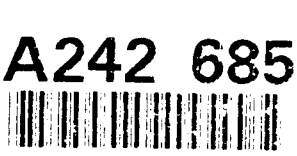


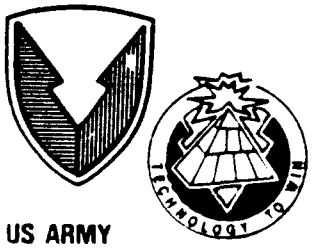
MTL TR 91-36

AD

A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION

I-JONG LIN and DAVID A. BULPETT
POLYMER RESEARCH BRANCH

September 1991



US ARMY
LABORATORY COMMAND
MATERIALS TECHNOLOGY LABORATORY

U.S. ARMY MATERIALS TECHNOLOGY LABORATORY
Watertown, Massachusetts 02172-0001

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

Mention of any trade names or manufacturers in this report shall not be construed as advertising nor as an official indorsement or approval of such products or companies by the United States Government.

DISPOSITION INSTRUCTIONS

Destroy this report when it is no longer needed.
Do not return it to the originator

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MTL TR 91-36	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION	5. TYPE OF REPORT & PERIOD COVERED: Final Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)*: I-Jong Lin and David A. Bulpett*	8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS: U.S. Army Materials Technology Laboratory, Watertown, Massachusetts 02172-0001, SLCMT-EMP	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: D/A Project: AH 84
11. CONTROLLING OFFICE NAME AND ADDRESS: U.S. Army Laboratory Command, 2800 Powder Mill Road, Adelphi, Maryland 20783-1145	12. REPORT DATE: September 1991
	13. NUMBER OF PAGES: 30
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

***Address all correspondence pertaining to this report to David A. Bulpett, U.S. Army Materials Technology Laboratory, Polymer Research Branch, Watertown, MA 02172-0001.**

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Mass spectra
Isotopes
Computer programs

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

(SEE REVERSE SIDE)

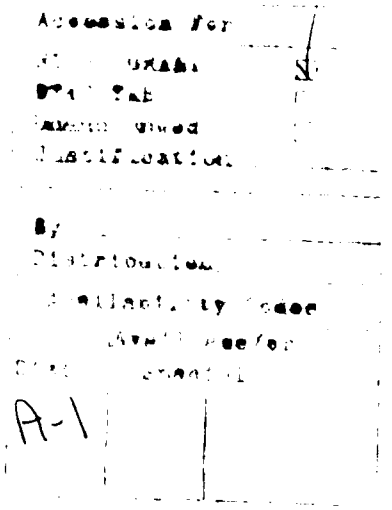
Block No. 20

ABSTRACT

A new isotope cluster identification program, named Reverse Cluster Search (RCS) is presented in full to assist the mass spectrometrist in the identification of unknowns. Given a clean, complete isotope cluster from a mass spectrum, its empirical formula can be found through the calculations of RCS. RCS was tested on known spectra which fit certain parameters of composition and spectral quality. Correlation between the matches given by RCS and the actual formula was excellent. RCS was written to run on both a DEC VAXStation 3500 and IBM PC compatibles. A companion program is also included which is called Forward Cluster Search (FCS). FCS will calculate theoretical mass spectral isotope clusters from empirical formula data.

Contents

	Page
INTRODUCTION	1
EXPERIMENTAL	1
RESULTS AND INTERPRETATION	3
SUMMARY	4
APPENDIX A. TURBO PASCAL CODE FOR RCS	5
APPENDIX B. TURBO PASCAL CODE FOR RSC	17
APPENDIX C. VAX PASCAL CODE FOR FCS	23
APPENDIX D. DATA FILE "CLUSTER.DAT"	27



INTRODUCTION

The purpose of this report is to provide the complete code for a Pascal program which is called Reverse Cluster Search (RCS). Our goal was to develop a computer program to rapidly identify chemical formulas from mass spectral isotope cluster data requiring minimal disk storage and memory.

Relying entirely upon low resolution mass spectral cluster data to predict or confirm empirical formulas is not recommended. High resolution, exact mass data is always preferred when complete confidence is required in the formula assignment; however, the presence of certain isotopic elements must be supported by the cluster data. For instance, one cannot assert that a mass cluster contains a chlorine atom unless the A+2 peak is at least 32.5% of the A peak intensity. Although many programs have been written to calculate the theoretical mass cluster to be expected from a given formula,[1-3] very few have been successful in providing a formula from a given mass cluster.[4-6] RCS requires less disk space and memory than a database approach and is rapid as well. Most clusters can be processed within a minute on a simple IBM PC.

From good quality isotope cluster ratios, RCS calculates partial empirical formulas of the common A+ elements: Br, Cl, Si, S, C, N, and O. RCS uses an iterative approach to calculating clusters. RCS starts its search at a monoisotopic ion and then adds each of the isotopic elements one at a time in the order of decreasing effect on the cluster (as listed above). As an option, RCS can also start its search at a given empirical formula. RCS can then use that fragment formula as a minimum limit on the elemental composition of the target cluster to shorten its search. By monitoring the difference between the target and calculated cluster, RCS can direct the calculated cluster toward the target.

The cluster submitted to RCS will most likely contain some elements which do not alter the observed intensity ratios; therefore, RCS calculates all possible complete formulas by adding combinations of monoisotopic elements, such as F, P, I, and H, until the correct weight is reached. Hydrogen is treated as monoisotopic since the natural abundance of ^{2}H and ^{3}H has a negligible effect on low molecular weight clusters. Certain A- elements such as B (^{10}B :^{11}B occurs in nature in a 1:4 ratio) cannot be handled by RCS at this time. Thus, a search result with a very poor match to the target cluster may contain an element such as B or one of the isotopic metals.

EXPERIMENTAL

RCS was run in VAX Pascal using a Digital Equipment Corporation (Maynard, Massachusetts) VAXStation 3500 computer. The program was also rewritten in Turbo Pascal for

1. CAI, M., and YU, S. *A BASIC Program for Calculating Isotopic Peak Abundances of Organic Molecules in Mass Spectrum*. Jisuanji Yu Yingyong Huaxue, v. 6, no. 6, 1989, p. 23.
2. HIBBERT, D. B. *A Prolog Program for the Calculation of Isotope Distributions in Mass Spectrometry*. Chemom. Intell. Lab. Syst., v. 6, no. 3, 1989, p. 203.
3. KUWABARA, H., TAMURA, Y., and KUBOTA, T. *Computer Programs for the Calculation of Isotope Peak Intensities in Mass Spectra*. Kiyo - Suzuka Kogyo Koto Senmon Gakko, v. 22, no. 1, 1989, p. 37.
4. EVANS, J. E., and JURINSKI, N. B. *Program ELAL: An Interactive Minicomputer Based Elemental Analysis of Low and Medium Resolution Mass Spectra*. Anal. Chem., v. 47, no. 6, 1975, p. 961.
5. TENHOSAARI, A. *Computer-assisted Composition Analysis of Unknown Compounds by Simultaneous Analysis of the Intensity Ratios of Isotope Patterns of the Molecular Ion and Daughter Ions in Low-resolution Mass Spectra*. Org. Mass Spectrom., v. 23, 1988, p. 236.
6. DO LAGO, C. L., and KASCHERES, C. *New Method of Isotope Pattern Analysis*. Computers Chem., v. 15, no. 2, 1991, p. 149.

use on an IBM PC. Table 1 provides a sample input for RCS. When RCS is run, the first prompt is for "TOLERANCE?". As a rule of thumb, every 1% deviation in a cluster is equivalent to 150 points of tolerance. Thus, a cluster with a 0.33% deviation would require a tolerance of about 50 to secure matches within the deviation.

The second prompt is for "WEIGHT LIMIT?". Enter the mass of the A peak from the unknown. RCS will then ask for the cluster, prompting the user to enter an A-1 intensity and the intensities of subsequent masses in the cluster. Raw data may be entered since the program automatically normalizes the cluster so that the most abundant intensity is 100%. After entering a value for up to seven mass intensities, RCS will remove any M-H effects using the method described in DO Lago and Kascheres.[6] RCS can be easily changed to accept more mass intensities by manipulating the program constants: MaxMassOffset and MaxNecc, as defined in the code (see Appendix A).

Table 1. SAMPLE RCS INPUT SESSION

```
Tolerance:150
Weight Limit:228
A-1 Peak Adjustment:0
A+ 0:25.31
A+ 1:2.52
A+ 2:24.81
A+ 3:2.49
A+ 4:0.20
A+ 5:0
A+ 6:0
Do you want default values [Y/N]?n
Max # of Br(Default=2):2
Max # of Cl(Default=6):5
Max # of Si(Default=8):0
Max # of  S(Default=7):0
Max # of  C(Default=19):19
Max # of  N(Default=16):0
Max # of  O(Default=14):14
Max # of  I(Default=1):0
Max # of  P(Default=7):0
Max # of  F(Default=12):0
Start search at fragment ion [Y/N]?Y
How many Br in Fragment Ion?0
How many Cl in Fragment Ion?0
How many Si in Fragment Ion?0
How many  S in Fragment Ion?0
How many  C in Fragment Ion?1
How many  N in Fragment Ion?0
How many  O in Fragment Ion?0
How many  I in Fragment Ion?0
How many  P in Fragment Ion?0
How many  F in Fragment Ion?0
How many  H in Fragment Ion?3
Start.
Done with 3 matches.
```

RCS will then ask whether default values should be applied to the search. The default for the limit on the number of each element is the weight of the A peak divided by the atomic weight of the element. If "N" is entered by the user, the program asks for limits on all elements, showing the default value for comparison.

The last prompt asks the user whether or not to start the search from a fragment ion composition. If the molecular formula of a fragment ion is known, the user can enter that formula to narrow the search. After this final piece of information, RCS searches for matches within tolerance and outputs the results to a textfile named "CLUSTER.OUT" (see Table 2).

Table 2. SAMPLE OUTPUT FOR RCS

```
The cluster adjusted for A-1 contributions:
A+ 0 100.0%:================================================
A+ 1  10.0%:=====
A+ 2  98.0%:===============================================
A+ 3   9.8%:=====
A+ 4   0.8%:
A+ 5   0.0%:
A+ 6   0.0%:

Computed Radii of Effect (x100)
 Br: 2.1E+04 Cl: 1.2E+04 Si: 3.1E+03  S: 3.1E+03
  C: 3.1E+03  N: 5.8E+02  O: 5.8E+02

Compounds:
   Br   Cl   Si    S    C    N    O    I    P    F    H Tolerance    MW
-------------------------------------------------------------------------
Limits:
    2    5    0    0   19    0   14    0    0    0            150   228

Start (MW=15):
    0    0    0    0    1    0    0    0    0    0    3    6975.7

    1    0    0    0    8    0    3    0    0    0    5      91.8
    1    0    0    0    9    0    2    0    0    0    9      34.7
    1    0    0    0   10    0    1    0    0    0   13      92.2
End of 3 Matches.
```

RESULTS AND INTERPRETATION

In the output file, RCS prints out the cluster to be searched to the first decimal place. M-H contributions have already been filtered out. Below the cluster, the computed search parameters and limits on the number of each element are printed. Then, viable empirical formulas, which were found by RCS and have a molecular weight specified by the user and whose errors are within tolerance, are listed below. Viability is determined by the formula found in Kavanagh.[7]

RCS was tested on an IBM PC using data from DeFargo and Kascheres,[6] and was compared to results from their program, AC. The tolerance was set at 150, approximately

7. KAVANAGH, P. E. *Program for Elemental Analysis Using Low or Medium Resolution Mass Spectra.* Org. Mass Spectrom., v. 15, no. 7, 1980, p. 334.

equivalent to 1.0% deviation. Stack space allocated was estimated at under 8 kilobytes of memory maximum. Three matches were found. RCS gave the correct formula, $C_9H_9O_2Br$, as the first choice as did AC. However, AC took 25 seconds to run while RCS worked in seven seconds on a similar computer system using the same limits and starting point. The output described above is provided in Table 2. Both input and output are shown as an aid to understanding the function of this program and as a test example for others running this program for the first time. In the sample program session given in Table 1, the search was started using a minimum formula of CH_3; however, this setting was only added as an example to illustrate the program's capability and function but was not used in the time trial.

SUMMARY

RCS is a useful tool for a mass spectrometrist to narrow down the possibilities of an empirical formula for an unknown ion. RCS analysis of a cluster with good resolution can give a mass spectrometrist an estimate of a compound's molecular or fragment chemical formula from a single mass spectral scan. RCS is seen as a handy on-line reference source for a mass spectrometrist. RCS requires only common and readily available computer hardware. The program is about 680 lines of commented Pascal. Compiled, the program is about 16 kilobytes long, which easily fits onto a low density 5-1/4 inch floppy disk. Code for RCS in Turbo Pascal is included in Appendix A. To rewrite the program for VAX Pascal, change the ASSIGN statement to OPEN and all packed integer arrays to integer arrays. In conjunction with other techniques of identification, RCS can either provide evidence for the proposed formula of a compound or limit the possible formulas of an unknown compound.

As an added check, the program Forward Cluster Search (FCS) calculates a theoretical isotope cluster from empirical formula data. Turbo Pascal code for FCS is included in Appendix B and VAX Pascal code in Appendix C. Both versions of the program require a datafile "CLUSTER.DAT" which is included in Appendix D. The program uses an established technique for calculation of isotopic clusters.[8] Data for isotope abundances are calculated from values, taken from *Handbook of Chemistry and Physics 66th Edition*.[9]

8. ROBINSON, R. J., WARNER, C. G., and GOHLKE, R. S. *The Calculation of Relative Abundance of Isotope Clusters in Mass Spectrometry*. J. Chem. Educ., v. 47, no. 6, 1970, p. 467.
9. *Handbook of Chemistry and Physics 66th Edition*. Weast, R. C., ed., CRC Press, Boca Raton, FL, 1986.

APPENDIX A. TURBO PASCAL CODE FOR RCS

```
program ReverseClusterSearch (Input,Output);

{Constants defined here are explained as follows:
     MaxMassOffset: the furthest A+ mass from the A peak which will be
                    in calculations.
     MaxNecc: Number of elements necessary in array for computation
              2+MaxMassOffset
These constants can be varied to accept larger clusters.}

const
    MaxMassOffset = 6;
    MaxNecc = 8;

type
   element = (Br,Cl,Si,S,C,N,O,over,I,P,F,H);
   compound =  packed array[Br..O] of integer;
   excompound = packed array[I..H] of integer;
   cluster = packed array [0..MaxNecc] of real;
   offsetNum = packed array[Br..Over] of real;
   formula = record
               iso:compound;
               noniso:excompound;
             end;

{Global variables are explained as follows:
    outfile: text file which receives output information
    Offn1,Offn2: arrays into which the offset values are placed for
                 efficiency
    Weightvalue: array into which monoisotopic weight for each element
                 are placed for efficiency
}

var Offn1,Offn2:offsetnum;
    weightvalue:compound;
    out:text;

function min(x,y:integer):integer;
begin
    if x>y then min:=y else min:=x;
end;

function Off1(elem:element):real;
{Given an element, this function returns the +1 percentage ratio to the A
peak.}

begin
    case elem of
         C:Off1:=1.1;
         Si:Off1:=5.1;
         S:Off1:=0.8;
         Cl:Off1:=0.0;
         Br:Off1:=0.0;
         N:Off1:=0.37;
         O:Off1:=0.04;
    end;
end;

function Off2(elem:element):real;
{Given an element, this function returns the +2 percentage ratio to the A
peak.}
```

```
begin
    case elem of
         C:Off2:=0.02;
         Si:Off2:=3.4;
         S:Off2:=4.4;
         Cl:Off2:=32.5;
         Br:Off2:=98.0;
         N:Off2:=0.0;
         O:Off2:=0.2;
    end;
end;

function Weight(elem:element):integer;
{Given an element, this function returns its unit weight.}

begin
    case elem of
         C:Weight:=12;
         Si:weight:=28;
         S:weight:=32;
         Cl:weight:=35;
         Br:weight:=79;
         N:weight:=14;
         O:weight:=16;
         I:weight:=127;
         H:weight:=1;
         F:weight:=19;
         P:weight:=31;
    end;
end;

procedure ReadValues;
{This procedure sets the global arrays to their particular values, avoiding
the repetitive calls of the functions.}

var count:element;
begin
   for count:=Br to pred(over) do
         begin
            offn1[count]:=off1(count);
            offn2[count]:=off2(count);
            weightvalue[count]:=weight(count);
         end;
end;


function valid(var frm:formula):boolean;
{This function is the final validity test of a compound, including the
elements which have no effect on the cluster.}

var temp:integer;
begin
    temp:=2*frm.iso[C]+frm.iso[N]+2*frm.iso[Si]+3*frm.noniso[P]+
          2-(frm.noniso[H]+frm.noniso[I]+frm.noniso[F]+
          frm.iso[Cl]+frm.iso[Br]);
    if temp>=0 then valid:=true
       else valid:=false;
end;

procedure NormalizeByArea(var workspace:cluster);
{Given a cluster, this procedure normalizes the sum of all intensities
to 100.}

var count:integer;
    sum:real;
begin
```

```
    sum:=0;
    for count:=0 to MaxMassOffset do
            sum:=sum+workspace[count];
    for count:=0 to MaxMassOffset do
         workspace[count]:=workspace[count]/sum*100;
end;

procedure normalizeByPeak(var workspace:cluster);
{Given a cluster, this procedure normalizes the peak to 100.}

var count:integer;
    sum:real;
begin
    sum:=0;
    for count:=0 to MaxMassOffset do
         if workspace[count]<0 then workspace[count]:=0.0;
    for count:=0 to MaxMassOffset do
         if sum<workspace[count] then sum:=workspace[count];
    for count:=0 to MaxMassOffset do
         workspace[count]:=workspace[count]/sum*100;
end;

procedure Offset(var workspace:cluster;M1,M2:real);
{Given a cluster and the percentage ratios of +1 and +2 isotopes to the
A peak, this procedure changes the cluster by adding the effect of one
isotopic atom without changing the sum of intensities.}

var count:integer;
    sum:real;
begin
    sum:=M1+M2+100.0;
    for count:=MaxMassOffset downto 0 do
        workspace[count+2]:=workspace[count+2]*(100/sum,+
                            workspace[count+1]*(M1/sum)+
                            workspace[count]*(M2/sum);
    workspace[1]:=workspace[1]*(100/sum)+workspace[0]*(m1/sum);
    workspace[0]:=workspace[0]*(100/sum);
end;


procedure PrElem(var outfile:text;elem:element;count:integer);
{Given an element, this procedure prints its name.}

begin
    case elem of
         C:write(outfile,'C':count);
         Si:write(outfile,'Si':count);
         S:write(outfile,'S':count);
         Cl:write(outfile,'Cl':count);
         Br:write(outfile,'Br':count);
         N:write(outfile,'N':count);
         O:write(outfile,'O':count);
         P:write(outfile,'P':count);
         F:write(outfile,'F':count);
         I:write(outfile,'I':count);
         H:write(outfile,'H':count);
    end;
end;

procedure SetSpace(var workspace:cluster);
{Given a cluster, this procedure clears it and starts the cluster at its
basis case, a single mass peak.}

var count:integer;
begin
    for count:=1 to MaxNecc do workspace[Count]:=0.0;
```

```
    workspace[0]:=100.0;
end;

procedure ClearExtra(var x2:excompound);
{This procedure clears the array that keep track of the elements that have
no effect on the cluster.}

var count:element;
begin
     for count:=I to H do x2[count]:=0;
end;

procedure NewComp(var x:compound);
{Given compound, this procedure clears the array of isotopic elements.}

var count:element;
begin
   for count:=Br to pred(over) do
        x[count]:=0;
end;


function Error(var x,y:cluster):real;
{Given two clusters, this function returns their Euclidian distance from each
other times 100 as represented by the vectors they represent.}

var count:integer;
    temp:real;
begin
    temp:=0;
    for count:=0 to MaxMassOffset do
          temp:=temp+sqr(x[count]-y[count]);
    Error:=sqrt(temp)*100;
end;


procedure PrintComp(var outfile:text;var x:formula;err:real);
{Given a compound and error, this procedure will print out all necessary
and formatted information.}

var count:element;
    temp:integer;

begin
   for count:=Br to pred(over) do
        write(outfile,x.iso[count]:5);
   for count:=I to H do
        write(outfile,x.noniso[count]:5);
   writeln(outfile,err:10:1);
end;


function Effect(elem:element;num:integer):real;
{This function finds the maximum error between two clusters which can be
overcome by a certain number of a certain element.}

var count:integer;
    x,y:cluster;
begin
     setspace(x);
     setspace(y);
     for count:=1 to num do
           Offset(y,offn1[elem],offn2[elem]);
     Effect:=error(x,y);
end;
```

```
procedure ComputeRadii(tol:real;var max,start:compound;var wayoff:offsetnum);
{This procedure computes the cumalative levels of error that can be overcome
at each level of recursion into global array.  When the index is OVER, no
isotopic elements will be added and then the level of error is equivalent
to the tolerance.  Tolerance level is added into all Radii of Effect.}

var elem1,elem2:element;
begin
   for elem1:=Br to pred(over) do
          wayoff[elem1]:=Effect(Elem1,max[elem1]);
   wayoff[over]:=tol;
   for elem1:=Br to pred(over) do
       for elem2:=succ(elem1) to over do
              wayoff[elem1]:=wayoff[elem1]+wayoff[elem2];
end;

function MW(var f:formula):integer;
{Given a formula, this function returns the molecular weight.}
var sum:integer;
    elem:element;

begin
    sum:=0;
    for elem:=Br to pred(over) do
        sum:=sum+f.iso[elem]*weightvalue[elem];
    for elem:=I to H do
        sum:=sum+f.noniso[elem]*weight(elem);
    MW:=sum;
end;

procedure MakeCluster(var f:formula;var c:cluster);
{Given a formula and a cluster, this procedure calculates the corresponding
cluster for the element.}
var elem:element;
    count:integer;

begin
    SetSpace(c);
    for elem:=Br to pred(over) do
            for count:=1 to f.iso[elem] do
                offset(c,Offn1[elem],offn2[elem]);
end;


procedure PrintMaster(var outfile:text;var master:cluster);
{This procedure sets up the formats the text file for the output from
FindCombos, printing out all the settings of the search and the header
for the list of matches. }

var count1,count2:integer;
begin
    rewrite(outfile);
    writeln(outfile,'The cluster adjusted for A-1 contributions:');
    for count1:=0 to MaxMassOffset do
        begin
           write(outfile,'A+',count1:2,' ',master[count1]:5:1,'%:');
           for count2:=1 to round(master[count1]/2) do
                    write(outfile,'=');
           writeln(outfile);
        end;
    writeln(outfile);
end;

procedure PrintRadii(var outfile:text;var wayoff:offsetnum);
{This procedure prints out the Radii of Effect search parameters.}
```

```
var elem:element;
begin
    writeln(outfile,'Computed Radii of Effect (x100)');
    for elem:=Br to pred(over) do
        begin
            PrElem(outfile,elem,3);
            write(outfile,':',wayoff[elem]:1);
            if elem=S then writeln(outfile);
        end;
    writeln(outfile);
    writeln(outfile);
end;

procedure PrintLimits(var outfile:text;var max:formula;tol,w:integer);
{This procedure prints out the limit search parameters.}

var elem:element;
begin
    writeln(outfile,'Compounds:');
    for elem:=Br to pred(over) do Prelem(outfile,elem,5);
    for elem:=I to H do PrElem(outfile,elem,5);
    writeln(outfile,'Tolerance':10,'MW':6);
    for elem:=Br to H do write(outfile,'-----');
    writeln(outfile,'-----------------');
    writeln(outfile,'Limits:');
    for elem:=Br to pred(over) do write(outfile,max.iso[elem]:5);
    for elem:=I to F do write(outfile,max.noniso[elem]:5);
    writeln(outfile,' ':5,Tol:10,w:6);
    writeln(outfile);
end;


procedure PrintStart(var outfile:text;var start:formula;err:real);
{This procedure prints out the chemical formula from which the search
will originate.}

begin
   writeln(outfile,'Start (MW=',MW(start):1,'):');
   printcomp(outfile,start,err);
   writeln(outfile);
end;


function FindPossible(var outfile:text;var x:formula;err:real;
                          leftover:integer;var max:excompound):integer;
{Given a file to print to, a compound, its corresponding error and the
discrepancy in weight, this function prints out all possible combinations
of non-isotopic elements which account for the discrepancy to the specified
file and returns the number of permutations which it finds.  The nested
procedure does the actual recursion; the outer procedure acts as a
template and sets the starting point for recursion.}

var hit:integer;
    temp:formula;

    procedure _FindPossible(x2:excompound;w:integer;elem2:element);
    {By recursive descent, this procedure finds and prints out the
    values (by a non-local reference) all variations between
    the elements which cause no isotopic effect and accounts for the
    discrepancies in the weight.  A For-loop is used to avoid unnecessary
    stack space usage.}

    var count:integer;
    begin
       if (elem2=H) and (w>=0) then
          begin
```

```
            x2[H]:=x2[H]+w;
            temp.noniso:=x2;
            temp.iso:=x.iso;
            if Valid(temp) then
               begin
                 PrintComp(outfile,temp,err);
                 hit:=hit+1;
               end;
          end
          else
       if w=0 then
          _FindPossible(x2,w,H)
          else
            begin
            _FindPossible(x2,w,succ(elem2));
            for count:=1 to
                  min(max[elem2],(w div weight(elem2))) do
              begin
              x2[elem2]:=x2[elem2]+1;
              w:=w-weight(elem2);
              _FindPossible(x2,w,succ(elem2));
              end;
         end;
    end;

begin
    hit:=0;
    _FindPossible(x.noniso,leftover,I);
    Findpossible:=hit;
end;


procedure FindCombos(var outfile:text;master:cluster;start,max:formula;
                         tolerance,weightlimit:integer);
var slave:cluster;
    matches:integer;
    wayoff:offsetnum;
    starterr:real;
    temp:formula;

{With all global variables set, this procedure recurses the tree of clusters
with the root of the tree specified by the corresponding cluster of the
formula START and outputs those clusters that fall within tolerance.}


  procedure TryCombos(x:compound;xc:cluster;lasterr:real;elem:element;
                      w:integer);
  {This procedure recursively traverses the tree of clusters.  The basis case
  is as follows: recursion ends when the compound is no longer valid, or
  error to the target cluster is increasing instead of decreasing.  The
  inductive step is to check whether the cluster is within tolerance (if so,
  print it) and recurse on the original compound twice: first, by trying
  another element and recursing if the error is within the radius of effect,
  and second, by adding one more of the current element and recursing again.
  The element parameter exists to make sure that no duplicates are recursed
  upon.  The code below follows this theory except has been modified to
  replace much of the recursion with a while-loop to save stack space. }

  var thiserr:real;
  begin
    if elem=over then
      begin
         temp.iso:=x;
         temp.noniso:=start.noniso;
         matches:=matches+FindPossible(outfile,temp,lasterr,w,max.noniso);
      end
```

```
    else
      begin
      thiserr:=lasterr;
      while ((lasterr>=thiserr) or (thiserr<=tolerance))
            and (x[elem]<=max.iso[elem]) and (w>=0) do
         begin
            if (thiserr<=wayoff[succ(elem)]) then
                        trycombos(x,xc,thiserr,succ(elem),w);
            x[elem]:=x[elem]+1;
            offset(xc,Offn1[elem],offn2[elem]);
            lasterr:=thiserr;
            w:=w-weightvalue[elem];
            thiserr:=error(xc,master);
         end;
      end;
  end;


begin
    MakeCluster(start,slave);
    matches:=0;
    ComputeRadii(tolerance,max.iso,start.iso,wayoff);
    NormalizeByPeak(master);
    PrintMaster(outfile,master);
    normalizeByArea(master);
    PrintRadii(outfile,wayoff);
    PrintLimits(outfile,max,tolerance,weightlimit);
    starterr:=error(slave,master);
    PrintStart(outfile,start,starterr);
    writeln('Start.');
    TryCombos(start.iso,slave,starterr,Br,weightlimit-MW(start));
    writeln(outfile,'End of ',matches:1,' Matches.');
    writeln('Done with ',matches:1,' matches.');
end;


function power(x:real;y:integer):real;
{This function computes positive, integer powers of x.}

var d:real;
    count:integer;
begin
     d:=1;
     for count:=1 to y do d:=d*x;
     power:=d;
end;

function MHerror(var x:cluster;ratio:real):real;
{Given a cluster and a guessed ratio between M-H contribution and the
original intensities, this function returns how far off it is.  By treating
the cluster as a polynomial of MaxMassOffset degree, the correct ratio
is found by varying this Error until it is zero.}

var y:cluster;
    count:integer;
    sum:real;
begin
    sum:=0.0;
    for count:=0 to MaxMassOffset do
        sum:=sum+x[count]*power(-ratio,count);
    MHerror:=sum;
end;

function FindRatio(var x:cluster;start,interval:real):real;
{Given a cluster, a starting point and step, this function returns the
value of the correct ratio between M-H contribution and original intensity.
```

```
Every time, since the error function starts out positive, at the point
where the error function passes below 0, the function recurses at the point
before it dips below and with a finer step until step has less than
0.01% effect.}

begin
    if interval<1.0001 then FindRatio:=start
        else
        begin
        while (MHError(x,start*interval)>0) and ((start*interval)<10) do
              start:=start*interval;
        if (start*interval)>=10 then
           FindRatio:=Findratio(x,0.000001,sqrt(interval))
           else
           FindRatio:=FindRatio(x,start,sqrt(sqrt(interval)));
        end;
end;


function ratio(x:cluster;MH:real):real;
{This function sets the cluster for calculation of the ratio for FindRatio,
starts off the recursion and returns this ratio.}

var count:integer;
begin
     for count:=MaxMassOffset downto 0 do
         x[count+1]:=x[count];
     x[0]:=MH;
     normalizebypeak(x);
     ratio:=FindRatio(x,0.000001,5);
end;

procedure Fix(var x:cluster;rat:real);
{Given a cluster and a known ratio between M-H and M intensities, this
procedure corrects the values .}

var count:integer;
begin
     Writeln('Computed Ratio between A-1 and A:',rat);
     for count:=(MaxMassOffset-1) downto 0 do
         x[count]:=x[count]-x[count+1]*rat;
end;


procedure SetMaster(var master:cluster);
{This procedure reads in the target vector to be matched. }

var count:integer;
    MH:real;
begin
   write('A-1 Peak Adjustment:');
   readln(MH);
   for count:=0 to MaxMassOffset do
         begin
            write('A+',count:2,':');
            readln(master[count]);
         end;
   if MH<>0.0 then Fix(master,ratio(master,MH));
   normalizebyPeak(master);
end;


procedure SetSearch(var tol,weightlimit:integer);
{This procedure prompts the user for the Tolerance level and WeightLimit.}

begin
```

```
    write('Tolerance:');
    readln(tol);
    Write('Weight Limit:');
    readln(weightLimit);
end;


procedure SetLimit(var max:compound;weightlimit:integer;
                        var max2:excompound);
{This procedure prompts the users to set limits on number of elements.}

var count:element;
    ch:char;
begin
    write('Do you want default values [Y/N]?');
    readln(ch);
    for count:=Br to pred(over) do
         Max[Count]:=weightlimit div weight(count);
    for count:=I to H do
         max2[count]:=weightlimit div weight(count);
    if (Ch='N') or (Ch='n') then
              begin
              for count:=Br to pred(over) do
                  begin
                    write('Max # of ');
                    PrElem(OUTPUT,count,2);
                    write('(Default=',Max[count]:1,'):');
                    readln(Max[count]);
                  end;
              for count:=I to F do
                  begin
                    write('Max # of ');
                    PrElem(OUTPUT,count,2);
                    write('(Default=',Max2[count]:1,'):');
                    readln(Max2[count]);
                  end;
              end;
end;

procedure ReadNum(elem:element;var x:integer);
begin
    write('How many ');
    PrElem(OUTPUT,elem,2);
    write(' in Fragment Ion?');
    readln(x);
end;


procedure SetStart(var f:formula);
{This procedure prompts the user to enter the formula from which to start
the search.  Default is an empty compound.}

var elem:element;
    ch:char;

begin
    ClearExtra(f.noniso);
    NewComp(f.iso);
    write('Start search at fragmer  .on [Y/N]?'),
    readln(ch);
    if (Ch='Y') or (Ch='Y') then
       begin
         for elem:=Br to Pred(over) do
              ReadNum(elem,f.iso[elem]);
         for elem:=I to H do
              ReadNum(elem,f.noniso[elem]);
```

```
        end;
end;


procedure FindCompounds(var outfile:text);
{Given a file, this procedure asks user for all relevant information
for the search and returns search results into the file.}

var start,max:formula;
    wayoff:offsetnum;
    tolerance,weightlimit:integer;
    master:cluster;
begin
   SetSearch(tolerance,weightlimit);
   SetMaster(master);
   SetLimit(max.iso,weightlimit,max.noniso);
   SetStart(start);
   FindCombos(outfile,master,start,max,tolerance,weightlimit);
end;



begin {main}
   ReadValues;
   assign(out,'cluster.out');
   FindCompounds(out);
   close(out);
end.
```

APPENDIX B. TURBO PASCAL CODE FOR FCS

```
program ForwardClusterSearch (Input,Output);
{Written by I-Jong Lin      Aug. 28,1991}
const
   maxmassoffset = 40;
   minmassoffset = -40;
   MaxCompounds = 50;

type
   elementname = Record
                  ch:array[1..2] of char;
                  two:boolean;
                  end;
   elemlib = record
               names:array [1..MaxCompounds] of elementname;
               num:integer;
             end;
   compound =  packed array[1..MaxCompounds] of integer;
   OffsetMatrix = array [1..MaxCompounds,-10..10] of real;
   cluster = packed array [minmassoffset..maxmassoffset] of real;
   stuff = string[255];



function NoCaps(ch:char):char;
begin
   if ch in ['a'..'z'] then NoCaps:=chr(ord(ch)-ord('a')+ord('A'))
      else NoCaps:=ch;
end;

function CompareNames(var x,y:elementname):boolean;
begin
    CompareNames:= (x.two=y.two) and (NoCaps(x.ch[1])=NoCaps(y.ch[1]))
                           and ( (NoCaps(x.ch[2])=NoCaps(y.ch[2]))
                           or (not x.two));
end;

procedure PrintElementName(var outfile:text;x:elementname);
begin
    write(outfile,x.ch[1]);
    if x.two then write(outfile,x.ch[2]);
end;

function index(var lib:elemlib;var elem:elementname):integer;
var count,temp:integer;
begin
   temp:=0;
   for count:=1 to lib.num do
         begin
           if comparenames(elem,lib.names[count]) then
                 temp:=count;
         end;
   if (temp=0) and not elem.two then begin
                    write('Error!  The element not found.');
                  end;
   index:=temp;
```

```
end;

procedure SkipSpaces(var data:stuff;var count:integer);
var ch:char;

begin
   while data[count]=' ' do count:=count+1;
end;

procedure ReadElementName(var data:stuff;var lib:elemlib;var x:elementname;
                                  var count:integer);
begin
    x.ch[1]:=data[count];
    count:=count+1;
    x.two:=(NoCaps(data[count]) in ['A'..'Z']);
    if x.two then begin
                   x.ch[2]:=data[count];
                   count:=count+1;
                  end;
    if index(lib,x)=0 then
                      begin
                         x.two:=false;
                         count:=count-1;
                      end;
    Skipspaces(data,count);
end;

procedure ReadElementName2(var infile:text;var x:elementname);
begin
    read(infile,x.ch[1],x.ch[2]);
    x.two:=(NoCaps(x.ch[2]) in ['A'..'Z']);
end;

function number(var ch:char):boolean;
begin
   number:=(ch in ['0'..'9']);
end;

function NextNumber(var data:stuff;var count:integer):integer;
var temp:integer;
begin
    if not number(data[count]) then
           begin
            nextnumber:=1;
           end
           else
           begin
            temp:=0;
            while number(data[count]) do
              begin
                temp:=temp+ord(data[count])-ord('0');
                temp:=temp*10;
                count:=count+1;
              end;
            NextNumber:=temp div 10;
           end;
end;


procedure ReadMatrix(var infile:text;var lib:elemlib;var off:offsetmatrix);
var count1,count2:integer;
begin
    for count1:=1 to MaxCompounds do
         for count2:=-10 to 10 do
             if count2=0 then off[count1,0]:=100
                 else
```

```
                    off[count1,count2]:=0;
    lib.num:=0;
    while not eof(infile) do
        begin
           lib.num:=lib.num+1;
           ReadElementName2(infile,lib.names[lib.num]);
           while not eoln(infile) do
              begin
               read(infile,count2);
               read(infile,off[lib.num,count2]);
              end;
           readln(infile);
        end;
end;

procedure newCompound(var workspace:compound);
var count:integer;
begin
    for count:=1 to MaxCompounds do
           workspace[count]:=0;
end;

procedure clear(var workspace:cluster);
var count:integer;
begin
    for count:=minmassoffset to maxmassoffset do
           workspace[count]:=0;
end;


procedure normalize(var workspace:cluster);
var count:integer;
    sum:real;
begin
    sum:=0;
    for count:=MinMassOffset to MaxMassOffset do
         if sum<workspace[count] then sum:=workspace[count];
    for count:=MinMassOffset to MaxMassOffset do
         workspace[count]:=workspace[count]/sum*100;
end;


procedure Offset(var workspace:cluster;var off:offsetmatrix;
                 elem:integer);
var count1,count2:integer;
    bottom,top:integer;
    sum:real;
    temp:cluster;
begin
    clear(temp);
    for count1:=-10 to 10 do
        begin
        if count1<0 then begin
                            bottom:=minmassoffset;
                            top:=maxmassoffset+count1;
                         end
                    else
                         begin
                            bottom:=minmassoffset+count1;
                            top:=maxmassoffset;
                         end;
        for count2:=bottom to top do
             temp[count2]:=temp[count2]
                    +workspace[count2-count1]*off[elem,count1]/100;
        end;
    workspace:=temp;
```

```
end;


procedure SetSpace(var workspace:cluster);
var count:integer;
begin
    for count:=MinMassOffset to MaxMassOffset do workspace[Count]:=0.0;
    workspace[0]:=100.0;
end;


procedure PrintComp(var outfile:text;var x:compound;var lib:elemlib);
var count:integer;
begin
   writeln(outfile,'Emprical Formula for Compound:');
   for count:=1 to MaxCompounds do
        if x[count]<>0 then
           begin
             PrintElementName(outfile,lib.names[count]);
             write(outfile,':',x[count]:1,' ');
           end;
   writeln(outfile);
end;


procedure EnterCompound(var x:compound;var lib:elemlib;var data:stuff);
var count,i:integer;
    elem:elementname;

begin
    newcompound(x);
    Writeln('The ',lib.num:1,' Possible Substituents:');
    for count:=1 to lib.num do
        begin
          PrintElementName(OUTPUT,lib.names[count]);
          write(' ');
          if (count mod 20)=19 then writeln;
        end;
    writeln;
    for count:=1 to 255 do data[count]:=' ';
    write('Enter Compound (e.g. HC3OOH):');
    readln(data);
    count:=1;
    while count<=ord(data[0]) do
      begin
         skipspaces'data,count);
         ReadElementName(data,lib,elem,count);
         i:=index(lib,elem);
         x[i]:=x[i]+NextNumber(data,count);
      end;
end;


procedure SetFile(master:cluster;x:compound;var lib:elemlib;s:stuff);
var count1,count2:integer;
    outfile:text;
    top,bottom:integer;
begin
    assign(outfile,'TCluster.out');
    rewrite(outfile);
    writeln(outfile,'Actual Chemical Formula:');
    writeln(outfile,s);
    printcomp(outfile,x,lib);
    writeln(outfile,'The cluster computed:');
    top:=maxmassoffset;
```

```
    while master[top]<0.01 do top:=top-1;
    bottom:=minmassoffset;
    while master[bottom]<0.01 do bottom:=bottom+1;
    for count1:=bottom to top do
        begin
           if count1>=0 then
                write(outfile,'A+',count1:2,' ',(master[count1]):6:2,'%:')
                   else
                write(outfile,'A-',abs(count1):2,' ',(master[count1]):6:2,
                      '%:');
           for count2:=1 to round(master[count1]/2) do
                     write(outfile,'=');
           writeln(outfile);
        end;
    writeln(outfile);
    close(outfile);
end;

procedure ComputeCluster;
var x:compound;
    xc:cluster;
    count1,count2:integer;
    off:offsetmatrix;
    lib:elemlib;
    infile:text;
    data:stuff;

begin
    assign(infile,'cluster.dat');
    reset(infile);
    ReadMatrix(infile,lib,off);
    close(infile);
    EnterCompound(x,lib,data);
    Setspace(xc);
    for count1:=1 to lib.num do
          for count2:=1 to x[count1] do
                Offset(xc,off,count1);
    normalize(xc);
    setfile(xc,x,lib,data);
end;


begin {main}
    ComputeCluster;
end.
```

APPENDIX C. VAX PASCAL CODE FOR FCS

```
program ForwardClusterSearch (Input,Output);

const
   maxmassoffset = 40;
   minmassoffset = -40;

type
   element = (C,H,B,N,O,Si,S,Cl,Ti,V,Cr,Fe,Ni,
              Cu,Zn,Ga,Ge,Se,Br,Zr,Mo,Ru,Pd,Cd,
              Sn,Sb,Te,Hg,Pb,over);
   compound =  packed array[C..over] of integer;
   OffsetMatrix = array [C..over,-10..10] of real;
   cluster = packed array [minmassoffset..maxmassoffset] of real;

procedure ReadMatrix(var off:offsetmatrix;var infile:text);
var elem:element;
    count:integer;
begin
    for elem:=C to pred(over) do
         for count:=-10 to 10 do
             if count=0 then off[elem,0]:=100 else off[elem,count]:=0;
    while not eof(infile) do
        begin
           read(infile,elem);
           while not eoln(infile) do
              begin
               read(infile,count);
               read(infile,off[elem,count]);
              end;
           readln(infile);
        end;
end;

procedure newCompound(var workspace:compound);
var count:element;
begin
    for count:=C to pred(over) do
           workspace[count]:=0;
end;

procedure clear(var workspace:cluster);
var count:integer;
begin
    for count:=minmassoffset to maxmassoffset do
           workspace[count]:=0;
end;


procedure normalize(var workspace:cluster);
var count:integer;
    sum:real;
begin
    sum:=0;
    for count:=MinMassOffset to MaxMassOffset do
         if sum<workspace[count] then sum:=workspace[count];
    for count:=MinMassOffset to MaxMassOffset do
         workspace[count]:=workspace[count]/sum*100;
```

```
end;


procedure Offset(var workspace:cluster;var off:offsetmatrix;
                 elem:element);
var count1,count2:integer;
    bottom,top:integer;
    sum:real;
    temp:cluster;
begin
    clear(temp);
    for count1:=-10 to 10 do
        begin
        if count1<0 then begin
                            bottom:=minmassoffset;
                            top:=maxmassoffset+count1;
                         end
                    else
                        begin
                            bottom:=minmassoffset+count1;
                            top:=maxmassoffset;
                         end;
        for count2:=bottom to top do
             temp[count2]:=temp[count2]
                    +workspace[count2-count1]*off[elem,count1]/100;
        end;
    workspace:=temp;
end;



procedure SetSpace(var workspace:cluster);
var count:integer;
begin
    for count:=MinMassOffset to MaxMassOffset do workspace[Count]:=0.0;
    workspace[0]:=100.0;
end;


procedure PrintComp(var outfile:text;var x:compound);
var count:element;
begin
   writeln(outfile,'Compound:');
   for count:=C to pred(over) do
        if x[count]<>0 then write(outfile,count:2,':',x[count]:1,' ');
   writeln(outfile);
end;


procedure EnterCompound(var x:compound);
var elem:element;

begin
    newcompound(x);
    repeat
      write('Components:');
      for elem:=c to pred(over) do
          write(elem:3);
      writeln;
      printcomp(OUTPUT,x);
      write('Which element (over to compute):');
      readln(elem);
      if elem<>over then
                      begin
                          write('How many:');
                          readln(x[elem]);
```

```
                        end;
      writeln('----------------');
    until (elem=over);
end;


procedure SetFile(master:cluster;x:compound);
var count1,count2:integer;
    elem:element;
    outfile:text;
    top,bottom:integer;
begin
    open(outfile,'Cluster.out');
    rewrite(outfile);
    printcomp(outfile,x);
    writeln(outfile,'The cluster computed:');
    top:=maxmassoffset;
    while master[top]<0.4 do top:=top-1;
    bottom:=minmassoffset;
    while master[bottom]<0.4 do bottom:=bottom+1;
    for count1:=bottom to top do
        begin
           if count1>=0 then
                write(outfile,'A+',count1:2,' ',(master[count1]):5:2,'%:')
                   else
                write(outfile,'A-',abs(count1):2,' ',(master[count1]):5:2,
                     '%:');
           for count2:=1 to round(master[count1]/2) do
                     write(outfile,'=');
           writeln(outfile);
        end;
    writeln(outfile);
end;

procedure ComputeCluster;
var x:compound;
    xc:cluster;
    elem:element;
    count:integer;
    off:offsetmatrix;
    infile:text;

begin
    open(infile,'cluster.dat',ReadOnly);
    reset(infile);
    ReadMatrix(off,infile);
    EnterCompound(x);
    Setspace(xc);
    for elem:=C to pred(over) do
          for count:=1 to x[elem] do
                Offset(xc,off,elem);
    normalize(xc);
    setfile(xc,x);
    close(infile);
end;


begin {main}
    Computecluster;
end.
```

APPENDIX D. DATA FILE "CLUSTER.DAT"

```
C 1 1.1 2 0.02
Si 1 5.1 2 3.4
S 1 0.8 2 4.4
Cl 2 32.5
Br 2 98.0
O 1 0.04 2 0.2
H 1 0.015
N 1 0.37
B -1 24.69
Ti -2 10.84 -1 9.89 1 7.45 2 7.32
V -1 0.25
Zn 2 57.41 3 8.44 4 38.68 6 1.23
Ge -4 56.1 -2 75.07 -1 21.37 2 21.37
Se -6 1.81 -4 18.15 -3 15.32 -2 47.38 2 18.95
Zr 1 21.90 2 33.37 4 33.68 6 5.40
Sn -8 3.09 -6 2.16 -5 1.23 -4 45.37 -3 23.77 -2 75.00 -1 26.54 2 14.20 4 17.47
Sb 2 74.52
Te -10 0.28 -8 7.69 -7 2.67 -6 14.25 -5 1.23 -4 56.07 -2 93.76
Pb -4 2.67 -2 45.99 -1 42.17
Ni 2 38.23 3 1.66 4 5.26 6 1.33
Mo -6 61.50 -4 38.33 -3 65.98 -2 69.13 -1 39.58 2 39.91
Fe -2 6.32 1 2.40 2 0.31
Cr -2 5.19 1 11.34 2 2.81
Cu 2 4.457
Cd -8 4.35 -6 3.10 -4 43.47 -3 44.55 -2 83.99 -1 42.53 1 26.07
Ga 2 66.39
Hg -6 0.51 -4 34.06 -3 57.34 -2 77.91 -1 44.52 2 22.93
Pd -4 3.73 -2 40.76 -1 81.71 2 96.82 4 42.88
Ru -6 17.47 -4 5.95 -3 40.19 -2 39.87 -1 53.80 2 59.18
```

DISTRIBUTION LIST

No. of Copies	To
1	Office of the Under Secretary of Defense for Research and Engineering, The Pentagon, Washington, DC 20301
	Commander, U.S. Army Laboratory Command, 2800 Powder Mill Road, Adelphi, MD 20783-1145
1	ATTN: AMSLC-IM-TL
1	AMSLC-CT
	Commander, Defense Technical Information Center, Cameron Station, Building 5, 5010 Duke Street, Alexandria, VA 22304-6145
2	ATTN: DTIC-FDAC
1	MIAC/CINDAS, Purdue University, 2595 Yeager Road, West Lafayette, IN 47905
	Commander, Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709-2211
1	ATTN: Information Processing Office
	Commander, U.S. Army Materiel Command, 5001 Eisenhower Avenue, Alexandria, VA 22333
1	ATTN: AMCSCI
	Commander, U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD 21005
1	ATTN: AMXSY-MP, H. Cohen
	Commander, U.S. Army Missile Command, Redstone Scientific Information Center, Redstone Arsenal, AL 35898-5241
1	ATTN: AMSMI-RD-CS-R/Doc
1	AMSMI-RLM
	Commander, U.S. Army Armament, Munitions and Chemical Command, Dover, NJ 07801
1	ATTN: Technical Library
	Commander, U.S. Army Natick Research, Development and Engineering Center, Natick, MA 01760-5010
1	ATTN: Technical Library
	Commander, U.S. Army Satellite Communications Agency, Fort Monmouth, NJ 07703
1	ATTN: Technical Document Center
	Commander, U.S. Army Tank-Automotive Command, Warren, MI 48397-5000
1	ATTN: AMSTA-ZSK
1	AMSTA-TSL, Technical Library
	Commander, White Sands Missile Range, NM 88002
1	ATTN: STEWS-WS-VT
	President, Airborne, Electronics and Special Warfare Board, Fort Bragg, NC 28307
1	ATTN: Library
	Director, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD 21005
1	ATTN: SLCBR-TSB-S (STINFO)
	Commander, Dugway Proving Ground, Dugway, UT 84022
1	ATTN: Technical Library, Technical Information Division
	Commander, Harry Diamond Laboratories, 2800 Powder Mill Road, Adelphi, MD 20783
1	ATTN: Technical Information Office
	Director, Benet Weapons Laboratory, LCWSL, USA AMCCOM, Watervliet, NY 12189
1	ATTN: AMSMC-CB-TL
1	AMSMC-LCB-R
1	AMSMC-LCB-RM
1	AMSMC-LCB-RP
	Commander, U.S. Army Foreign Science and Technology Center, 220 7th Street, N.E., Charlottesville, VA 22901-5396
3	ATTN: AIFRTC, Applied Technologies Branch, Gerald Schlesinger

No. of Copies	To
1	Plastics Technical Evaluation Center, (PLASTEC), ARDEC, Bldg. 355N, Picatinny Arsenal, NJ 07806-5000
	Commander, U.S. Army Aeromedical Research Unit, P.O. Box 577, Fort Rucker, AL 36360
1	ATTN: Technical Library
	Commander, U.S. Army Aviation Systems Command, Aviation Research and Technology Activity, Aviation Applied Technology Directorate, Fort Eustis, VA 23604-5577
1	ATTN: SAVDL-E-MOS
	U.S. Army Aviation Training Library, Fort Rucker, AL 36360
1	ATTN: Building 5906-5907
	Commander, U.S. Army Agency for Aviation Safety, Fort Rucker, AL 36362
1	ATTN: Technical Library
	Commander, USACDC Air Defense Agency, Fort Bliss, TX 79916
1	ATTN: Technical Library
1	Clarke Engineer Scool Library, 3202 Nebraska Ave. North, Ft. Leonard Wood, MO 65473-5000
	Commander, U.S. Army Engineer Waterways Experiment Station, P. O. Box 631, Vicksburg, MS 39180
1	ATTN: Research Center Library
	Commandant, U.S. Army Quartermaster School, Fort Lee, VA 23801
1	ATTN: Quartermaster School Library
	Naval Research Laboratory, Washington, DC 20375
1	ATTN: Code 5830
1	Dr. G. R. Yoder - Code 6384
	Chief of Naval Research, Arlington, VA 22217
1	ATTN: Code 471
1	Edward J. Morrissey, WRDC/MLTE, Wright-Patterson Air Force, Base, OH 45433-6523
	Commander, U.S. Air Force Wright Research & Development Center, Wright-Patterson Air Force Base, OH 45433-6523
1	ATTN: WRDC/MLLP, M. Forney, Jr.
1	WRDC/MLBC, Mr. Stanley Schulman
	NASA - Marshall Space Flight Center, MSFC, AL 35812
1	ATTN: Mr. Paul Schuerer/EH01
	U.S. Department of Commerce, National Institute of Standards and Technology, Gaithersburg, MD 20899
1	ATTN: Stephen M. Hsu, Chief, Ceramics Division, Institute for Materials Science and Engineering
1	Committee on Marine Structures, Marine Board, National Research Council, 2101 Constitution Ave., N.W., Washington, DC 20418
1	Librarian, Materials Sciences Corporation, 930 Harvest Drive, Suite 300, Blue Bell, PA 19422
1	The Charles Stark Draper Laboratory, 68 Albany Street, Cambridge, MA 02139
	Wyman-Gordon Company, Worcester, MA 01601
1	ATTN: Technical Library
	General Dynamics, Convair Aerospace Division, P.O. Box 748, Fort Worth, TX 76101
1	ATTN: Mfg. Engineering Technical Library
	Director, U.S. Army Materials Technology Laboratory, Watertown, MA 02172-0001
2	ATTN: SLCMT-TML
2	Authors

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION -
I-Jong Lin and David A. Bulpett

Technical Report MTL TR 91-36, September 1991, 30 pp-illus-tables, D/A Project: AH 84

AD____________
UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Mass spectra
Isotopes
Computer programs

A new isotope cluster identification program, named Reverse Cluster Search (RCS) is presented in full to assist the mass spectrometrist in the identification of unknowns. Given a clean, complete isotope cluster from a mass spectrum, its empirical formula can be found through the calculations of RCS. RCS was tested on known spectra which fit certain parameters of composition and spectral quality. Correlation between the matches given by RCS and the actual formula was excellent. RCS was written to run on both a DEC VAXStation 3500 and IBM PC compatibles. A companion program is also included which is called Forward Cluster Search (FCS). FCS will calculate theoretical mass spectral isotope clusters from empirical formula data.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION -
I-Jong Lin and David A. Bulpett

Technical Report MTL TR 91-36, September 1991, 30 pp-illus-tables, D/A Project: AH 84

AD____________
UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Mass spectra
Isotopes
Computer programs

A new isotope cluster identification program, named Reverse Cluster Search (RCS) is presented in full to assist the mass spectrometrist in the identification of unknowns. Given a clean, complete isotope cluster from a mass spectrum, its empirical formula can be found through the calculations of RCS. RCS was tested on known spectra which fit certain parameters of composition and spectral quality. Correlation between the matches given by RCS and the actual formula was excellent. RCS was written to run on both a DEC VAXStation 3500 and IBM PC compatibles. A companion program is also included which is called Forward Cluster Search (FCS). FCS will calculate theoretical mass spectral isotope clusters from empirical formula data.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION -
I-Jong Lin and David A. Bulpett

Technical Report MTL TR 91-36, September 1991, 30 pp-illus-tables, D/A Project: AH 84

AD____________
UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Mass spectra
Isotopes
Computer programs

A new isotope cluster identification program, named Reverse Cluster Search (RCS) is presented in full to assist the mass spectrometrist in the identification of unknowns. Given a clean, complete isotope cluster from a mass spectrum, its empirical formula can be found through the calculations of RCS. RCS was tested on known spectra which fit certain parameters of composition and spectral quality. Correlation between the matches given by RCS and the actual formula was excellent. RCS was written to run on both a DEC VAXStation 3500 and IBM PC compatibles. A companion program is also included which is called Forward Cluster Search (FCS). FCS will calculate theoretical mass spectral isotope clusters from empirical formula data.

U.S. Army Materials Technology Laboratory
Watertown, Massachusetts 02172-0001
A PASCAL PROGRAM FOR MASS SPECTRAL ISOTOPE CLUSTER IDENTIFICATION -
I-Jong Lin and David A. Bulpett

Technical Report MTL TR 91-36, September 1991, 30 pp-illus-tables, D/A Project: AH 84

AD____________
UNCLASSIFIED
UNLIMITED DISTRIBUTION

Key Words

Mass spectra
Isotopes
Computer programs

A new isotope cluster identification program, named Reverse Cluster Search (RCS) is presented in full to assist the mass spectrometrist in the identification of unknowns. Given a clean, complete isotope cluster from a mass spectrum, its empirical formula can be found through the calculations of RCS. RCS was tested on known spectra which fit certain parameters of composition and spectral quality. Correlation between the matches given by RCS and the actual formula was excellent. RCS was written to run on both a DEC VAXStation 3500 and IBM PC compatibles. A companion program is also included which is called Forward Cluster Search (FCS). FCS will calculate theoretical mass spectral isotope clusters from empirical formula data.