The models used for transportation planning are rarely run right through as one continuous computer job.

It is usual to break the models down into smaller self-contained steps, and manually examine the output from each of these steps before proceeding to the next.

RRLTAP is a suite of computer programs containing many of these model building steps. It has been designed to be used to run a complete model or a single step in each run. The essential features of the basic design are:

1. all steps are entirely self contained
2. any step communicates with any other via standard files written to tape or disc
3. any number of steps may be run in any order under RRLTAP control.

The RRLTAP suite has so far been used for research on assignment models at both the strategic and the tactical level. This note is to document the structure of the suite, and the design approach used to construct it.

1. INTRODUCTION

The RRLTAP Suite of Transportation Analysis Programs has been written to provide a flexible tool for research into network models of transportation at both the strategic and the tactical level. The networks encountered in conurbations are extremely large, ranging up to about 10,000 distinct pieces of road. The matrices used to record the number of trips between the different areas of the conurbations are also very big. It is not uncommon to break a conurbation down to 700-1000 geographical areas (or "zones"), and every matrix describing a particular kind of movement between these zones will then have about one million cells.

Transportation models are generally run in small, self contained steps. Each step may require a considerable amount of computing time, and it is usual to examine the results manually to ensure that all is well. As a result a complete transportation model can take one or two months to run through completely; even for comparatively small problems (say: 1-2,000 roads, 1-200 zones).

This is reasonable for the analysis stage of a large and continuing Land Use/Transportation study of an area, where a particular model is being applied to the particular area. It is no longer reasonable if a wide range of possible alternative transport investment proposals are to be studied, and is quite out of the question when research is being carried out on complete models of parts of the transportation planning process. To handle these strategic requirements one must be able to run the entire model in one go.
The tactical problems in the analysis of large areas require program steps with the capacity to handle large networks and matrices. In general, no two steps will fit at the same time into the core store of the computer being used: if they could, then capacity would be enlarged. The strategic problems require long chains of programs and, as RRLTAP is required to deal both with large problems and strategic modelling, the data generated at each stage must be kept outside the core of the computer. So also must all the programs not actually in use for the particular stage of the process that the computer is working on.

RRLTAP is organised in this way, so that all the input data required for a given stage is read in from a disc or a tape, and all the results of that stage written back to another disc or tape. In this way the programs used at each stage have all the available space in the computer core, as all intermediate results have been kept outside the core of the computer on tapes or discs.

As the RRLTAP suite can deal with any size of problem up to its maximum capacity, strategic runs may be carried out on simplified and smaller versions of the complex and detailed problem area, while a full tactical analysis can also be made by running the same programs on the networks and matrices covering this area at a finer scale of detail. The capacity of the current version of RRLTAP is 4000 one-way road links, 3600 road junctions ("nodes"), and 999 geographical areas.

2. DESIGN OF THE RRLTAP SUITE

At the start of this work no transportation planning models existed for the RRL computer, an ICL 4-50 at the time. RRLTAP was designed to carry out both tactical and strategic functions on the 4-50. The design aimed to reduce as far as possible any inevitable subsequent changes in the construction of the suite and hence minimise disruption due to later developments. These happened when disc files became available and when the ICL 4-70 was installed in place of the 4-50 with an Operating System different from that being used on the 4-50.

The suite has been designed to be run as a single program, with data cards read in to specify the operations that are to be carried out and the order of their execution.

Model development has been largely done on a comparatively small network, and the complete model finally adopted can now be processed either in stages or in one run on any network for which data is available on magnetic tapes or cards.

A principle feature that RRLTAP shares with many other transportation planning models is that a set of operations are carried out on (road) networks and matrices describing trips between (specified) areas.
The most common operations are the addition, subtraction, and multiplication of matrices of trips made by people or vehicles. These steps form a part of almost every modelling process that requires - or produces - a description of multiple movements from place to place.

One of the self-contained steps (e.g. TSW (4)) in RDLTA is therefore a sub-program that adds and subtracts multiples of matrices in standard RDLTA format.

There are thirty or more steps of this kind available under RDLTA, and every one reads in standard RDLTA files from tape or disc and writes out RDLTA files to tape or disc when the computation is complete.

This structure means that any of these steps can be run in any order, as they communicate only through the disc or tape files. The penalty paid is quite high for small problems, as the files are read and written to magnetic tape or disc many times in a run, and the time taken for this can be much greater than spent actually carrying out the calculations for each step.

The use of magnetic disc reduces this penalty and every 'step' in RDLTA can be told to use either discs or tapes as required.

RDLTA has been designed for research with the objective of comparing the economic benefits of different road investment proposals. Each 'step' in RDLTA has several types of options. Some of these refer to the location of input and output data, others to the actual operations to be done. When a sequence of steps has been defined to specify a particular type of model, judicious use of these options can produce a very wide range of models.

It is not appropriate to go into too much detail here, as a separate Note (4) has been produced to describe some of these models. But, to give some idea of how RDLTA may be used, consider the process:

1. Build a road network
2. Find shortest Paths between each pair of geographical areas (zones)
3. Allocate zone-to-zone trips to these paths.

At this level, the process may be specified by three RDLTA 'steps' which are described in detail in reference 2. Each 'step' corresponds to a sub-program which carries out the step:

1. *TSW10 : which commands RDLTA to construct the network from data provided.
2. *TSW1 : which commands RDLTA to construct shortest paths
3. *TSW4 : which commands RDLTA to allocate zone-to-zone trips to these paths.
These major functional steps describe an 'all-or-nothing' assignment of traffic flow, where every trip between a specified pair of zones travels along the same path between the zones. The 'model' defined by these three steps is that used in many transportation studies to simulate the flow of traffic over a road network.

For each 'step' of this example, we shall consider the selection of particular alternatives from the options offered by each.

In 'step' 1, we can choose to construct our new network entirely afresh from data cards, or to modify an existing network by adding and deleting links.

In 'step' 2, we can choose a particular shortest path criterion from the alternatives of shortest time, shortest distance, shortest cost, or a minimum generalised cost (made up from time, cost and distance by placing a value on time and specifying a cost/mile charge).

In 'step' 3 we can decide to allocate only a percentage of the trips between each zone, or only allocate trips between specified zones to the network. We may allocate more than one matrix of trips, or multiples of several matrices of trips.

From this simple example we have shown that the broad structure of a given model is specified by the user as a series of 'steps' from the range offered under RRLTAP. The detailed structure is controlled by options offered by each 'step' as input data.

Extending this example, it is clear that we could stop between steps 1 and 2 or 2 and 3, and restart again at a later date as long as the user made sure that the 'Network' or 'shortest paths' files were to be written to magnetic tape at the end of steps 1 and 2 by using the appropriate input data for each step. In fact a restart can be made between any pair of stages, as long as the data required for the next stage has been written out to tape rather than disc.

This is exploited to provide a simple way of giving the computer operator the option of terminating a long run at one of these possible breakpoints. The technical details are given in reference 3. Reference 2 gives an example of the practical use of the method. A schematic view of RRLTAP is shown in Fig 1, illustrating these points.

The actual design of the files used for 'step'-to-'step' communication is of great importance if RRLTAP is to be used effectively.
The purpose of this discussion is to show how the files used by KDLTHA are protected and identified. The precise details of the various forms of standard file used in KDLTHA are to be found in reference 2: Section 12.

The two most common types of files are:— (1) Network files
(2) Matrix files

3.1 Network files always contain a complete specification of the structure of a (road) network in forms of (road) links between numbered points (nodes). For each link a small list of data is held: this list contains the length of the link, any monetary charges made on the link, an identifying code number for the flow-speed characteristics of the link, an identifying code number for the geographical position of the link, and the cost to travel from one end of the link to the other.

A term 'resistance' may be used as the unit of measure and may be cost, time, or a combination of the two. The 'resistance' is the 'cost' of travelling from one end of the link to the other, and the measure is specified by the coefficients $A$ to $D$ in the complete generalised cost equation:

$$\text{Cost} = (\text{length of link}) \times (A + \frac{B}{\text{speed}} + C(\text{speed})^2 + \frac{D}{\text{speed}}) + \text{toll}$$

* is a direct monetary charge.

Where 'speed' is the mean speed of travel along the link concerned. This equation is made up of three separate components:

(1) \( (A + \frac{B}{\text{speed}} + C(\text{speed})^2) \) pence per mile

This is the operating cost/mile of the vehicle itself as perceived by the user.

(2) \( \frac{D}{\text{speed}} \) pence per mile

This is the further contribution of time spent/mile to the generalised cost equation: i.e. 'D' is a "valuation" of time, in pence/hr.

(3) direct monetary charge or toll

This toll is any parking, road pricing, or direct cash charge levied on the user to travel from one end of the link to the other. (If the far end of the link is the end of his total journey, this charge is effectively a parking charge).
All network files contain a record of the generalised cost function \((A, B, C, D)\) (abbreviation of above) used to construct the link measures, and if the network is altered it is adjusted accordingly.

If we use a particular cost function to find the minimum cost paths between points on a network, the output of a program used to build these paths is a Network-Tree file, and also contains a copy of the network used.

A Network-Tree file consists of a complete network description file, followed by a list of all the minimum paths constructed by the program.

The generalised cost function used to construct these minimum paths need **not** be the same as that used for the link measures in the network. For example: the link measures on the network may be simply time taken to travel along the link \((0, 0, 0, 1 = A, B, C, D)\) whereas the 'shortest paths' we require may be shortest distance \((1, 0, 0, 0 = A, B, C, D)\).

To make sure that we can keep track of these (possibly) different cost functions, an entry is made to describe a 'network' file as merely a Network - when the entry will be 'blank' or as a Network-Trees file - when the entry will be 'T' - a record of these details is held at the beginning of all network files.

When we have obtained a set of minimum paths we take the number of vehicles or trips between each pair of points and load them onto the minimum path between these points.

The result of this process is a Loaded Network file, which consists of a complete network file followed by a list of the flows on every link of this network. The network identifying record is given the value 'L' to show that these flows are appended.

This system of specifying network types and cost functions is used extensively in RELTAP to ensure that only valid operations are carried out.

3.2 The matrix files have a number of different interpretations, but all matrix files include a single (first) record containing identifying data.

Matrices may be numbers of trips between zones, or distances, costs or times for such movements. If the matrices refer to costs, distances or time, the first (identifying) record contains a list of constants specific to that matrix. The first four are the cost function coefficients, to specify the precise meaning of the matrix entries. Unless they are specifically set up in this way, all these constants are zero.
The use of these cost functions will be demonstrated by a simple example:

step 1. Build a network, using cost function \( c.f(1) \)
step 2. Build shortest paths, using cost function \( c.f(2) \)
step 3. Find the total cost between each pair of zones incurred by following the paths defined in step 2, and applying a cost function \( c.f(3) \)
step 4. Print these zone-to-zone costs.

In this process we may build the network on the basis of time:-

\[ (i.e. \ c.f(1) = (0, 0, 0, 1) ) \]

We then build paths of shortest distance

\[ (i.e. \ c.f(2) = (1, 0, 0, 0) ) \]

The total cost of a trip between a specified pair of zones \( i, j \) can then be found by valuing time at the rate of, say, 200 d/hr, and assuming that the cost of travel was 4d/mile.

\[ (i.e. \ c.f(3) = (4, 0, 0, 200) ) \]

The identifying record at the beginning of each matrix ensures that the user (and the program) always knows precisely what is in any given matrix. However no check is made to ensure that a particular input file is a matrix or a network, as full diagnostic information will automatically be produced and printed out before \( H.M.L.T.A.P \) subsequently stops due to the different formats in which networks and matrices are stored.

A total of up to eleven matrices may be kept on one matrix file, and the identifying record contains the cost function and constant list defined earlier for all eleven separately. All the elements of the first row of matrix 1 are written, then all the elements of the first row of matrix 2 etc.

The main use of this interleaved matrix system is to retain several different pieces of information on one file that apply to a single set of numbered geographical areas (zones). For example, trips by car, bus, and rail: or trips on summer, pleasure, or commuting. Interleaved storage allows us to handle all these matrices at once, without having to hold them all in the computer at the same time. This is essential for the large matrices that often occur in transportation planning.

A special form of \( H.M.L.T.A.P \) file has been designed to act as a complete 'archive' of an \( H.M.L.T.A.P \) model run. This is specified under 'constant file' in ref. 2. The creation and manipulation of this archive from matrix, network, trees, and loaded network files is specified under "T.S. 18" in the program specifications of ref 2.
The files for any run are collected from several different tape and disc files, and a material saving in the number of magnetic tapes required for the storage of results can be obtained.

This section has been used for a brief description of the most important file structures used throughout RRLTAP, full details are given in ref 2.

4. CONTROL STRUCTURE OF RRLTAP

RRLTAP offers a large number of operations which act on input standard files and output standard files. These operations may be carried out one by one, or in as long a sequence as may be required for a single run.

Models may be built and run by selecting a suitable sequence of operations, providing suitable data is inserted for each operation to control the precise action of the operation.

Every RRLTAP sub-program (or 'operation') must be able to handle the largest problems that the suite can accommodate, and many of the sub programs require almost all the available space in the core of the computer.

The RRLTAP program is stored on a disc, and the sub programs we require at any given time are read in. A small part of RRLTAP must always reside in the core of the computer, and this part must analyse the user's requests, and issue requests for the particular sub program required.

By a simple programming artifice, it is possible to initiate this process in a part of RRLTAP that need not always remain in core, and thus release more space for all the RRLTAP subprograms to use.

The way that control is passed through the RRLTAP program is illustrated in Fig. 2. In this example only RRLTAP 'sub-program' $\gamma$ is actually allowed to return control to the point from which it was called, the other operations in the illustrated loop are never allowed to do this. Strictly speaking all the subroutines on the loop labelled 'a, a, a' call themselves ('recursive' calling: which cannot normally be used in FORTRAN), but the complications that this could cause have been bypassed. Fig. 3 shows the essential FORTRAN details of the method, which depends on a 'dummy' subroutine (NSUB) being called. NSUB is replaced by the actual name of the subroutine called on each occasion. Subprograms $\alpha, \beta, \gamma \ldots$ etc are the RRLTAP sub programs, and will be handled in exactly the same way as 'y' in Fig. 2.

There are many advantages in this method, but the essential purpose is to give more core store space to the RRLTAP sub programs by making it possible to use the same
space for the necessary data processing of control transfers as is used by the RRLTAF subprograms that are involved.

This freedom to expand the 'control' subroutine without affecting the space available to RRLTAF subprograms has been used to provide a number of useful features. These are illustrated in an example in Annex 1 of the output resulting from a specific list of data cards submitted to RRLTAF.

The 'control' routine searches for cards of the form:—
**COMMENT', or '"TSW'

"COMMENT" is merely a convenience for the user to print comment cards on his output, '"TSW' cards select the major RRLTAF subprograms which all have names 'TSW nm'. If TSW n (or TSWnm) is not included in RRLTAF, an error message is given, and the program searches for the next '"TSW,' '"COMMENT' card. Until it finds one, it will print out every card encountered with an 'error' warning.

The user can request that the program give the operator a 'Check point' message at the end of any specific '"TSW' subprogram. This has the effect of halting the program with all the backing files in the state that the last '"TSW' subprogram has just left them, and gives the operator a chance to terminate, or dump, or continue the job at this clear breakpoint.

This is an option that the user can exercise when setting up the data pack for a particular run of RRLTAF. It is equally important to have some means of allowing the computer operator to terminate the RRLTAF run while it is actually going, i.e. for example it has over-reached the time allocated.

An extra feature is included in the RRLTAF Checkpoint system that allows the operator to do this without upsetting the operation of the sub program running when he interrupts it for this purpose. The result is that a checkpoint is issued at the end of this subprogram, so that all files in use for the run are "clean" and complete when the run is prematurely terminated.

The illustration in Annex 1 includes all these features (except the printing produced when responding to an operator-induced checkpoint).

5. OVERLAY DESIGN

RRLTAF has been segmented for efficient running from the disc on which the entire program resides. The details are given in ref 2 (Users Manual) in Computer terms as a list of cards specifying the detailed overlay design used for the current RRLTAF release. The purpose of this section is to give the basic design of the overlay
structures used for RRLTAP, and is not dependent on the finer details of each part of the overlay used for a specific release of RRLTAP.

The basic rule applying to overlay structures in ICL 4/70 FORTRAN is that all subroutines in a given chain of calls down to the subroutine being executed must be present in store while this subroutine is working. Overlays may be structured to follow the calling structures closely, so that each calling chain is realised in the overlay with every subroutine lying 'above' all the subroutines that it calls. This need not be done. The overlay is not required to follow the calling structure, and in RRLTAP an inverted structure has been adopted where all subroutines called by the main RRLTAP subroutine lie at the very bottom of the overlay. The straightforward principles used are illustrated in Annex 3. The RRLTAP overlay therefore has the most commonly used input-output subroutine 'high up' in every chain, and the major RRLTAP subprograms at the bottom. This means that it is necessary only to replace the end of a chain when transferring control from one major RRLTAP subprogram to the next, and saving computer time by reducing the amount of 'program' to be brought from the disc.

It is worth noting that an inverted overlay structure cuts down the size of the specification of the overlay itself, and reduces both user errors and the computer time required to construct the whole program from the component subroutines specified for the overlay.

A schematic layout of the overlay structure constructed in this way for RRLTAP is given in fig 4, and illustrates the general grouping of the major RRLTAP sub-programs.

6. EXTENSION OF RRLTAP

The modular form of the RRLTAP suite makes it very easy to add new major subprograms. The input/output, the standard storage packing structures used, and the access to such packed data is all embodied in standard subroutines. The overlay structure is such that it is easy to add a further major sub program, as all these lie at the 'bottom' of the overlay.

All the file structures used for subprogram communications and storage of intermediate and final results are fully specified in ref 2.

The only program modification required for the addition of a subprogram is to add the statements

EXTERNAL TSWnm
IF (IX.PQ.mm) CALL NCALL (TSWnm)

to the subroutine named 'NEXT' in Fig 3 and listed in ANNEX 2.
If a 'call' is set-up for a major sub routine that is not included in RRLTAP: (eg "TSW??" in the example of ANNEX 1), a message to this effect will be printed and the program proceeds to the next card of the input data. This action, and many others, is illustrated in the example of ANNEX 1.

A brief list of some of the major subprograms of RRLTAP, with a specification of their function, is given in ANNEX 4.

7. SUMMARY
We have discussed the design and implementation of an extendable system of programs for Transportation Studies. The suite is of modular structure and standard format, backing files are used to communicate results between the various modelling operations. The system used to transfer control from one major subprogram to the next has been described.

This Note has been produced to supplement the Users Manual(2) by explaining the more general design features implicit in RRLTAP.

This Note was prepared in Road Systems Section of Traffic Division.

8. REFERENCES


9. ANNEX I

1. Data Cards for an RRLTAP run:

```
*COMMENT
ABCDEFGLJK MEANINGLESS TO RRLTAP = ERROR CARD
*TSW77
*TSW95: STOP: REQUESTS A "CHECKPOINT" AFTER TSW95 HAS ENDED
*TSW95
*COMMENT
```

2. Output written by RRLTAP to the line printer will be:

```
RRLTAP STARTS
INPUT DATA CARD PACK EXAMPLE
RRLTAP ERROR CARD: - ABCDEF..LJK MEANINGLESS TO RRLTAP = ERROR CARD
RRLTAP CONTROL CARD: - *TSW77:......: NOT IN RRLTAP = ERROR IN CALL: CALL 77 FAILURE
RRLTAP CONTROL CARD: - *TSW95: STOP: REQUESTS A CHECKPOINT AFTER TSW95 HAS ENDED
(printing from subroutine TSW95)
JOB TSW9RLTAP01P RUN 000201 "CHECKPOINT" 95" ............ 2 AT "9P : 50 : 12"
TYPE-N OFF TO GO ON : 1 FOR EQJ : 2 FOR EQJD
OPERATOR REPly: - N 2
RRLTAP CONTROL CARD: - *TSW95:............:
(printing from subroutine TSW95)
LAST CARD FOR RRLTAP EXAMPLE
RRLTAP ENDS
```

3. Key to illustration:

- a: RRLTAP message printing
- c: RRLTAP comment printing
- e: RRLTAP error printing
- se: RRLTAP subroutine calling and call-error printing
- p: ordinary output from an RRLTAP subroutine
- sc: RRLTAP checkpoint setting + subroutine calling

Example of the flow of control as set up by the use by the data cards
Alternative overlay designs to satisfy a given subroutine calling structure.

1. Calling structure: $\alpha \rightarrow \beta$ indicates that the $n^{th}$ call is by subroutine $\alpha$ to subroutine $\beta$.

2. Overlay structure closely following the calling structure

   - $A$ is always in store
   - Only one of $B$, $C$, $D$, $E$ can be in store at the same time
   - Only one of $X$, $Y$ can be in store at the same time

3. Overlay structure inverted with respect to the calling structure.

   - Here the first call to each chain refers to the bottom of the overlay, and subsequent calls are upwards, to steps in the chain that had not been passed through on the way to the call to the bottom.
Some examples of MULTAP major subprograms

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSW1</td>
<td>To construct and print out paths of minimum cost, time or distance between points on a network.</td>
</tr>
<tr>
<td>TSW2</td>
<td>To print out the description of a network, with full details of the length, speed, cost of every link.</td>
</tr>
<tr>
<td>TSW3</td>
<td>To construct from cards a matrix of the number of trips between a set of zones on a network.</td>
</tr>
<tr>
<td>TSW4</td>
<td>To put the trips between specific zones on a network onto the individual links of the network forming a shortest path defined by TSW1.</td>
</tr>
<tr>
<td>TSW10</td>
<td>To construct update or amend a network.</td>
</tr>
<tr>
<td>TSW14</td>
<td>To add, factor, or merge matrices of trips.</td>
</tr>
<tr>
<td>TSW25</td>
<td>To compress the size of a matrix of trips.</td>
</tr>
<tr>
<td>TSW28</td>
<td>To plot out a network, the flows of trips over each link, or the shortest paths defined by TSW1 on to an 11&quot; CALCOMP graph plotter.</td>
</tr>
</tbody>
</table>

(This is a brief selection from the list of sub programs specified in detail in the User's Manual(2).)
* TSW cards select the next subprogram to be used. The subsequent data cards are the symbolic files to be used by the subroutine.
Pass control to the subprogram whose name has been supplied

Return to read data card for next RRLTAP subprogram

Read in next data card, specifying the RRLTAP Subprogram required

Identify the subprogram and send its 'name' up to be called in and carried out

RRLTAP Subprogram α

RRLTAP Subprogram β

RRLTAP Subprogram γ

RRLTAP Subprogram δ

Group 1

Group 2

Group 3

Fig. 2. RRLTAP CONTROL STRUCTURE, SHOWING THE WAY TRANSFER OF CONTROL FLOWS THROUGH THE PROGRAM

a.e.a. is the (recursive) loop that is followed to carry out RRLTAP control transfers in accordance with the input data cards provided by the user.
Program RRLTAP
CALL SET
END

Subroutine NCALL
CALL NSUB (NSUB)
CALL SET
END

Root segment

Node

Subroutine SET 1
C Run initialisation
ENTRY SET
READ (-) NXTSUB
CALL NEXT (NX7SUB)
GO TO 1
END

Subroutine NEXT (IX)
External TSW1, TSW2...
IF (IX.EQ.2) CALL NCALL (TSW2)
RETURN
END

Segment 1

Segment 0 2
RETURN
END

Segment 0 3
RETURN
END

(Only one of segments 1, 2, 3, 4 is allowed in core at any one time)

Fig. 3. RRLTAP CONTROL STRUCTURE: FORTRAN SCHEMATIC
Fig. 6. SCHEMATIC LAYOUT OF RRLTAP OVERLAY STRUCTURE