83-23

File No. UIUDCDS-F-83-910

A PROGRAMMER'S GUIDE FOR CLUSTER
A PROGRAM FOR CONJUNCTIVE CONCEPTUAL CLUSTERING

Brian Biswas

Department of Computer Science
University of Illinois
at Urbana-Champaign

ISG 83-10

July 1983

This work was supported in part by International Business Machines, Inc. under grant IBM KNOWL BASE and in part by the National Science Foundation under grant NSF MCS 82-05166.

ABSTRACT

The following is a detailed recipe for the regeneration of IBM PC CLUSTER from CYBER CLUSTER. Indicated is all processing required to transfer source code, modify the code to fit the compiler, compile the code, and run the program. Also included are the steps required to go from the CYBER to the VAX, and then to the IBM PC. Shown is the text of all commands in the form in which it would be entered by a programmer repeating the above process. An experiment comparing the running times on the IBM PC, VAX, and CYBER computers is included, as is a list of all changes that have been made to the original source code on the CYBER to adopt CLUSTER to the VAX and the IBM PC.

Table of Contents

I INTRODUCTION

This report is a detailed reference guide for using and maintaining CLUSTER on the VAX and IBM PC. Information is provided on transferring, compiling, and running CLUSTER on both machines. The reader interested only in running CLUSTER programs on the IBM PC should first read Section V of this report; the reader interested only in running CLUSTER programs on the VAX should first read Section III; for running CLUSTER/2 on the VAX or IBM PC see section IX.

An experiment testing the running times of CLUSTER on the VAX, IBM PC, and CYBER computers is described in Section VI. The actual input files for this experiment are listed in Section VIII.

An excellent paper on the CLUSTER/2 program (the successor to CLUSTER) is found in "A DESCRIPTION AND USER'S GUIDE FOR CLUSTER/2," by Robert Stepp [4]. The reader who wishes to use the CLUSTER program to its full potential is urged to read that document.

II TRANSFER TO VAX

It is assumed the program currently exists on the CYBER. To transfer to the VAX the following steps are necessary:

First log on the VAX A. Then type the command:

```
acs cyber-log <filename>
```

This will log you onto the CYBER and prepare things for transfer. Finally, type the command:

```
type <filename>
```

where <filename> is the same as listed before. The file will appear on the screen and a copy of it will appear in a file of the same name on the VAX A.

To transfer to the VAX B type the commands:

```
chmod 777
uucp <filename>  uiuccsb\!/tmp/<filename>
```

To transfer to the VAX A type the commands:

```
chmod 777
uucp <filename>  uiucdcs\!/tmp/<filename>
```

To transfer to your directory type:

```
mv /tmp/<filename>  <filename>
```

III COMPILATION ON VAX

Now on the VAX, it is necessary to modify the code to fit the Berkeley Pascal compiler. CLUSTER is such a large program that it is best to use the separate compilation facility provided with the Berkeley Pascal compiler 'pc'. This facility allows programs to be divided into any number of files which are then compiled individually, to be linked together at some later time. Thus, small changes made later in developing CLUSTER will not require time-consuming re-compilation of the entire program.

Two types of files are needed: .h files and .p files. They are compiled into .o files by the Berkeley compiler to be linked together at a later time.

The Berkeley Pascal compiler performs type checking across separately compilable files. Since Pascal type definitions define unique types, any types which are shared between separately compilable files must be the same definition. These definitions are placed in .h files and thus define unique types; all definitions from the same .h file define the same type. The facility also allows the definition of constants, and the declaration of labels, variables, and external functions and procedures. Thus, procedures and functions used between separately compiled files must be declared external in a .h file (and only in a .h file). This .h file must be included (use the INCLUDE command) by any file which calls the function or procedure. These files are included only at the beginning of the file, and thus define or declare global objects.

An example should make this clear:

Let the main program be:

```
program test(input,output);
#include "globals.h"
#include "relationaltables.h"

begin
    { main program }
end.
```

Where the file globals.h looks like:

```
consts :
        { constants }

vars  :
        { variables }
```

And the file relationaltables.h :

```
function semantics(var code: codetype) : tokentype;external;

procedure replace( token : codetype); external;
```

Then function semantics and procedure replace would be in a separably compiled file that looked like :

```
#include "globals.h"
#include "relationaltables.h"

function semantics;
begin
    {semantics code}
end;

procedure replace;
begin
    {replace code}
end;
```

Any number of separately compilable source and definition files is allowed. (There are six source and six definition files in the present CLUSTER version.) However, the separate source files must all have the extension .p, the separate definition files the extension .h.

The various .p files can now be compiled via the Berkeley Pascal compiler with the command:

```
pc -c *.p
```

This will compile all files in the directory with a .p extension. The compiler itself will call up the .h files as needed-- they should not be included in the pc command.

Alternately, one can use a makefile so that only those .p files that need be recompiled are actually recompiled. The structure of the makefile for CLUSTER is as follows:

```
a.out: oper.o prim.o rltb.o ccclib.o subr.o sys.o
     pc *.o

oper.o: globals.h subr.h oper.h sys.h prim.h

ccclib.o: globals.h subr.h sys.h oper.h prim.h rltb.h

rltb.o: globals.h sys.h oper.h prim.h rltb.h

prim.o: globals.h prim.h subr.h sys.h oper.h

subr.o: globals.h subr.h sys.h oper.h prim.h

sys.o : globals.h sys.h
```

(Using a makefile is a more efficient way to compile. Only those files that depend on .h files that have been modified since the last compilation are recompiled.)

Then compile via the command

```
make
```

The compiler has produced a separate .o file for each .p file in the program. Compile these with the command:

```
pc *.o
```

The compiler will produce a file called a.out. This is the compiled CLUSTER program. Run it with the command:

```
a.out <inputfile  >outputfile
```

where inputfile and outputfile are the names of the files from which input and output is taking place.

A separate CLUSTER program exists as a single program. It takes approximately 10-15 minutes to compile, so it is not recommended for developmental work. It resides on the VAX B under the file name cluster.source. Compile via the command:

```
pc cluster.source
```

The compiler will produce an a.out file as before. (It will be slightly smaller, also.) Run it with the command:

```
a.out <inputfile  >outputfile
```

IV TRANSFER TO IBM PC

For CLUSTER to run on the IBM it must be a single program. (The IBM UNITS feature, analogous to the Berkeley separate compilation facility, will not work with CLUSTER because of its complex 'interwoven' structure, see appendix, IBM change eight.) As the editor on the IBM PC only accepts program blocks of size sixty or less, divide the program into chunks of approximately that size (i.e. about 36,000 characters). Immediately after the global initialization section use an INCLUDE statement for each file to be included in the program. Note: during compilation the actual file is written into the program starting at the place the INCLUDE statement is encountered, so the include statements must be in the proper order.

Transfer is accomplished as follows:

Insert the TRANSFER disk into unit 4, the disk to receive the program into unit 5. Boot the IBM PC. (Turn it on.) When the system is ready, type X <cr>. Engage in the following user-system dialogue:

```
User:       X <cr>

System:      execute what file?

U:          tran2
```

The system will respond with buzzes and clicks. When these subside continue as follows:

```
U:          <ctrl>\

S:          Tran: B(reak C(hsp G(et L(ogon I(nfo P(ut Q(uit S(wsp ?

U:          C

S:          change to what speed?
```

```
U:          9600 <cr>

S:          changing speed to 9600

U:          <user's name> <cr>

U:          <user's VAX password> <cr>
```

This will log the user onto the VAX. When the VAX prompt appears continue with:

```
U:          stty 1200 <cr>

U:          <ctrl>\

S:          Tran: B(reak C(hsp G(et L(ogon I(nfo P(ut Q(uit S(wsp ?

U:          C <cr>

S:          change to what speed?

U:          1200 <cr>

U:          stty nl2
```

The user can now transfer files between the VAX and the IBM PC:

```
U:          <ctrl>\

S:          Tran: B(reak C(hsp G(et L(ogon I(nfo P(ut Q(uit S(wsp ?
```

Now type 'G' if you want to transfer from the VAX to the IBM, or P is you want to transfer from the IBM to the VAX. Say the user typed 'G' (the following prompts are reversed if 'P' is typed):

```
S:          file name from VAX?

U:          <filename>
```

S: file name to IBM?

U: #5:<filename>.text

S: initing buffer...

The system prints dots while the file is being transferred. Periodically the message: <buffer flushing... done> will appear. This may be followed by more dots and more buffer flushing messages, depending on how long the file is.

When the transfer is complete, the system will respond with:

S: transfer complete

The user is still logged onto the VAX. To transfer more files type:

U: <ctrl>\

This will get you the prompt. Proceed as before, uploading or downloading files. When you are done, log off the VAX, then type:

U: control-d

U: <ctrl>\

S: Tran: B(reak C(hsp G(et L(ogon I(info P(ut Q(uit S(wsp ?

U: Q

Notes: The transfer program behaves erratically. Expect difficulty in setting up communications between the VAX and IBM. Also, the programs to be transferred must be on the VAX A; no communication link currently exists between the VAX B and the IBM. For reference purposes, a 15,000 character file takes about five minutes to transfer.

Rx and tx are two files which MUST reside on the user's VAX A directory (the same directory on which the files to be transferred reside). They perform the actual transfer.

V RUNNING CLUSTER ON THE IBM PC

To compile the program use the USCD 4-word compiler and system:4 operating system. Boot the IBM PC (system:4 and compiler on one disk in unit4, the source program in unit5).

Compile the program as follows:

```
U:          C

S:          compile what file?

U:          #5:CLUSTER

S:          to where?

U:          #5:CLUSTER

S:          compiling...
```

Compilation takes about five minutes. When done, the compiled file will be in a file called CLUSTER.CODE. Run this via the commands :

```
U:          X

S:          execute what file?

U:          #5:CLUSTER

S:          executing...
```

Input is from a file called test.text. It must reside on unit5. Output is normally to the terminal. It can be redirected via the command :

```
U:          X
```

```
S:          execute what file?

U:          #5:CLUSTER o=<outputfile>
```

Output will appear on both the terminal and in the designated output file.

VI COMPARISON OF RUNNING TIMES

Three experiments were run on the CYBER, VAX, and IBM PC. The first, FACES, contained four variables and eight events. The second, DIAMONDS, contained two variables, and sixteen events. The third, FAMILY2, contained 5 variables and 8 events. Running times (in seconds) for each experiment were as follows :

	Experiment		
Machine	FACES	DIAMONDS	FAMILY2
CYBER	90.0	70.0	70.2
VAX	30.0	15.0	15.0
IBM PC	660.0	420.0	600.0

The above times were recorded with a watch. They indicate the time taken to execute and print the results on the CRT. More accurate times were recorded for the CYBER and VAX versions using the computer's internal clock. They indicate only the time taken to execute :

	Experiment		
Machine	FACES	DIAMONDS	FAMILY2
CYBER	2.22	1.79	2.18
VAX	7.97	4.70	5.75

As the above results clearly show, the CYBER version is the fastest, followed by the VAX; the IBM PC version is a distant third.

An important point to note, however, is that although the CYBER version is roughly three times as fast as the VAX version, the response time, as noted by the user, is the other way around, i.e. the VAX version is roughly three times as fast as the CYBER version as far as the user receiving his output is concerned. This fact is do to the greater system load on the CYBER. The IBM PC, being a personal computer, is unaffected by this problem.

VII VERSION DIFFERENCES

The following differences exist between the CYBER and VAX versions of CLUSTER:

1) Berkeley Pascal does not allow you to write pointers to a text file. Thus, statements of the form:

```
writeln(<ptr> :6oct) were changed to
writeln( ord(<ptr>))
```

2) Berkeley Pascal does not allow the 'then' part of an 'if then else' statement to be null. This was fixed by negating the condition of the 'if' part and dropping the 'else'.

```
ex. If A then else go to 999
       was rewritten as
     If not A then go to 999
```

3) Type conflict of operands occurred in procedure prtmap and in function maplow. Both call procedure prtset with an argument of type evtmapunit, where an argument of type selector is expected. Both types are themselves defined to be 'set of refval', but standard Pascal requires the names to be identical (since only they and not the underlying definitions are looked at). Function switch was written to take a parameter of type evtmapunit and return the parameter's value, but of type selector. A variant record was used to accomplish this.

4) Procedure copyplx was rewritten. It used knowledge about the number of CYBER words actually used by the variants of the record 'complex', using its own definition of a complex; now it simply copies the fields of the record 'complex' into a new complex of the same type.

5) Function getcplx was rewritten. It used knowledge of memory usage to gage the amount of free space available (to get the best size for the next complex). This was dispensed with. Now getcplx generates a new complex of standard size.

6) Set notation is different. In Berkeley Pascal the relationals '<' and '>' indicate proper set inclusion. But on the CYBER, 'a<b' means 'not (a >= b)', and 'a>b' means 'not (a <=b)'. Consider the example where a = [0,2] and b = [1]. 'a<b' on CYBER would evaluate to true, but in Berkeley Pascal would evaluate to false. The notation was changed to fit the standard Pascal definition: only '<=', '>=', and '<>' are used.

7) The value of the variable 'typecodes' was changed, since VAX uses ASCII whereas the CYBER uses its own coding scheme. (Typecodes is used to denote the character classes--numeric, character, filler, quote, or missing data--of the input characters.) The setting of this variable was simply rearranged.

8) The name of procedure main was changed to mymain, since

the assembler considered main a reserved word.

9) The program itself exists in two versions on the VAX. In one version it has been broken into six separately compilable .o files; this is the separate compilation facility provided on the VAX. (As an aside, this resulted in a savings of 6,000 characters over the CYBER version.) The second version consists of a single segment. Both versions are useful: the separately compilable version is best for program development (only the .o files that are changed need be recompiled); the single segment version is best for use by others--when the two versions are compiled the single segment version is 40,000 characters less than the separately compilable version. Input to both versions is expected to come from the terminal, but can be redirected.

In addition to the above, the following differences exist between the VAX and the IBM PC (using UCSD Pascal) versions of CLUSTER:

1) Only the first eight characters of a variable are recognized on the IBM. This necessitated the following changes in variable names:

VAX	IBM
covertype	covtype
selectors	selectrs
structure	struct
variables	variabls

initmethod initmhd

2) Functions linelimit and clock do not exist in UCSD Pascal; they were removed from the program.

3) No pre-defined function to find the cardinality of a set exists in UCSD Pascal. Function 'card' was added to the program to calculate the cardinality of a set.

4) The IBM version expects input to come from a file named test.text.

5) UCSD Pascal does not allow functions to return sets. Therefore, function Switch was rewritten as procedure Switch, and appropriate changes made in the calling statements.

6) UCSD Pascal does not allow octal numbers to be written directly. Therefore, in procedure Setup the number 77o was changed to 63, and the number 777o to 511.

7) On the VAX, array typecodes, in procedure Setup, was initialized by assigning a 128 character string to it. In UCSD Pascal no string longer than 64 characters can be assigned to a variable. Initialization of typecodes now takes place a character at a time using a loop.

8) The IBM PC version is a single program; forward declarations were used so that various procedures and functions would not need be moved from their original place. (Thus, the program is still modular.) The possibility of

using UNITS on the IBM was seriously considered, but the following problem would have resulted: With several of the original modules, e.g. A calls B which in turn calls A. Units which mutually call each other are not allowed in UCSD Pascal. It would have been necessary to copy the code A needed from B into A. (Include statements could not have been used since they are not allowed with UNITS.) The resulting UNITized program, it was calculated, would have used thirty more blocks than a version written as one large program. Considering the limited amount of space on the IBM PC, the latter alternative was adopted. As the editor on the IBM accepts only blocks of size thirty-six or less, the program was divided into seven 'chunks' of approximately thirty-two blocks each.

9) The IBM PC version needs 192K to execute. The compiled file is fifty-three blocks long. To improve efficiency the program is segmented. (This allows the operating system to swap code into and out of memory to save space when executing.) As execution is still quite slow, we are currently looking at ways to improve efficiency.

VIII SAMPLE INPUT FILES

The following input files were used in Section VI for the running times experiment (see diagrams on pages 26-27):

FACES

```
1 'faces'

parameters
  k  debug criterion
  2  2    one
  2  2    two

one-criterion
  #  criterion tolerance
  1   1        0.0

two-criterion
  #  criterion tolerance
  1   2        0.2
  2   1        0.0

variables
   #  name  type  levels
   1  cir   lin    3
   2  ovl   lin    3
   3  tri   lin    4
   4  squ   lin    3

events
   #  cir  ovl  tri  squ
```

```
1  2  2  2  0

2  1  2  3  2

3  2  2  2  1

4  1  2  1  1

5  2  0  3  0

6  2  2  3  1

7  2  2  1  2

8  2  2  0  1

      DIAMONDS

1 'diamond clustering'

parameters

  k  debug criterion

  2  2   one

  2  2   two

one-criterion

 #  criterion tolerance

 1   1        0.0

two-criterion

 #  criterion tolerance

 1   2        0.2

 2   1        0.0

variables

   #  name  type  levels

   1  cir   lin    10

   2  ovl   lin    5
```

events

#	cir	ovl
1	0	2
2	1	1
3	1	3
4	2	0
5	2	4
6	3	1
7	3	3
8	4	2
9	5	2
10	6	1
11	6	3
12	7	0
13	7	4
14	8	1
15	8	3
16	9	2

FAMILY2

1 'family2'

parameters

k	debug	criterion
2	2	one

one-criterion

#	criterion	tolerance
1	1	0.3

2 2 0.0

variables

#	name	type	levels
1	cir	lin	2
2	ovl	lin	2
3	tri	lin	2
4	squ	lin	2
5	sum	lin	4

events

#	cir	ovl	tri	squ	sum
1	0	0	0	1	1
2	0	0	1	0	1
3	0	1	0	0	1
4	1	0	0	0	1
5	1	0	1	1	3
6	1	1	0	1	3
7	1	1	1	0	3
8	0	1	1	1	3

LOGICAL DIAGRAMS FOR INPUT FILES

(Xi's refer to variable numbers in corresponding variable tables.)

```
x1 x2
----------------------------------------
   0|  |  |  |  |  |  |  |  |  |  |  |  |
  ----------------------------------------
 0 1|  |  |  |  |  |  |  |  |  |  |  |  |
  ----------------------------------------
   2|  |  |  |  |  |  |  |  |  |  |  |  |
----------------------------------------
   0|  |  |  |  |  |  |  |  |  |  |  |  |
  ----------------------------------------
 1 1|  |  |  |  |  |  |  |  |  |  |  |  |
  ----------------------------------------
   2|  |  |  | X|  |  |  |  |  |  |  | X|
----------------------------------------
   0|  |  |  |  |  |  |  |  |  | X|  |  |
  ----------------------------------------
 2 1|  |  |  |  |  |  |  |  |  |  |  |  |
  ----------------------------------------
   2|  | X|  |  | X| X| X|  |  | X|  |  |
----------------------------------------
    | 0| 1| 2| 0| 1| 2| 0| 1| 2| 0| 1| 2|  x4
    |    0   |    1   |    2   |    3   |  x3
```

FACES

```
x2
--------------------------------------
5|  |  |  |  |  |  |  |  |  |  |  |
--------------------------------------
4|  |  | X|  |  |  |  | X|  |  |  |
--------------------------------------
3|  | X|  | X|  |  | X|  | X|  |  |
--------------------------------------
2| X|  |  |  | X| X|  |  |  | X|  |
--------------------------------------
1|  | X|  | X|  |  | X|  | X|  |  |
--------------------------------------
0|  |  | X|  |  |  |  | X|  |  |  |
--------------------------------------
 | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|     x1
```

DIAMONDS

```
x1 x2 x3
-----------------------------------
         0|  |  |  |  |  |X|  |  |
           --------------------------
       0 1|  |X|  |  |  |  |  |  |
 0         --------------------------
         0|  |X|  |  |  |  |  |  |
           --------------------------
       1 1|  |  |  |  |  |  |  |X|
-----------------------------------
         0|  |X|  |  |  |  |  |  |
           --------------------------
       0 1|  |  |  |  |  |  |  |X|
 1         --------------------------
         0|  |  |  |  |  |  |  |X|
           --------------------------
       1 1|  |  |  |X|  |  |  |  |
-----------------------------------
          | 0| 1| 2| 3| 0| 1| 2| 3|        x5
          |     0     |     1     |        x4

           - FAMILY2
```

IX CLUSTER/2

CLUSTER/2 is the successor to CLUSTER. It allows one to use structured variables, is able to combine duplicate events, and uses weighted events and variables, among other things. The reader should see [4] for a full description of CLUSTER/2.

CLUSTER/2 exists on the VAX as a single program. No separately compiled version exists. Compile it via the command:

```
pc cluster/2.source
```

The compiler will produce a file named a.out. Run it with the command:

```
a.out <inputfile  >outputfile
```

CLUSTER/2 exists on the IBM PC as a single program. It is divided into five files so it can be easily edited. Input is from a file called cinput.text. It must reside on unit5.

Compilation of CLUSTER/2 takes about fifteen minutes. Due to the large size of the compiled file (132 blocks) it will not fit on the source disk. Compilation is accomplished by using a third disk and interchanging it with the disk in unit4 (the operating system and compiler disk). To compile the program use the UCSD 4-word compiler and system:4 operating system. Boot the IBM PC (system:4 and compiler on one disk in unit4, the source program in unit5). Compile the program as follows:

```
U:      C

S:      compile what file?

U:      #5:CLUSTER/2

S:      to where?

U:      #4:CLUSTER/2
```

Now put a blank disk in unit4 and press "return". The system will respond with:

```
S:      compiling...
```

Periodically the system will ask for the operating system disk to be placed back in unit4, then for the (originally) blank disk. Do as the system asks. When done, the compiled file will be in a file called CLUSTER/2.CODE. Put the disk containing CLUSTER/2.CODE into unit5 and the disk containing the operating system in unit4. Run via the commands:

```
U:    X

S:    execute what file?

U:    #5:CLUSTER/2

S:    executing...
```

X REFERENCES

[1] Michalski, R. S. and Stepp, R., "An application of AI techniques to structuring objects into an optimal conceptual hierarchy," *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, August 24-28, pp. 460-465, 1981.

[2] Michalski, R. S. and Stepp, R. E. "AUTOMATED CONSTRUCTION OF CLASSIFICATIONS: Conceptual Clustering versus Numerical Taxonomy," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 4, July 1983, pp. 396-410.

[3] Michalski, R. S., "KNOWLEDGE ACQUISITION THROUGH CONCEPTUAL CLUSTERING: A theoretical framework and an algorithm for partitioning data into conjunctive concepts," A Special Issue on Knowledge Acquisition and Induction, *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 3, pp. 219-144, 1980.

[4] Stepp, R. E., "A Description and User's Guide for CLUSTER/2: A Program for Conjunctive Conceptual Clustering," UIUCDCS-R-83-1084, Department of Computer Science, University of Illinois at Urbana-Champaign, 1983.

ACKNOWLEDGMENTS

The author expresses his thanks for the support and encouragement of Bob Stepp without whom this report would never have been completed. He would also like to thank Pravin Vaidya for proofreading. Partial support of this work was provided by International Business Machines, Inc. and the National Science Foundation which is gratefully acknowledged.