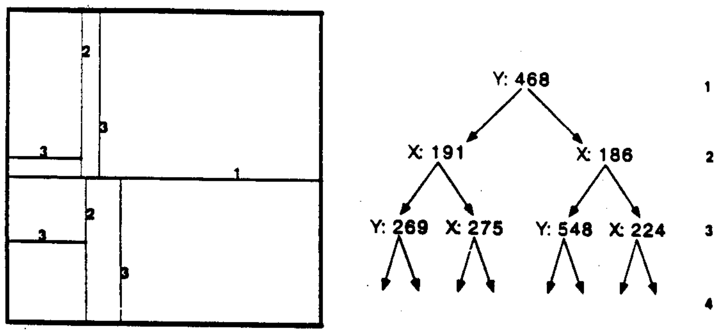
K-D Tree Generation

Since their introduction in 1975 (1), multidimensional binary search trees (abbreviated k-d trees where k is the number of dimensions) have found numerous applications in diverse fields such as databases, computational geometry, and pattern recognition (8,9). Figure 1 describes the salient parts of a k-d tree. The k-d tree decomposition of the training set produces an almost-balanced tree whose depth is $\mathrm{LOG}_2(n)$, where n is the number of points in the training set. We will demonstrate the technique on a two-dimensional data set for ease of drawing; the extension to higher dimensionality will be clear.

Consider the 5,000 event dot plot of FACS data shown in Figure 2A. The leaves of the k-d tree decomposition, as shown in Figure 2B, are represented by rectangles, each containing one event. Thus, if another cell is randomly selected from the same sample which generated Figure 2A, it has equal probability of falling in any of these rectangles.

To construct this decomposition in two dimensions, one starts with the rectangle formed by the entire data space as the root of the tree. This root rectangle contains the entire training set and is divided into two rectangles by the median of the data on the parameter which has the largest range, producing two nodes, each of which contains half the training data set, which are children of the root. In the example shown, it is only necessary to consider the X and Y parameters, but if the training set contains more than two parameters, then each will have to be scanned to find the one with the largest range. Each of these two rectangles is subsequently divided in the same manner, producing four rectangles. As before, for each rectangle, all parameters of the data contained in that rectangle are evaluated and the split occurs on the median of the parameter with the largest range. This process is repeated until each rectangle contains only one data point. These rectangles are the leaves of the tree. Figure 3 shows the first few divisions, together with the corresponding k-d tree that is internally constructed along with this division.

Now each point in the data set has an associated sort decision defined by a user analysis. In the example shown in Figure 4 this is accomplished by a polygon on the X and Y parameters. (For higher dimensional training sets this might be accomplished by an N-dimensional clustering algorithm with different clusters



```
         CMPW    R2,  #486
         BLEQ    $1
         CMPW    R1,  #186
         BLEQ    $2
         CMPW    R1,  #224
         BLEQ    $3

$1:      CMPW    R1,  #191
         BLEQ    $4
         CMPW    R1,  #275
         BLEQ    $5

$2:      CMPW    R2,  #548
         BLEQ    $6

$3:      (Other processing code)

$4:      CMPW    R2,  #269
         BLEQ    $7
              .
              .
              .
```

A B C

FIG. 3. Example of k-d tree construction. The first three steps of the construction of the k-d tree used to classify the data displayed in Figure 2 are shown here. A: The resulting rectangles from the tree fragment in B. The Y parameter has the greatest difference between largest and smallest values so the root, which corresponds to the entire two-dimensional data space in this example, is split on the median of all Y-values at channel 468 (out of 1,024), resulting in two rectangles at level 2. Each of these rectangles are split on the parameter with the greatest spread, resulting in a total of four rectangles at level 3. To obtain the full tree, the splitting occurs recursively until each rectangle has only one data event in it (see Fig. 2B). C: The prototype sort code generated from this tree fragment using VAX Macro-32. It is assumed that the jacketing procedure for this code placed the data values corresponding to the X-axis in Register 1 and those corresponding to the Y-axis in Register 2 of the VAX processor.

being sorted in different directions, or by any dimension reduction scheme, such as defining sort regions on the first two principal componentes of the training set.) The leaf containing that point is assigned the same sort decision. If two leaves which are children of the same parent have the same sort decision, they may be replaced by their parent. This is called "pruning the tree." Figure 4B shows the leaves associated with the pruned tree defined by the single polygonal gate in Figure 4A.

Note that the number of branch points in an unpruned k-d tree is approximately $\sum_{i=0}^{\log_2 n} 2^i - n$ (n as above). Each of these branch values requires the computation of a median on one parameter of a subset of the training set. For a 50,000 cell 6 parameter training set this results in about 65,000 median computations. To accomplish this efficiently we use an algorithm devised by Hoare (3), a modification of Quicksort, which finds the median of a set of K numbers in order K time. Because this algorithm also partially sorts the data set, simple bookkeeping allows us to apply this method recursively to our training set to determine all the branch points rapidly. For a 50,000 cell 6 parameter training set, a C implementation of Hoare's algorithm on a VAXStation 3100 M76 (Digital Equipment Corp., Maynard, MA) running at approximately 10 MIPS produces a tree in less than 30 s.

The time required to prune a tree will vary greatly depending on both the sort criteria selected and the details of the program code implementation. The larger the percentage of cells that have the same sort decision (and are spatial neighbors), the more the underlying tree can be pruned. The fine resolution of one cell per leaf will only be maintained for cells which have neighbors with a different sort decision. This can be seen clearly in Figure 4B, where small rectangles are required only along the lower part of the polygon defining the sort region. Efficient program implementation requires that the entire k-d tree and an auxiliary set of pointers to the leaves be kept in memory. For the 50,000 cell training set and hardware described above, pruning 45,000 leaves required less than 15 s when all the data were memory resident and several minutes if paging in and out of memory was required.

Generation of Sort Code From the K-D Tree

Sort code is generated from the k-d tree in the following manner. Each branch point in the tree translates into a COMPARE instruction. Typically one child of the node will be reached by a BRANCH instruction, and the other is the default fall-through, but the actual details of code generation will depend on the processor instruction set available. Figure 3C illustrates a sample of how code may be generated from the corresponding k-d tree. For a training set of n data points, the sort code will have approximately n compare/branches. However, since the depth of the corresponding tree is approximately $\text{LOG}_2(n)$, the number of instructions executed to reach a sort decision is small and bounded. For example, a 50,000 point training set will require a maximum of 16 compare/branch sequences to reach a

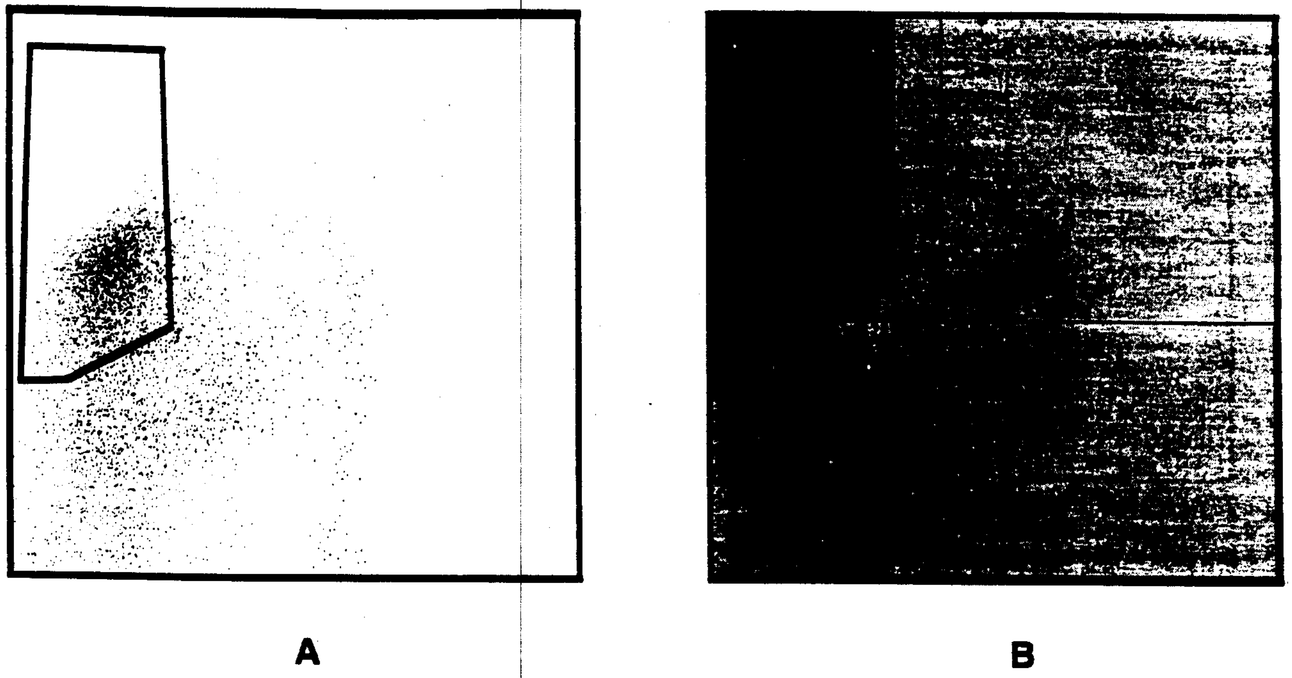


FIG. 4. Sorting using k-d trees. A: The data set from Figure 2 with a sort gate. B: The pruned k-d tree for making sort decisions using the gate in A. Note that in general the "larger" leaves of a pruned k-d tree are "closer" to the root. Thus, except for certain border regions on the gate, most sort decisions will be reached after only a few levels in the pruned tree. This figure also shows that in areas of high cell density the resultant tree faithfully approximates the specified gate, while in areas of low cell density the approximation is poor.

decision, even though the total program is about 100,000 instructions.

The reduction in tree depth from pruning is highly dependent upon the relationship between the training set and the sort regions. If the sort region encloses or excludes large numbers of training set events that are well separated from sort boundaries, then for those events the average tree depth will shrink considerably. If the maximum tree depth is needed for only a fraction of the events in the training set, then the ability of run time event pileups to overrun the sort processor will be minimized. The maximum tree depth will, in general, still occur for events that are close to the sort region border, and the frequency of such events will be high if parts of the border cut through areas of high event density in the training set. Tree size will also be affected by pruning, with the same considerations as above.

For the sample training set shown in Figures 1 and 4, pruning reduced the average tree depth from 13 to 8 while reducing the total number of leaves from 5,000 to 132. However, around the lower border of the sort region there are some areas where the full depth of the tree is required. Since each level compiles to two executable machine instructions in our prototype, pruning saves for this training set an average of about 10 μs per sort decision on a 1 MIPS sort processor. On a faster processor, the sort speed increase from pruning will probably be negligible. Note, however, that the number of assembly language instructions for implementing the tree reduces from over 10,000 to a few hundred. This decrease in size can be very beneficial if the sort processor has an instruction cache, because smaller code is more likely to fit entirely in the cache.

Finally, note that the code representing the k-d tree must be jacketed with code that polls and reads data from the cell sorter, loads the data values into registers, and returns the sort decision to the cell sorter. Using standard microprocessor buses and parallel digital I/O boards, these functions can be more of a throughput bottleneck than the sort decision calculation.

IMPLEMENTATION OF PATTERN SORTING

The prototype computer hardware for our implementation of pattern sorting is shown in Figure 5. An rtVAX 3000 and a MicroVAX processor (Digital Equipment Corp.) are configured in a Q-bus box in an arbiter/slave relationship; the details of this have been described previously (4). Three parallel digital interfaces are used for communication between the VAX processors and the cytometer electronics.

The arbiter processor, called the network server, handles data I/O, instrument control information, network connectivity, and loads programs into the slave processor. This makes the cytometer function as if it were a device on the ethernet, offering services (data

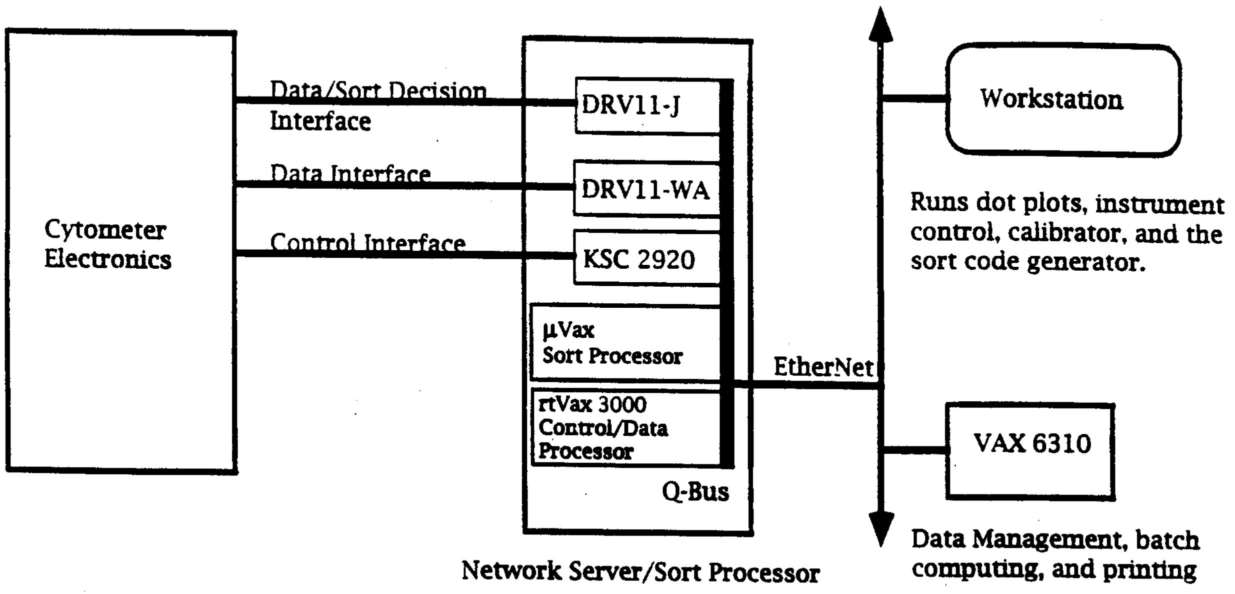

Cytometer Electronics
Data/Sort Decision Interface
Data Interface
Control Interface
DRV11-J
DRV11-WA
KSC 2920
µVax Sort Processor
rtVax 3000 Control/Data Processor
Q-Bus
Network Server/Sort Processor
EtherNet
Workstation
Runs dot plots, instrument control, calibrator, and the sort code generator.
VAX 6310
Data Management, batch computing, and printing

FIG. 5. Computer hardware for implementing pattern sorting. This figure shows the prototype computer system used to implement pattern sorting. The cytometer electronics, a custom-built eight parameter flow instrument, is connected to a dual processor computer using three parallel interfaces. The Control/Data processor, an rtVAX 3000, runs VAXELN. It reads data from the cytometer over a DRV11-WA interface (Digital Equipment Corp.) and sends it out to requesting clients over ethernet via an on-board connector. It receives cytometer control messages from clients and loads them into the cytometer electronics through a KSC 2920 CAMAC interface (Kinetics Systems Corp., Lockport, IL). Lastly, it also receives sort programs and loads them into the sort processor. The sort processor, a microVAX II, runs the current sort tree program. It reads data from the cytometer over a DRV11-J interface (Digital Equipment Corp.), processes the data to make a sort decision, and returns it over the same interface. When sorting is not desired, a default program is run which returns a "NOSORT" for each event without processing it through any sort tree.

streams) and accepting commands (cytometer control changes and new sort programs). The server code is written in DEC C and runs under the VAXELN real-time operating system (Digital Equipment Corp.).

In our system we have a data storage client running on a VAX 6310 and several user clients running on a VAXStation 3100 (Digital Equipment Corp.). They include dot plots, an autocalibrator, the cytometer control panel, the data collection manager, and the sort code generator. The data collection manager, which is part of our FACS/Desk software (7), starts and stops the data storage client and insures that proper sample annotation information is stored with the data. The dot plots display, in real time, any pair of user-selected parameters for instrument and cell sample monitoring. The control panel allows the various instrument parameters such as amplifiers, sort gates, etc., to be set. The sort code generator combines user gating information along with a training set of data it receives from the server to produce a compiled sort tree as described above. This computer program is then sent over the net to the server which loads and starts the slave processor. The slave processor then reads and runs cell data through this tree (or a default program which produces a "NOSORT" for each event without processing it through any sort tree) and returns sort decisions to the cytometer.

Figure 6 shows the timing in this implementation for an 8 parameter sort with an average tree depth of 16. The speeds shown are not inherent in the pattern sorting algorithm but reflect very closely the hardware we implemented this prototype on. The MicroVax slave processor barely runs at 1 MIPS; a modern RISC or CISC processor could easily reduce the sort decision calculation time from 40 to 1 µs or less. Indeed, with careful software and hardware engineering we believe this prototype system can be implemented so that both the network server and sort code run on a single processor.

CONCLUSIONS

Pattern sorting is a generalized method for computer-generated N-dimensional cell classifications in real time. A multidimensional binary tree is constructed from a training data set to which sort decisions have been assigned to each cell. The tree is then compiled into machine instructions which are loaded into a dedicated microprocessor for subsequent execution.

The principal advantage of pattern sorting over conventional bit-mapped lookup techniques is flexibility. Conventional techniques limit sort regions to projections upon planes defined by the data parameter axes. e.g., forward by side scatter and fluorescein by phycoerythrin. These regions are then combined to form hypercubes in the N-dimensional data space. For many applications this will be adequate because the presort analysis breaks down easily into these two-dimen-

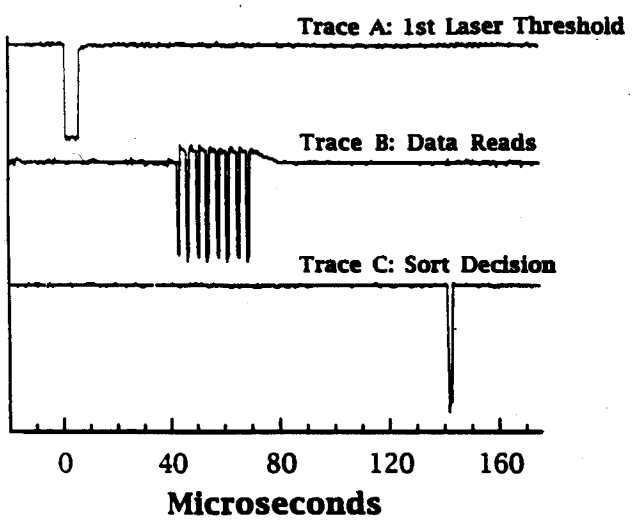


FIG. 6. Sort decision timing. This figure shows the oscilloscope traces monitoring the data and sort decision signals of the slave processor (microVAX) as it runs code generated from a k-d tree of depth 16. They were obtained using a TDS 420 Digitizing Oscilloscope (Tektronix, Inc., Beaverton, OR). Sort gates were chosen so that the full depth of the tree was always traversed. Trace A shows the peak timing for the first laser; it is also the trigger signal for the oscilloscope. Approximately 600 μs after this timing signal the cell will be at the drop break-off point. Trace B shows the eight data reads by the sort processor through the DRV11-J interface (see Fig. 5). They occur approximately 40 μs after the timing signal for two major reasons. First, this cytometer has three lasers; the third laser peak detect is approximately 13 μs after the first. Second, there is a delay in the sort processor polling of the DRV11-J card to avoid contention on the bus shared with the network server. Trace C shows that the sort decision is returned approximately 60 μs after the data read is finished, consistent with the known processor speed of the microVAX. The total time for event processing is 140 μs well within the 600 μs required for a sort decision (see text). Moreover, the sort decision processing time (data reads, decision calculation, and return) is 100 μs, consistent with an overall sort rate of 10,000 events per second.

sional descriptions. However, as more parameters are added to flow cytometers and analysis methods such as N-dimensional clustering or principal component projections are used, defined sort regions may not be accurately characterized by regions in planes defined by the data axes. Pattern sorting, however, will provide very good approximations to these regions and will allow real-time sorting to be based upon them.

LITERATURE CITED

1. Bentley JL: Multidimensional binary search trees used for associative searching. Commun ACM 19:509–517, 1975.
2. Bierre P, Thiel D, Mickaeis SA: Multiparameter cluster analysis using attractors. Abstract: XVI Congress of the International Society for Analytical Cytology, Colorado Springs, CO, March 20–26, 1993. Cytometry Suppl 6:44, 1993.
3. Hoare CAR: Proof of a program: FIND. Commun ACM 19(1):39–45, 1970.
4. KA630-AA CPU Module User's Guide: Part Number EK-KA630-UG-001, Digital Equipment Corporation, Maynard, MA, 1986.
5. Leary JF: Strategies for rare cell detection and isolation. In: Methods in Cell Biology, Ed. 2, Darzynkiewicz Z, Robinson JP, Crissman HA (eds). Academic Press, Orlando, in press.
6. Lindmo T, Fundingsrud K: Measurement of the distribution of time intervals between cell passages in flow cytometry as a method for the evaluation of sample preparation procedures. Cytometry 2:151–154, 1981.
7. Moore WA, Kautz RA: Data analysis in flow cytometry. In: Handbook of Experimental Immunology, Ed. 4, Vol. 1, Weir DM, Herzenberg LA, Blackwell C, Herzenberg LA (eds). Blackwell Scientific Publications, Oxford, 1986, pp 30.1–30.11.
8. Omohundro SM: Efficient algorithms with neural network behavior. Complex Syst 1:273–347, 1987.
9. O'Rourke J, Sloan KR Jr: Dynamic quantization: Two adaptive data structures for multidimensional spaces. IEEE Trans Pattern Anal Mach Intelligence PAMI-6(3):266–279, 1984.
10. Salzman GC, Parson JD, Beckman BJ: Autogate: A Macintosh cluster analysis program for flow cytometry. Abstract: XVI Congress of the International Society for Analytical Cytology, Colorado Springs, CO, March 20–26, 1993. Cytometry Suppl 6:43, 1993.