International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

LPS/11 Users Manual

Orchard-Hays, W.

IIASA Research Memorandum
December 1978

LPS/11 USERS MANUAL

Wm Orchard-Hays

December 1978

PREFACE

The set of programs described in this paper were originated partly as a result of frustration. In 1976, the writer found that although access to large mathematical programming systems (MPS) was possible at IIASA, it was awkward, time-consuming and expensive. Furthermore, most of the linear programming (LP) models used at the time were for small pilot studies which did not require the power of the large MPSs.

Although a PDP 11/45 was in constant use in-house, operating under the excellent UNIX system (developed at Bell Telephone Laboratories) no LP system was available. In summer 1976 the writer was faced with the necessity of analysing a small LP energy model which existed only in the form of a deck of punched cards in bad condition. To facilitate the analysis, he wrote a FORTRAN program to read the deck (which was in standard MPS input format), check for obvious errors, and produce a tabular listing (tableau) of the model. Satisfied that the model was now in quite good shape, the next step was to solve some cases. It was quickly realized that the model-analysis program was, in essence, a CONVERT procedure very similar to those in MPSs, and it was decided to proceed with writing a SETUP procedure and the simplex algorithm. Having designed and implemented many MPSs, it was only a matter of a couple of weeks work to get an acceptable, small package of programs functioning.

Although large MPSs are now more conveniently accessible and many LP models at IIASA are quite large, the small package on the PDP has proven very useful and has been in almost constant use. It has been refined and enhanced into a reliable, robust set of programs reported in this paper.

SUMMARY

A package of programs for solving small linear programming (LP) problems on a PDP-11 computer is described. Although the paper constitutes a users manual, the style is conversational. The paper is not a tutorial on LP or on mathematical programming systems (MPS) in general. The reader should be familiar with standard MPS input formats (originally developed for IBM's MPS/360 and now a de facto industry standard), and have had some experience in solving LP problems on a computer.

The package consists of eight executable programs, equivalent to corresponding procedures in an MPS, and includes both the primal and dual algorithms and two parametric algorithms. The programs are written entirely in FORTRAN IV (the PDP dialect) and are designed for all in-core operation during solution of the model case. The size of models is arbitrarily limited to 100 rows and 250 columns to make this mode of operation possible on a relatively small computer.

The eight programs are quite similar to the corresponding procedures in large MPSs, both in construction and operation, and give essentially identical results. Output formats are also very similar to those of standard MPSs. (Readers unfamiliar with the algorithms actually used in computer systems will find considerable discussion in the IIASA publication "Some Additional Views on the Simplex Method and the Geometry of Constraint Space", RR-76-3, May 1976, by the same writer.)

It is possible to modify the programs for larger models and to transfer them to other computers but certain internal changes will be required. Some of these considerations are discussed in the final section.

This paper was produced from computerized text formatted by the NROFF program under UNIX on a PDP 11/70 with Diablo terminal.

LPS/11 LINEAR PROGRAMMING SYSTEM

CONTENTS

LPS/11 LINEAR PROGRAMMING SYSTEM

Wm. Orchard-Hays

International Institute for Applied Systems Analysis

INTRODUCTION AND GENERAL CHARACTERISTICS

LPS/11 is a linear programming system available on the PDP 11/70 under the UNIX operating system. It exists as eight executable programs which are coordinated in the same sense as procedures in mathematical programming systems (MPSs) such MPSX, APEX, SESAME, etc. However, there is no in-core residency between programs and no control language in an MPS sense. The first is handled by means of scratch files and the second by standard UNIX command conventions and some specialized interactive responses.

LPS/11 is written entirely in FORTRAN IV and consequently is less efficient than a large MPS. Nevertheless, its execution speed is acceptable for the size problems for which it is suitable. The programs operate entirely in-core and hence problem size has been arbitrarily limited to the following:

100 rows maximum

250 columns maximum

Approximately 1200 nonzero elements of all kinds including all coefficients, right-hand-side (RHS) elements, range values and bound specifications. Each bound specification counts for two values, regardless of type.

The last limit depends on available core and could be easily in-

creased for a larger machine. The limit on number of rows and columns could likewise be increased with a minimum amount of change in declaration statements. To make the programs completely general with respect to size, however, would require the installation of intermediate I/O which would be a sizeable undertaking.

The eight executable programs are for the following functions:

Input conversion (LPINPUT)
Case setup (LPSETUP)
Main solution algorithm, primal and dual (LPSOLVE)
Parametric objective function algorithm (PARAOBJ)
Parametric RHS algorithm (PARARHS)
Auxiliary solution procedure (LPSOLN)
Initial crash algorithm (LPCRASH)
LP variable values to unformatted file (LPXVARS)

The LPSOLN program is only a convenience for recreating the solution output from a previously obtained basis. It may also be used to produce an iteration 0 solution (all-logical or starting basis) which is sometimes desired.

The system uses the following preempted file names which are explained where appropriate:

LPTEMP, LPBASIS, LPOUT, LPVAR

If it is desired to save one or more of these, they can be renamed. However, all file names are restricted to a maximum length of 7 characters. LPBASIS cannot be used under another name except by LPSETUP (see below).

The internal structure of LPS/11 is very similar to the basic procedures and algorithms of MPS/360 and the SESAME system, a large interactive MPS developed by the author. However, no partial or multiple pricing is included. (Each iteration makes a full pass of the matrix.) The inclusion of such features would

have increased core requirements both due to additional program logic and for expanded vector regions.

MODEL INPUT

The LP model is input from a file in MPS/360 CONVERT format, prepared in any appropriate way.* The following exceptions must be observed:

1. The first card may have either of two formats: either a standard NAME card (NAME in columns 1-4 and the name in 15-21) or merely the name (not 'NAME') in columns 1-7. (The name is a file name and hence limited to 7 characters.) However, in either case, columns 9-16 are meaningful and used as follows:

   LIST Only a tabular listing of the model is produced.
   NOLIST A model file is produced but no tabular listing.
   (other) A model file and tabular listing are produced.

   For most cases, NOLIST should be specified.

2. The Dx option for rows is not supported and may not be used.

3. MARKER columns and SCALE values are not recognized and may not be used.

4. There is a limit of 4 RHSs, 4 RANGES columns, and 4 BOUNDS sets.

The tabular listing is a tableau-format listing of the model, untransformed. It is useful for small models but, for larger ones,

---

* The DATAGEN programs are specifically designed for matrix generation. See IIASA Software Series No. 4: TABGEN/MATGEN. (DATAGEN is a single substitute for both these original programs.)

it produces too much paper output.

The input conversion program is invoked with the following UNIX command:

LPINPUT 5=source 6=+output

"source" is the name of the input file in MPS/360 format. The model name in the first card must be different from "source" and from any of the preempted file names.

"output" is a print file name such as OUTPUT or any unique file name. If NOLIST is specified, this specification may be omitted and the statistics summary (plus any error messages) will appear at the console.

LPINPUT reads the MPS/360 input, produces a full tabular listing (10 columns per strip of pages) of all columns, RHSs and RANGES vectors, and BOUNDS in input format. Row types are shown with the RHS. The listing goes to "output". (All this is omitted if NOLIST is specified.) A summary of statistics appears at the end, including the number of errors (also commented where detected) and the number of zero elements specified. Any such elements should be deleted from "source" file to save space and time in subsequent programs, and LPINPUT rerun with the corrected file.

Successful completion of LPINPUT is indicated by

STOP -- 30

on the terminal. The model statistics are always produced.

CASE SETUP

A model file is not usable directly by the algorithmic programs although it is read by them for identification and output purposes. (The row and column names exist only on the model

file.) The program LPSETUP is invoked to set up a model for solution. Its input file is the model file and it produces both LPTEMP and LPBASIS unless a basis is specified in which case it reads an old LPBASIS file (which may have been renamed) rather than producing one. Only one LPTEMP can exist at a time.

LPSETUP is interactive. It first asks for your model name. An incorrect name results in a "failure to open file" error. You must then restart.

When a correct name is given, LPSETUP prints the statistics produced by LPINPUT and asks "ok? (y/n)". A "y" response continues. You are then asked for the row number of the functional (obj row) which must be typed in positions 1-3. The row names are not available at this stage which is why the functional must be specified by row number.

LPSETUP checks that the row number exists. If not, you are given the message

wrong! try again

It later checks that the row is type N (free) and, if not, aborts the run after a message.

The next request is for a scale value for the functional. For standard minimization, just return a null line. To maximize, enter -1.0 or some negative value. A magnitude other than unity has the same effect as the scale factor in most MPSs. (Less than 1.0 terminates sooner, greater than 1.0 drives optimization further.)

All RHS names (up to 4) are then displayed and you are requested to specify one. An incorrect name results in the same response as for a functional row number out of range.

If there are RANGES or BOUNDS the same procedure is followed as for RHS except that a null or incorrect response is interpreted to mean that no RANGES or BOUNDS set is desired for this run.

Next, you are asked whether an old basis is to be input.

This is an LPBASIS file produced by a previous run. The dimensions of the model must be the same or LPSETUP aborts. If no basis is to be loaded, return a null line. LPSETUP then writes the all-logical basis to file LPBASIS.

You are then asked to specify log-line frequency. Only numbers 1 through 25 are accepted. A zero or null response is interpreted as 1, a large number as 25.

If you think the dual algorithm will be more efficient, answer "y" to the next request. Any other response causes the primal algorithm to be used. The dual algorithm is terminated (or not started) in any of the following circumstances:

(a) No primal infeasibilities exist initially.

(b) Dual infeasibilities exist initially.

(c) Dual infeasibilities are created (This can only occur due to numerical tolerance difficulties, which should be very rare.)

(d) All primal infeasibilities have been removed.

In case (d), one primal pass is made which should indicate an optimal solution. If no primally feasible solution exists, the usual "NO FEASIBLE SOLUTION" termination occurs.

If no dual infeasibilities exist initially, it has been found more efficient in all cases tried so far to specify the dual algorithm. Since a full pricing pass is made each iteration anyway, the primal efficiencies obtained in large MPSs are not achieved; the dual algorithm is then more efficient in most cases.

LPSETUP writes a file with the name LPTEMP, only one of which may exist at a time. This is more or less equivalent to the scratch MATRIX file in MPS/360 and derivative systems. This file is used by all other programs. However, if errors are detected by LPSETUP, either this fact is recorded in LPTEMP so that other programs refuse to execute, or no LPTEMP is produced

at all.

Successful completion of LPSETUP is indicated by

STOP --1

at the terminal.

MAIN SOLUTION ALGORITHM

The program LPSOLVE is the main solution algorithm (primal and dual). Its only inputs are the LPTEMP file and the model file from which LPTEMP was created by LPSETUP. However, it takes two arguments: BASIS and TOLCHK in that order. (To use the second without the first, put a dummy word for the first.) Their use is explained below. At termination, LPSOLVE produces a solution output and writes or rewrites the final basis to file LPBASIS. All printed output is written to file LPOUT.

The BASIS argument causes a basis to be loaded from file LPBASIS, initially. This is often more convenient than doing it in LPSETUP but no renaming is permitted.

The algorithm is the revised simplex method with product form of inverse, both primal and dual forms. The program reads the latest existing LPTEMP and starts to iterate. If a basis was specified to LPSETUP or by the argument BASIS, the subroutine INVERT is executed first. This is a triangularization scheme followed by a Markowitz-like selection scheme for the non-triangularizable columns (the "bump" in Hellerman-Rarick terminology).

The usual messages and loglines are produced. All iterating output (except a few fatal error messages) is written both to the terminal and to LPOUT. However, final output is written only to LPOUT. This file should be printed with the ASA filter for full page control. Output format is sufficiently close to MPS/360, et

al, to require no explanation. (The column of "row values" and the column of "input costs" are omitted. However, the status of a row refers to the constraint rather than the slack value. Thus a row shown at upper limit (UL) means that the row value has attained its highest value, not the slack.) There is one major deviation from standard practice, however. GE rows are converted to LE rows internally (as in almost all MPSs) but the inverse transformation is not done at output. The lower and upper limits and other values are shown correctly for LE format but the user may have to mentally invert the sense of his original GE constraints.

A complete primal and dual check is made whenever a primal feasible solution is obtained (after an infeasible one), after execution of INVERT, and at all terminations. The tolerance for both primal and dual errors is 10^{-4}. An error larger than this in magnitude causes an error message showing the largest primal and/or dual error, the constraint or variable on which it occurred (denoted by J-number defined below), and a logline with an appropriate message. INVERT is then called and the check made again. If errors persist, the iterative procedure is terminated and a full solution output is produced. However, an error after an initial INVERT causes an immediate stop with

```
STOP -- 399
```

displayed at the terminal.

If it is necessary to use a larger tolerance to complete solution of a model which goes through ill-conditioned bases, the argument TOLCHK can be used. This causes a prompt at the terminal at which time the user types an appropriate tolerance (real number).

The full set of tolerances used are as follows:

Primal infeasibility	10^{-6}
Dual infeasibility	10^{-5}
Smallest pivot (except INVERT)	10^{-7}
Threshhold magnitude	10^{-15}
Maximum magnitude ("infinity")	10^{30}
Check error tolerance	10^{-4}

None of these except the last is settable by the user as presently implemented. They are defined in LPSETUP and, since this program is relatively small, additional interactive logic could be added to make the tolerances settable. However, long experience has shown that only the check tolerance should be changed by most users since the effects of the other ones are intricate and subtle.

In addition to the uses of INVERT described above, the subroutine is called whenever core available for the eta file (product form of inverse) fills up. This will be indicated by a minus sign for length of eta file in the logline which is always output in this case. There is no invert frequency as in most MPSs. (If core fills up in INVERT itself, STOP -- 390 occurs.)

The normal terminations possible are:

NO FEASIBLE SOLUTION.
UNBOUNDED SOLUTION.
OPTIMAL SOLUTION....
INDICATED TERMINATION.

The last is used when repeating check errors occur at one of the other terminations. A logline and caption is output when the solution goes primal feasible but no stop occurs. If this happens more than once, it indicates that a feasible solution has gone slightly infeasible due to roundoff error. This should happen only rarely and the amount of infeasibility should be insignificant. Phase switching is automatic, in either direction.

Operation of LPSOLVE is automatic except that every 50 iterations the message

50 iters. continue? (y/n)

is displayed followed by a pause for terminal input. A "y" response continues; any other causes a termination with full checking and output, including a valid LPBASIS file.

An explanatory header is produced every 25 loglines, approximately often enough that it is usually on the terminal scope. It is fairly self-explanatory but is detailed here:

ITER iteration number. Always starts from 0.

JIN sequence number of incoming variable, i.e. the J-number. Logical variables are numbered 1 to m, structural variables m+1 to m+n.

JOUT sequence number of outgoing variable. If JOUT=JIN, the variable changed bound. Variables coming in from or going out to upper limit are indicated by an adjoining U.

DJ pricing derivative for JIN (reflects phase pricing form).

VALUE sum of infeasibilities in phase 1, value of functional in phase 2. (Latter reflects value of SCALE.)

PINF number of primal infeasibilities.

DINF number of dual infeasibilities (reflects phase).

NETA number of eta columns.

LETA length of (in-core) eta file.

(A few debugging stops have been left in the routines which can only occur in "impossible" situations. If any of these actually occur, as much information as possible should be forwarded to the author.)

LPSOLN

This program simply inverts the basis currently existing on LPBASIS and produces a solution output. If no basis was loaded in LPSETUP or produced afterward, the output reflects the all-logical basis ("iteration 0" solution).

PARAMETRIC OBJECTIVE ALGORITHM (PARAOBJ program)

In order to use this algorithm, the model must contain at least two type N (free) rows. One (say OBJ) is used as the objective function and an optimal solution is obtained with LPSOLVE in the normal way. LPSOLVE must have written an optimal basis to file LPBASIS before PARAOBJ can be used.

Another type N row (say COBJ) is then used as a change row to OBJ. The functional is progressively modified to

$$OBJ + \emptyset\ COBJ, \quad \emptyset \geq 0$$

for increasing values of $\emptyset$, starting from $\emptyset = 0.0$. Both a limit value for $\emptyset$ and an incremental value must be specified. If the incremental value is zero or greater than the maximum value, it has no effect. Otherwise, a solution output is produced when the incremental value is attainable (since the last time).

The specification of the change row and the $\emptyset$-values is done interactively when PARAOBJ starts to execute. It first displays the message

```
no. of delta obj row
```

and pauses for the user to type in an integer, just as for the obj row in LPSETUP. If this row number does not exist or is the same as the obj row, an appropriate message is displayed followed by

```
STOP -- 399
```

The same occurs if the row specified is not type N. In any of these cases, PARAOBJ must be restarted.

PARAOBJ then reads the file LPBASIS and if the basis read is for a model of different dimensions, a message is displayed followed by the same stop as above. No other checking for a proper basis is possible at this point but an incorrect basis will (with almost probability of 1.0) cause termination when the solution is found not to be optimal.

If all appears OK, the following message is displayed

type initial and final para values (real, 2 lines)

Unless PARAOBJ itself is being restarted (see below), the first value is 0.0 followed by a carriage return (end of line) and the second the upper limit which Ø is to attain, often 1.0. Next the message

type delta value for solns

is displayed to prompt the user to type in the increment. Thus, for example, responding with

0.0
1.0

.25

to the previous two displays means to drive Ø to 1.0 and produce a solution at .25, .50, .75, and 1.0. If it is not possible to drive Ø to the limit specified, a solution is produced at the largest possible value preceded by a logline with the caption

PARAMETER AT ABS. MAX

Conversely, if the specified limit has not yet been reached but it is discovered that no maximum exists, the algorithm will terminate with the caption

PARAMETER UNBOUNDED

but the specified limit will be honored. In other cases, one of

the captions

INCREMENT ON PARAMETER

or

PARAMETER AT LIMIT

will be used.

A full check is performed before each output. If check errors occur, the caption

(INDICATED OUTPUT)

is used followed by execution of INVERT. If errors persist, an appropriate message is displayed, a full output is produced and the procedure terminates. The use of INVERT, handling of errors, loglines and interruption (at 50 iterations) are exactly the same as in LPSOLVE except that the DJ value on loglines is replaced by the value of Ø, denoted by PARAMETER.

PARAMETRIC RHS ALGORITHM (PARARHS program)

In order to use this algorithm, the model must include at least two RHSs. One (say BRHS) is used as the base RHS and an optimal solution is obtained with LPSOLVE in the normal way. LPSOLVE must have written an optimal basis to file LPBASIS before PARARHS can be used.

Another RHS (say CRHS) is then used as a change column to BRHS. The RHS is progressively modified to

$$BRHS + \theta\ CRHS, \quad \theta \geq 0$$

for increasing values of θ, starting from $\theta = 0.0$. Both a limit value for θ and an incremental value must be specified. The usage, formats and action of PARARHS are otherwise exactly like those for PARAOBJ except that the change RHS is specified by name and must be different from the base RHS.

MULTIPLE RUNS OF PARAOBJ AND PARARHS

Either of the parametric algorithms can be restarted from the value of the parameter previously attained. It is only necessary to type in the old terminal value as the new initial value and a higher value for the new terminal value (assuming it was not at an absolute maximum before). The LPBASIS file produced by the prior run must still be available.

On the other hand, PARAOBJ and PARARHS cannot be run end to end. LPSOLVE does not recognize Ø or Θ, PARAOBJ does not recognize Θ, and PARARHS does not recognize Ø. However, the following sequence is possible. Let there be three RHSs, say RHS1, CRHS, RHS2 such that RHS1 + CRHS = RHS2. Set up for RHS1 and obtain an optimal solution with LPSOLVE. Then run PARARHS for CRHS and the limit of Θ specified as 1.0. Now set up again for RHS2. (The basis from PARARHS must be specified.) Now PARAOBJ can be run starting from the optimal RHS2 (with, of course, some change row to the objective). Various combinations of this technique can obviously be formulated.

Note that if the PARARHS basis is not specified (as LPBASIS), LPSETUP will write the all-logical basis to file LPBASIS, destroying the basis previously obtained. If bases are to be used at a later time, the LPBASIS file should be renamed and then specified to LPSETUP by the new name at the appropriate time. Note that every termination of every algorithm (except LPSOLN) writes on LPBASIS.

LPCRASH

This program may only be run immediately after LPSETUP with no loaded basis. It produces a (somewhat) advanced basis and writes it to LPBASIS. Loglines are produced and the solution is checked but no LP output is written. It should be followed by LPSOLVE with the BASIS argument.

LPCRASH is not highly effective for small models. It often takes as much time as starting LPSOLVE from scratch. The user should try it and see if it provides any benefit for his models.

LPXVARS

This program extracts LP results and writes them to an unformatted file called LPVAR. This file can be read with a FORTRAN program for downstream use, such as report writing.

LPXVARS only obtains values associated with main LP variables (columns), not logical or slack variables (rows). It requires LPTEMP, the model file from which it was produced, and a valid LPBASIS (no renaming).

The program requires two arguments. The second must be properly specified or no output will be produced. The first is optional; if not desired, use a dummy such as "x". This will cause all variable names selected by the second argument to be output along with the primal value, as two 8-byte fields. The second argument is a standard mask, enclosed in quotes. For example, 'abc*****' selects all variables whose names start with 'abc'. The length of the mask is significant; thus '****10' selects all names with '10' is positions 5 and 6 and which are blank in positions 7 and 8.

The first argument is used to extract more information for

each variable selected and to arrange these values. Three additional values are possible, denoted by L for lower limit, U for upper limit, and D for dual value. These characters are written together with no spaces between. Thus the command

```
LPXVARS LUD '********'
```

selects all variables and outputs the name, primal value, lower and upper limits and the dual value, in that order. Each field is 8-bytes. However, the order of L, U and D are significant. They may be permuted in any way and the fields output will be permuted in the same way. The first two fields (name and primal value) are fixed and always output first in that order. Thus

```
LPXVARS DU  '********'
```

outputs four 8-byte fields for each variable in the order

```
name, primal value, dual value, upper limit
```

The unformatted file LPVAR is continuous and must be read with appropriate FORTRAN FORMAT statements. The file ends with a dummy variable whose name is 8 blanks and all values zero. The number of values is the same as for all other variables, as per the first argument.

OTHER POSSIBILITIES

Several experimental procedures have been produced fairly easily, most of them for special purposes. An experimental MIP algorithm exists but is not generally available since its performance is erratic. If you have special requirements, contact the author; it may be quite feasible to accomodate them with a minimum of effort.

CONSIDERATIONS FOR EXTENDABILITY AND PORTABILITY

The restriction to 100 rows and 250 columns is quite arbitrary, intended to keep the largest procedure and all active data within the approximately 56K bytes of core available to a user program. The dimensions have in fact been changed for special cases where the known sparsity of the model insured that available space would not be exceeded. Maximum dimensions of 150 by 200 might be allowable, for example. The indexing used is not itself restrictive up to much larger dimensions.

The maximum model dimension involve several different considerations. First, they are built into the (fixed) dimensions of certain LOGICAL*1 arrays in the COMMON area named TYPES (except in LPINPUT where the array is not in COMMON). These have to be changed in all main programs and a fairly large number of subroutines. Second, LPINPUT has checks for maximum number of rows (100), columns(250) and total elements of all kinds (3000) which are coded numerically and a little hard to find. (They should be variables set at the top.) Third, enough room must be left for a reasonable-sized inverse representation ("eta" columns for product form). Fourth, total space available is controlled by various vectors in LPINPUT and by the array KORE defined in all other main programs. The size of KORE is also built into a couple of statements in the main programs for defining the amount left for the inverse. With a good editor and knowing where to look, all these changes can be made in about an hour or less but one can expect to miss a couple of required changes on the first attempt.

Core assignment, per se, is done primarily in LPSETUP and it is possible that adjustments have to be made there, not only for sizes but possibly for different options. LPSETUP works in an interactive question-response mode.

In order to extend the package for arbitrarily large (within reason) model dimensions, intermediate input/output buffering of matrix and eta files would have to be installed. This would be a large undertaking and one should consider whether the use of standard large MPSs would not be more practical. Such buffering could be done in FORTRAN but, for efficiency, should be done in a better system programming language. Furthermore, such features as multiple and partial pricing and a more powerful inversion routine would become desirable. Thus one could be quickly led into a large system effort.

Since LPS/11 is all-FORTRAN, it might be assumed that it is easily portable. This is not quite true. First, the use of LOGICAL*1 may not be available in other systems. (It doesn't work correctly with some older sets of library subroutines even under UNIX.) Second, the PDP default for integers is two bytes (which is sensible) but four bytes in IBM FORTRAN (causing subtle problems in call statements if mixed) and a full 60 bits in CDC FORTRAN (which is somewhat ridiculous). A working conversion of LPS/11 for a CDC Cyber 74 exists but required extensive modification for a number of reasons peculiar to that system. A further consideration is the definition and accessing of files which is particularly convenient on the PDP. System programmers will be familiar with such matters but implementing the package on another system may take a good deal of hunting and fixing.

A final consideration is the continuous nature of unformatted files with the library subroutines used. There is no concept of a logical record. READ and WRITE statements must specify exactly how many bytes are to be transmitted. Some rather intricate sequences exist for writing and reading intermediate files which must match across programs.