REPORT NO. FRA-76-26-II
FRA/NECPO-76/22, II

MODELS OF RAILROAD PASSENGER-CAR REQIREMENTS IN THE NORTHEAST CORRIDOR

VOLUME II: USER'S GUIDE

ROBERT FOURER

National Bureau of Economic Research
Computer Research Center for Economics and Management Science
575 Technology Square
Cambridge MA 02139

SEPTEMBER 1976

FINAL REPORT

This document is available to the U.S. public
through the National Technical Information Service
Springfield VA 22161

Prepared for

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL RAILROAD ADMINISTRATION

Northeast Corridor Project Office
Washington DC 20590

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

NOTICE

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FRA/NECPO-76/22		

4. Title and Subtitle	5. Report Date
MODELS OF RAILROAD PASSENGER-CAR REQUIREMENTS IN THE NORTHEAST CORRIDOR Volume II: User's Guide	September 1976
	6. Performing Organization Code
7. Author(s) Robert Fourer	8. Performing Organization Report No. DOT-TSC-FRA-76-26,II
9. Performing Organization Name and Address National Bureau of Economic Research* Computer Research Center for Economics and Management Science 575 Technology Square Cambridge MA 02139	10. Work Unit No. (TRAIS) RR622/R6301T
	11. Contract or Grant No. DOT-TSC-1179 -2
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Railroad Administration Northeast Corridor Project Office Washington DC 20590	13. Type of Report and Period Covered Final Report April- July 1976
	14. Sponsoring Agency Code

15. Supplementary Notes

*Under contract to: U.S. Department of Transportation Transportation Systems Center Kendall Square Cambridge MA 02142

16. Abstract

Models and techniques for determining passenger-car requirements in railroad service were developed and applied by a research project of which this is the final report. The report is published in two volumes, as follows:

Volume I: Formulation and Results. The first part of this volume considers a general problem of determining optimal passenger-car allocations given a fixed schedule and predetermined demands. Requirements for car movements are modeled as a set of linear constraints having a transshipment structure, and alternative linear objectives are formulated. Various optimization techniques are developed for one or more objectives, and properties of the sets of optimal solutions are demonstrated.

The remainder of Volume I shows how the linear model and optimization techniques may be applied to the Northeast Corridor. Derivations of a schedule and demands are explained, and results of a number of optimizations and analyses are displayed.

Volume II: User's Guide. The solution and analysis of the Northeast Corridor models required the creation of a number of computer programs of several kinds. These programs are available for the use of others and are described in Volume II of this report.

17. Key Words	18. Distribution Statement
Inter-City Travel (Rail) Linear-Programming Models Passenger-Car Requirements (Railroad) Northeast Corridor	This document is available to the U.S. public through the National Technical Information Service Springfield VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	66	

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

PREFACE

In 1973, Congress passed the Regional Railroad Reorganization Act which became law on January 2, 1974. This complex piece of legislation dealing with passenger as well as freight operations called for the Department of Transportation (DOT) to implement improved passenger rail service in the Northeast Corridor (NEC) as recommended in the 1971 Northeast Corridor Report. Planning for the improved service included engineering studies, demand projections, and financial analysis. This study uses the demand projections to compute fleet requirements for the Corridor. The results contained in the base runs do not represent official or final estimates of fleet size, train schedules, and fleet management but are illustrative examples based upon assumptions stated in the text.

The work described herein was performed for the Transportation Systems Center under contract no. DOT-TSC-1179 with the National Bureau of Economic Research. The contract was carried out by Robert Fourer at NBER's Computer Research Center for Economics and Management Science. Judith Gertler and Howard Simkowitz of TSC developed the demand-projection procedures and provided general guidance for the project. The PL/I routines for simulating annual patronage were written by John Prokopy and Diane Ruina of Peat, Marwick, Mitchel, and Co.

The author wishes to thank William Northup and Michael Harrison of NBER for their occasional technical assistance, and Karen Glennon for her patience in typing the manuscript.

Prospective users of the programs documented in Volume II should contact the author, care of NBER, or Howard Simkowitz at the Transportation Systems Center.

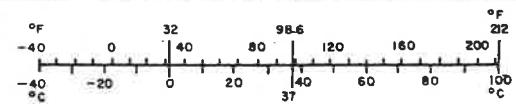
METRIC CONVERSION FACTORS

Approximate Conversions to Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
in	inches	2.5	centimeters	cm
ft	feet	30	centimeters	cm
yd	yards	0.9	meters	m
mi	miles	1.6	kilometers	km
		AREA		
in^2	square inches	6.5	square centimeters	cm^2
ft^2	square feet	0.09	square meters	m^2
yd^2	square yards	0.8	square meters	m^2
mi^2	square miles	2.6	square kilometers	km^2
	acres	0.4	hectares	ha
		MASS (weight)		
oz	ounces	28	grams	g
lb	pounds	0.45	kilograms	kg
	short tons (2000 lb)	0.9	tonnes	t
		VOLUME		
tsp	teaspoons	5	milliliters	ml
Tbsp	tablespoons	15	milliliters	ml
fl oz	fluid ounces	30	milliliters	ml
c	cups	0.24	liters	l
pt	pints	0.47	liters	l
qt	quarts	0.95	liters	l
gal	gallons	3.8	liters	l
ft^3	cubic feet	0.03	cubic meters	m^3
yd^3	cubic yards	0.76	cubic meters	m^3
		TEMPERATURE (exact)		
°F	Fahrenheit temperature	5/9 (after subtracting 32)	Celsius temperature	°C

Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
		LENGTH		
mm	millimeters	0.04	inches	in
cm	centimeters	0.4	inches	in
m	meters	3.3	feet	ft
m	meters	1.1	yards	yd
km	kilometers	0.6	miles	mi
		AREA		
cm^2	square centimeters	0.16	square inches	in^2
m^2	square meters	1.2	square yards	yd^2
km^2	square kilometers	0.4	square miles	mi^2
ha	hectares (10,000 m^2)	2.5	acres	
		MASS (weight)		
g	grams	0.035	ounces	oz
kg	kilograms	2.2	pounds	lb
t	tonnes (1000 kg)	1.1	short tons	
		VOLUME		
ml	milliliters	0.03	fluid ounces	fl oz
l	liters	2.1	pints	pt
l	liters	1.06	quarts	qt
l	liters	0.26	gallons	gal
m^3	cubic meters	35	cubic feet	ft^3
m^3	cubic meters	1.3	cubic yards	yd^3
		TEMPERATURE (exact)		
°C	Celsius temperature	9/5 (then add 32)	Fahrenheit temperature	°F



CONTENTS OF VOLUME II

CONTENTS OF VOLUME II (Concl'd)

LIST OF EXAMPLES

§1 THE COMPUTER ENVIRONMENT

The techniques set forth in Volume I of this report [2] were applied by use of several interactive computer systems in conjunction with a set of programs written specially to run under these systems. These specialized programs are available to other users in many cities through a national telecommunication network; a brief guide to using the programs comprises the present volume.

The programs described herein operate under the Conversational Monitor System (CMS) of the Virtual Machine Facility/370 (VM/370). CMS is an interactive system that you operate by typing commands at a remote terminal. It runs in a virtual-machine environment: that is, an individual virtual computer is simulated for each user of the system.

You enter VM/370 and CMS by typing a sequence of logon commands. The format and order of these commands varies between installations; inquire for instructions at the installation you are using. To leave VM/370 and end your terminal session, type LOGOFF.

Many of the routines described here employ, or require you to use, the SESAME linear programming system. SESAME is an interactive system that operates under CMS. SESAME's main routines are used to solve and analyze models, while a subsystem of SESAME, called DATAMAT, is employed to generate LP models and tables of result data.

The volume is not intended to introduce new users to wither CMS or SESAME. If you are not familiar with these systems, you are urged to acquire the following manuals and to consult them whenever questions arise:

VM/370 and CMS. See the *Command Language Guide for General Users* [3], *EDIT Guide* [4], and *EXEC User's Guide* [5]; or, if you are using Tymshare CMS, the *CMS Reference Manual* [9] and *EXEC Language Reference Manual* [10].

SESAME and DATAMAT. Consult the *SESAME Primer* [7], *SESAME Reference Manual* [8], and *DATAMAT Reference Manual* [1].

§1.1 Virtual machine requirements

Your virtual machine should have a read-write virtual disk, attached as your A-disk, that is initialized to contain the program files listed in Appendix A. Every file in CMS has two names: a *file-name* and a *file-type*. To print any CMS file at your terminal, type the following command:

TYPE *file-name file-type*

Files whose *file-type* is EXEC are actually programs of CMS commands that can be executed from the terminal or called from other EXEC programs. The program in file PROFILE EXEC is a special one: it is automatically executed at the beginning of every CMS session to initialize your virtual environment in various ways. Other EXEC programs are documented in subsequent sections of this report.

Your virtual machine should have at least 1024K of storage to run most of the programs documented in this volume. You can define any amount of storage (up to the limit for your account) by typing the following CMS command:

DEFINE STORAGE *n*K

To see how much storage you currently have, type QUERY STORAGE.

Certain programs' output files are usually too long to be printed routinely at the terminal. These files are sent instead to your virtual printer. The standard PROFILE EXEC initializes CMS so that files in your virtual printer are printed at the central computer and mailed to you. You may wish to change this arrangement, however, depending on the availability of high-speed offline printers at more convenient locations.

§1.2 General instructions for terminal operation

To delete the most recently typed character on a line, type @. To delete the *n* most recently typed characters, type *n* @'s.

To delete all characters that have been typed on a line, type ¬ on EBCDIC terminals, \ on ASCII terminals. (You can tell whether a terminal is EBCDIC or ASCII by whether it has a ¬ or a \ key.)

To stop execution of a program or to halt or resume typing at the terminal while execution continues, you must first enter the CMS-immediate environment. The procedure for doing this varies between computer installations: ask for instructions at the installation you are using. Once in the CMS-immediate environment, you can type the following commands:

Command	Action taken
HX	Execution of the program that is currently running is terminated. You are returned to the CMS environment and prompted for another CMS command.
HT	Typing at the terminal is suspended, but execution of the currently-running program continues. Typing will resume when execution of the current program ends.
RT	Typing at the terminal resumes immediately.

VM/370 prints a "blip" character at your terminal wherever you use up another one or two (depending on the installation) seconds of computer processing time. Blips give you a rough idea how fast a program is running, and how much you are spending. The standard PROFILE EXEC make the blip character a dollar sign ($).

§2 COMPUTING ANNUAL PATRONAGE

Annual patronage projections for the Northeast Corridor are computed by a 4-mode (rail, air, bus, auto) estimation program that was developed for a Corridor study by Peat, Marwick, Mitchell and Company [6]. This section describes program compilation and execution, the required input, and the resulting output.

A terminal session showing a typical run of the patronage program is reproduced in Example 1.

§2.1 Compilation

The patronage program is written in PL/I. Its source code is in three files: MODAL PLI, SNOW PLI, and DMDOUT PLI. MODAL PLI contains the main program; SNOW PLI and DMDOUT PLI hold subroutines that print summary statistics and rail patronage, respectively (see §2.4). These files have already been compiled; you need to recompile them only if you make changes to their contents. (Changes are most easily made by use of the CMS EDIT subsystem [4,9].)

To compile a source file, type

PLICOMP *file-name*

where *file-name* identifies one of the three source files (MODAL, SNOW, DMDOUT). Warnings or errors are listed at your terminal by the compiler; a source listing and other compilation data are sent to your virtual printer. A file of compiled code, named *file-name* TEXT, is created by the compiler on your A-disk; it replaces any previously-created TEXT file of the same name.

PLICOMP is implemented as a CMS EXEC program [5,10]. It is stored in file PLICOMP EXEC on your A-disk.

Example 1. A typical compilation and run of the patronage program.

```
C>plicomp modal
PL/I OPTIMIZER V1 P2.3 TIME: 17.23.29  DATE: 29 JULY 76
OPTIONS SPECIFIED
 A IS MAP NEST OF OP S STMT STG XREF


NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME    0.47 MINS        SPILL FILE:     0 RECORDS, SIZE  3491
P;

C>plicomp snow
PL/I OPTIMIZER V1 P2.3 TIME: 17.25.49  DATE: 29 JULY 76
OPTIONS SPECIFIED
 A IS MAP NEST OF OP S STMT STG XREF


NO MESSAGES OF SEVERITY W AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  I I.
COMPILE TIME    0.13 MINS        SPILL FILE:     0 RECORDS, SIZE  3491
P;

C>plicomp dmdout
PL/I OPTIMIZER V1 P2.3 TIME: 17.25.38  DATE: 29 JULY 76
OPTIONS SPECIFIED
 A IS MAP NEST OF OP S STMT STG XREF


NO MESSAGES OF SEVERITY W AND ABOVE PRODUCED FOR THIS COMPILATION
MESSAGES SUPPRESSED BY THE FLAG OPTION:  I I.
COMPILE TIME    0.02 MINS        SPILL FILE:     0 RECORDS, SIZE  3491
P;
```

```
C>medal patr
EXECUTION BEGINS...

LOWYR=1982,HIGHYR=1982,POPSW='EXP',INCSW='EXP',PRINTSW='DETAIL',
LHT_FACTOR(1)=1.0,LHT_FACTOR(2)=1.0,LHT_FACTOR(3)=1.0,LHT_FACTOR(4)=1.0,
LHC_FACTOR(1)=1.00,LHC_FACTOR(2)=1.000,LHC_FACTOR(3)=1.00,LHC_FACTOR(4)=1.0000,
FREQ_FACTOR(1)=1.0,FREQ_FACTOR(2)=1.0,FREQ_FACTOR(3)=1.0,FREQ_FACTOR(4)=1,
MODE_FACTOR(1)=1.0,MODE_FACTOR(2)=1.0,MODE_FACTOR(3)=1.0,MODE_FACTOR(4)=1,
AUTOCOST=.0502,DUMPSW=0,ITERCNT=00,YKA=1.0000,CPSW=00,BASEYR=1974,
A2(1,1)=3.384,A2(2,1)=3.384,A2(3,1)=3.384,A2(4,1)=3.384,A2(1,2)=1.5821,
A2(2,2)=1.5821,A2(3,2)=1.5821,A2(4,2)=1.5821,
A3(1,1)=0.483,A3(2,1)=0.483,A3(3,1)=0.483,A3(4,1)=0.483,A3(1,2)=1.5821,
A3(2,2)=1.5821,A3(3,2)=1.5821,A3(4,2)=1.5821,
A4(1,1)=0.000,A4(2,1)=0.000,A4(3,1)=2.2790,A4(4,1)=2.2790,A4(1,2)=0.000,
A4(2,2)=0.000,A4(3,2)=2.0462,A4(4,2)=2.0462,BUSSW=2,
KAY(1,1)=0.12,KAY(2,1)=0.12,KAY(3,1)=0.12,KAY(4,1)=0.12,KAY(1,2)=0.180,
KAY(2,2)=0.180,KAY(3,2)=0.180,KAY(4,2)=0.180,
INCN1=1.77,INCN2=203.0,OCC(1)=1.32,OCC(2)=2.13,
TOTA(1,1)=0.0000,TOTA(2,1)=-0.0763,TOTA(3,1)=0.1162,TOTA(4,1)=0.3920
TOTA(1,2)=0.0000,TOTA(2,2)=0.3273,TOTA(3,2)=-0.2527,TOTA(4,2)=0.6872,
BETA=.6,INDUCE_SUPPRESS='NO',
MAX.CTY_PRS=90,MAX.LINKS=11,CALIBRATION='YES';
$$$$$
$
R;

C>type patr fortdata

   231422   106030   941846    94480   254700  1467776   100100    30647    13847    16460    91386
    34070   334283    10241   115887   604158    10655    10502     2005     7100    21258
   197841     4736    52604   316555     7260     4533      604     1072     3752
        0   841322  3085705    52346    12103    12833    26008   100300
        0  1812035     4710     3037      955     3827     9606
        0        0        0     5284    12010    61419
        0   613866   204046   266890  1180730
        0   120135    73273   304556
    34785    20031   128506
    18449   102153
   530007

R;

C>
```

§2.2 Input data

Six input files are required by the patronage program:

```
NECDICT   PLIDATA
FPOPINC   PLIDATA
NEC1974B  PLIDATA
FORE2A    PLIDATA
NECLINKS  PLIDATA
PARMS     PLIDATA
```

The contents and formats of these files are described in Appendix B to this volume.

You can make changes to these files by use of the CMS EDIT subsystem [4,9]. EDIT can also be employed to create new input files.

§2.3 Execution

To execute the patronage program, type

MODAL *output-file*

where *output-file* is the name to be given to a file of annual rail patronages produced by the program (see §2.4). If you are going to use this file as input to the demand program (§3), *output-file* should be PATR.

MODAL is implemented as a CMS EXEC program [5,10]. It is stored in file MODAL EXEC on your A-disk.

§2.4 Output

The patronage program produces a summary of demand and other data by city-pair and transportation mode. The summary file is written to your virtual printer.

Total rail patronages only for all city-pairs are written separately to the file *output-file* FORTDATA, where *output-file* is the name specified in the MODAL command. This file has a special format required for input to the demand program (§3), as shown in Example 1. Successive lines in the file

correspond to successive cities in the Corridor, running south to north. The numbers on a line are round-trip patronages between the line's corresponding city and all cities farther north; again, the order is south to north. The first number on the first line is thus for Washington-Baltimore, the last number on that line is for Washington-Boston, and the only number on the last line is for Providence-Boston.

§3 CALCULATING DEMANDS

Demands for trains in a particular schedule are computed from annual patronage by a program developed at the Transportation Systems Center by Judith Gertler and Howard Simkowitz [2, §§3.4-3.7]. Instructions for compiling and executing this program are given below, along with requirements for input and a description of output.

Example 2 shows how the demand program is used in a typical terminal session.

§3.1 Compilation

The demand program is written in FORTRAN. Its source code is in five files:

```
DEMAND FORTRAN
SOUTH  FORTRAN
PRNT   FORTRAN
MAXLK  FORTRAN
DEM    FORTRAN
```

DEMAND FORTRAN contains the main program. Each of the others holds one or more subroutines. These files have already been compiled; you need to recompile them only if you make changes to their contents. (Changes are most easily made by use of the CMS EDIT subsystem [4,9].)

To compile one of the source files, type

FORTCOMP *file-name*

where *file-name* identifies one of the five demand-program source files. A return code of zero from FORTCOMP indicates a successful compilation; a positive return code indicates that there were compilation errors. The

compiler's source listing is sent to your virtual printer. A file of compiled code, named *file-name* TEXT, is written to your A-disk; it replaces any existing TEXT file of the same name.

FORTCOMP is implemented as a CMS EXEC program [5,10]. It is stored in file FORTCOMP EXEC on your A-disk.

§3.2 Input data

Three input files are required by the demand program. PATR FORTDATA is a set of annual patronages for city-pairs on the Corridor, as may be produced by the patronage program (§2). TITLE FORTDATA lists city abbreviations for use in printed output. SCHEDULE FORTDATA is an 11-city schedule for which demands are to be calculated by the methods described in [2, §§3.4-3.7]. Each line of this file describes a single run along the corridor, its direction indicated by the initial index-number: odd for southbound, even for northbound. The following numbers on the line are scheduled stopping times at successive stations, in hours and decimal fractions of an hour; an entry of -1.0 indicates that no stop is made.

You can make changes to these files by use of the CMS EDIT subsystem [4,9]. EDIT can also be employed to create new input files.

§3.3 Execution

To invoke the demand program, type

DEMAND *table-file*

where *table-file* is the name to be given to a file of tables that is produced by the program (see §3.4) for model generation by DATAMAT (§4).

After invocation, the demand program eventually types the following at your terminal:

INSERT PARAMETER CHANGES:

At this point you have the option of entering values for the following parameters:

Parameter name	Standard value	Interpretation
YEAR	1982	date used in printed output
CARCAP	75	car capacity, in passengers
TRNLTH	14	maximum number of cars in each section of a scheduled train
TABS	.TRUE.	table printout switch: if .TRUE., a full set of DATAMAT tables is filed; if .FALSE., the tables are not filed
BNDS	.TRUE.	bound-value printout switch: if .TRUE., you are prompted concerning bound-value decks as described below; if .FALSE., you are not prompted and a standard deck is produced (see also below)

To leave all parameters at their standard values, hit return. To change one or more parameters, type one space and then the following:

&CHANGE *parameter* = *value*, ... &END

where *parameter* names one of the parameters and *value* is the value you want to give to it. For example,

```
&CHANGE YEAR = 1990 &END
&CHANGE TRNLTH = 16, CARCAP = 80, TABS = .FALSE. &END
```

All parameters not listed remain at their standard values.

If parameter BNDS is .TRUE., you are also prompted:

```
ENTER 'DECK-NAME' AND FACTOR FOR DEMANDS:
```

Respond

'deck-name' factor

where *deck-name* is a unique identifying name (maximum 8 characters), and *factor* is a number by which demands will be multiplied before lower and upper bounds are determined. A set of bounds with the given *deck-name* is written, and the prompt is repeated. You can request decks for as many different factors as you wish. When you are finished, hit return in response to the prompt.

Example 2. A typical compilation and run of the demand program.

```
C>fortcomp demand
G1 COMPILER ENTERED
$SOURCE ANALYZED
$PROGRAM NAME = MAIN
*  NO DIAGNOSTICS GENERATED
SOURCE ANALYZED
$$PROGRAM NAME = BLK DA
*  NO DIAGNOSTICS GENERATED
    *STATISTICS*  NO DIAGNOSTICS THIS STEP
P;

C>fortcomp south
G1 COMPILER ENTERED
$SOURCE ANALYZED
PROGRAM NAME = SOUTH
*  NO DIAGNOSTICS GENERATED
SOURCE ANALYZED
$PROGRAM NAME = NORTH
*  NO DIAGNOSTICS GENERATED
SOURCE ANALYZED
PROGRAM NAME = INVERT
*  NO DIAGNOSTICS GENERATED
SOURCE ANALYZED
PROGRAM NAME = BLK DA
*  NO DIAGNOSTICS GENERATED
    *STATISTICS*  NO DIAGNOSTICS THIS STEP
P;

C>demand nec1982
EXECUTION BEGINS...

INSERT PARAMETER CHANGES:

> &change carcap=80, trnlth=13, tabs=.false. &end
$$$ENTER 'DECK-NAME' AND FACTOR FOR DEMANDS:
?
>'dem100' 1.0
ENTER 'DECK-NAME' AND FACTOR FOR DEMANDS:
?
>'dem110' 1.1
ENTER 'DECK-NAME' AND FACTOR FOR DEMANDS:
?
>
P;
```

```
C>type nec1982.data

*** TRAIN SIZE BOUNDS FOR 100.0% OF 1982 DEMANDS ***

NAME            DEM100
  1  2  1  2  1  2  2  3  2  4  5 10  4  7  6 12  4  8  6 11  4  7  5 10
  3  6  5  9  3  6  4  8  3  6  5  9  3  6  5  9  4  7  5 10  4  7  5 10
  4  8  5  9  4  7  4  8  3  5  3  6  2  4  3  5  2  3  2  4  2  4  1  2
  1  2  1  2  1  2  1  2  1  2  7 13 16 26  9 13 17 26  9 13 15 26  8 13
 13 13  7 13 12 13  7 13 10 13  6 12 10 13  6 11  9 13  6 11  8 13  5 10
  7 13  5  9  7 13  4  8  6 12  4  7  6 11  4  7  5  9  3  6  4  8  3  5
  4  8  2  4  1  2  2  3  4  8  4  8  2  3  3  6  2  3  3  6  2  4  3  6
  2  4  3  6  2  4  3  6  2  4  3  6  2  4  4  7  2  4  4  7  2  4  4  8
  3  6  5  9  3  5  4  7  2  4  3  5  2  3  3  5  2  3  2  4  1  2  2  3
  1  2  1  2  1  2  1  2  2  3  5  9  3  5  4  8  3  5  4  8  3  5  4  7
  2  4  4  7  2  4  3  6  2  4  3  6  2  4  3  6  2  4  4  7  2  4  4  7
  2  4  3  6  2  4  3  6  2  4  3  5  2  3  2  4  2  3  2  4  1  2  2  3
  2  4  2  3  1  2  5  9  6 11  4  7  6 11  4  8  6 12  4  8  7 13  5  9
  7 13  5  9  8 13  5 10  8 13  5 10  8 13  6 11  9 13  6 11 10 13  9 13
 18 26 11 13 16 26  7 13 11 13  6 11  8 13  5  9  7 13  4  8  6 12  4  7
  5 10  3  5  2  3  1  2  5  9  4  7  3  6  4  8  3  6  4  8  4  7  5  9
  4  7  4  8  3  6  4  8  3  5  4  8  3  6  5  9  3  6  5 10  4  7  6 11
  4  7  6 12  4  7  6 11  3  6  5  9  3  5  4  7  2  4  3  5  2  3  2  4
  1  2  1  2  1  2
ENDATA

*** TRAIN SIZE BOUNDS FOR 110.0% OF 1982 DEMANDS ***

NAME            DEM110
  1  2  1  2  1  2  2  4  2  4  6 11  4  8  7 13  5  9  6 12  4  8  6 11
  4  7  5 10  3  6  5  9  3  6  5  9  4  7  5 10  4  8  6 11  4  8  6 11
  4  8  5 10  4  7  5  9  3  6  4  7  3  5  3  5  2  4  2  4  2  4  2  3
  1  2  1  2  1  2  1  2  1  2  7 13 17 26 10 13 18 26 10 13 16 26  9 13
 14 26  8 13 13 13  7 13 11 13  7 13 10 13  6 12 10 13  6 12  9 13  5 10
  8 13  5 10  8 13  5  9  7 13  4  8  6 12  4  7  5 10  3  6  5  9  3  6
  5  9  2  4  2  3  2  3  4  8  5  9  2  4  4  7  2  4  4  7  2  4  3  6
  2  4  3  6  2  4  4  7  2  4  4  7  2  4  4  7  3  5  4  8  3  5  5  9
  3  6  5 10  3  5  4  8  2  4  3  6  2  4  3  5  2  3  2  4  2  3  2  3
  1  2  1  2  1  2  1  2  2  3  5 10  3  6  5  9  3  5  4  8  3  5  4  8
  3  5  4  7  3  5  4  7  3  5  4  7  2  4  4  7  2  4  4  7  2  4  4  7
  2  4  4  7  3  5  4  7  2  4  3  6  2  3  3  5  2  3  2  4  1  2  2  3
  2  4  2  3  1  2  5 10  6 12  4  8  6 12  4  8  7 13  5  9  8 13  5 10
  8 13  5 10  8 13  6 11  9 13  6 11  9 13  6 12 10 13  6 12 11 13 10 13
 20 26 12 13 18 26  8 13 12 13  6 12  9 13  5 10  8 13  5  9  7 13  4  8
  6 11  3  6  2  3  1  2  5 10  4  8  3  6  4  8  4  7  5  9  4  7  5 10
  4  7  5  9  3  6  4  8  3  6  4  8  3  6  5  9  4  7  6 11  4  8  6 12
  4  8  7 13  4  7  6 12  4  7  5 10  3  6  4  7  3  5  3  6  2  3  2  4
  1  2  1  2  1  2
ENDATA
P;
```

If parameter BNDS is .FALSE., you are not prompted. A single standard deck is filed, with *deck-name* BOUND and *factor* 1.0.

DEMAND is implemented as a CMS EXEC program [5,10]. It is stored in file DEMAND EXEC on your A-disk.

§3.4 Output

Principal output of the demand program is the file *table-file* DATA, where *table-file* is as specified in the DEMAND command. If parameter TABS was .TRUE., this file contains tables employed by DATAMAT to generate a model (§4) and to form tables of results (§7). The tables in the file include the following:

Table name	Contents
M:SCHEDULE	a representation of the full schedule as read by the demand program
G:ACTIVE	a tabulation of the "active" times at which trains enter or leave each city (a 1 or 2 in the table indicates that a city is active, a 0 indicates that it is not); the column INDEX is the number of minutes elapsed in the day, since 1:00, for each active time
G:TRAVEL	distances between pairs of terminals
M:HEADING	direction indicator: N means north, S south
G:CARMILE	equivalent car-mile demands: for each train, the number of passenger-miles of service demanded, divided by the passenger capacity per car
G:DEMAND	car requirements: the minimum number of cars required to meet effective demand for each train [2, §3.6]
M:NORTH	an alternative representation of the schedule of northbound trains, employed in producing the NSTRAINS table of results (§6.1)
M:SOUTH	equivalent of M:NORTH for southbound trains

Table-file DATA also contains the requested decks of bound values, any of which may be read into the model by SESAME after it is generated (§5).

The demand program also writes a summary of input and output data, and a full listing of *table-file* DATA, to your virtual printer.

§4 GENERATING AN LP MODEL

For any Northeast Corridor schedule, the associated transshipment LP [2, §1.4] may be generated by use of the DATAMAT subsystem of SESAME [1]. This section describes programs and commands employed in model generation.

Example 3 shows a terminal session in which a typical model is generated.

§4.1 Initialization

To initialize DATAMAT and the files employed in model generation, type

DATAMAT *model-file table-file*

Model-file is the name (maximum 7 characters) that will identify the file that holds the generated model. *Table-file* should refer to an output file, named *table-file* DATA, from the demand program (§§3.3-3.4). This file must be the result of a demand-program run that had the schedule of interest as input, and parameter TABS equal to .TRUE. .

The DATAMAT command invokes both SESAME and DATAMAT, performing routine initializations for both. The tables on *table-file* DATA are then read by DATAMAT, translated into an internal form, and stored on a file named

T*model-file* MPFILE

(That is, the name of this file is *model-file* prefixed by the letter T.) A list of the tables read is printed at your terminal. If there is already a file T*model-file* MPFILE on your A-disk, existing tables on it are replaced by the new ones just read.

Example 3. Generation of a typical model, using the DATAMAT and TRAINS commands.

```
C>datamat basemod nec1982

SESAME V9.2

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  > > > > $>

> M:READ      =TYPE
   DEMAND   =GN
   CAPMILE  =GN
   SCHEDULE=MN
   ACTIVE   =GN
   TPAVEI   =GN
   HEADING  =MN
   NORTH    =MN
   SOUTH    =MN
> >
8      TABLES ENFILED
> >trains basemod 20
$$$$$
N:CITY      = WA
$$$$$$
N:CITY      = PH
$$$$$$$
N:CITY      = NY
$$$$
N:CITY      = BO
$$$$$$
N:PAIR      = WAPH
$$
N:PAIR      = PHNY
$
N:PAIR      = NYBO
$$
N:PAIR      = BONY
$$
N:PAIR      = NYPH
$$
N:PAIR      = PHWA
$$
```

```
ENELLING MODEL
$$$$$$$$$$$$
 ROWS      COLUMNS  RHS       RANGES    BOUNDS    CUB-S    STR COEFS:DENSITY  INDIRECT
 531       1275     1         4         1         0         332E      .0040126? 041
> >quit

TERMINATING DATAMAT, RETURN TO SESAME
QUIT

SESAME COMMAND:  >quit
R;

C>
```

If you omit *table-file* from the DATAMAT command, SESAME and DATAMAT are still initialized but no tables are read and translated. This option may be useful if you have already read the tables into T*model-file* MPFILE but wish to regenerate the model. If you type only DATAMAT, the model file is given the SESAME default name MODELS, and any tables subsequently filed go to the default table file TABLES MPFILE.

When execution of the DATAMAT command ends, you are left in the DATAMAT interactive environment. You may immediately type any DATAMAT commands.

The DATAMAT command is implemented as a CMS EXEC program that stacks SESAME and DATAMAT command lines for execution. It is stored in file DATAMAT EXEC on your A-disk.

§4.2 Model Generation

Once the initialization steps are complete, generate an LP by typing

TRAINS *model delay*

Model is the name by which the generated model will be known; it may be the same as *model-file* (§4.1). *Delay* is the turnaround delay, in minutes, to be reflected in the model network [2, §1.5].

Columns for variables that represent cars in storage are generated first, for one terminal at a time. A message of the form

N:CITY = *city*

is printed at your terminal each time generation for a new city begins; *city* is either WA (Washington), PH (Philadelphia), NY (New York), or BO (Boston).

Columns for variables that represent cars in trains are generated next, for one segment and direction at a time. Each time a new segment is begun, you receive at your terminal a message of the form:

N:PAIR = *city*1*city*2

where *city*1 and *city*2 are abbreviations as given above. This indicates that columns for trains from *city*1 to *city*2 are being generated; for example, PHNY refers to trains from Philadelphia to New York.

When all columns are generated, you receive the message

ENFILING MODEL

The generated model is written to a file named *model-file* MPFILE, which is created if it did not previously exist. If there is an existing LP named *model* on *model-file* MPFILE, it is replaced by the newly-generated one and a message to this effect is printed. When enfiling is completed, DATAMAT prints a summary of model statistics.

Finally, you are returned to the DATAMAT interactive environment. You may then generate another model (with a different delay, perhaps), or you may leave DATAMAT as described below.

The TRAINS command is implemented as a program in the DATAMAT macro language. It is stored in file TRAINS DATAMAC on your A-disk.

§4.3 Leaving DATAMAT

To leave the DATAMAT interactive environment at any point, type

QUIT

This returns you to the SESAME environment, where you may proceed to solve and analyze any model that you have just generated (see §§5, 6). If you do not want to use SESAME at this point, wait for a prompt of

SESAME COMMAND:

and then type QUIT again to return to the CMS environment.

§5 SOLVING A MODEL

A previously-generated corridor LP (§4) can be solved by use of SESAME's simplex algorithms. This section describes the commands and programs employed in initializing SESAME, choosing a set of bounds, and iterating to a solution from an all-slack or previously-saved basis.

Example 4 reproduces a typical terminal session in which a model is solved.

§5.1 Initialization

To initialize SESAME for a particular corridor LP, type the following command from the CMS environment:

SESMODEL *model-file*

Model-file should be the name of a file on which a previously-generated LP was stored, as described in §4. When initialization is complete, you are left in the SESAME interactive environment, and the following prompt is typed at your terminal:

SESAME COMMAND:

When you enter the SESAME environment immediately after generating a model (§4.3), SESAME is already initialized to solve the model and the SESMODEL command is not needed.

SESMODEL is implemented as a CMS EXEC program that stacks SESAME commands for execution. It is stored in the file SESMODEL EXEC on your A-disk.

§5.2 Selection of bound values

You must specify upper and lower limits for the model's train-size variables by choosing one of the bound decks filed for the model by the demand program (§§4.2-4.3).

To choose an initial bound deck, first type the following SESAME command:

$INFILE = *table-file* $DECKNM = *deck-name*

Table-file should name the output file on which the bound decks were written by the demand program, while *deck-name* should identify the bound deck that you want to use. When SESAME prompts for another command, type

RUN BOUNDS DEFINE

This command translates the bound deck and writes it to an internal region. It then sets up the LP matrix in the form required by SESAME's simplex algorithm, incorporating the bound-deck values for upper and lower limits.

You can change to a different set of bounds at any point in the SESAME session. To do this, first type

$DECKNM = *deck-name*2

where *deck-name*2 identifies a new deck of bound values that you want to incorporate in the model. Then type

RUN BOUNDS CHANGE

to set up the LP matrix again with the new bound values replacing the old ones. You can repeat this procedure to change bounds as many times as desired during a single SESAME session.

Both RUN BOUNDS DEFINE and RUN BOUNDS CHANGE execute programs of simple SESAME commands. These programs are stored in file BOUNDS DATA on your A-disk.

§5.3 Choice of an objective

You can minimize any of the following three objectives defined in Volume 1 of this report [2, §2.1]:

Name of objective	Defined in Volume 1 as	Represents
MINCARS	Z_{CAR}	number of cars
MINMILES	Z_{MILE}	car-miles run per day
MINTURN	Z_{TURN}	total number of times the direction of any car is changed

To specify the objective to be minimized, type

$OBJ = *objective-name*

The *objective-name* must be the name of one of the three objectives (MINCARS, MINMILES, MINTURN). You can change your selection of objective at any time by repeating this command with a different *objective-name*.

§5.4 Minimization from an all-slack basis

When SESAME is solving an LP, it maintains an internal region in which a current basis is stored. The simplex algorithm changes the current basis, one pivot at a time, until it produces a current basis that is optimal.

When RUN BOUNDS DEFINE or RUN BOUNDS CHANGE is executed (§5.2) the current basis is initialized to one composed entirely of slack variables (also known as artificial variables). You can start the simplex algorithm at this all-slack basis by typing

CALL ITERATE

Phase 1 of the algorithm is first applied to find a feasible basis; then phase 2 is employed to find an optimal basis for the selected objective. Messages FEASIBLE SOLUTION and OPTIMAL SOLUTION indicate successful conclusion of phase 1 and phase 2, respectively.

Normal output from CALL ITERATE includes one "log" line summarizing each iteration, and two summary lines for each reinversion of the basis. You can save time by suppressing printing of most or all of the log lines. To print only every *n*th log line, type the following command before CALL ITERATE:

$FLOG = *n*

Example 4(a). Solution of the model generated by DATAMAT and TRAINS from an all-slack basis.

```
C>sesmodel basemod

SESAME V9.2

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  >$infile = nec1982  $decknm = dem100

SESAME COMMAND:  >run bounds define
$
1    REVERSED OR TRIVIAL BOUND,RANGES.  FIXED AT LI

SESAME COMMAND:  >$obj = mincars

SESAME COMMAND:  >$flog = 0

SESAME COMMAND:  >call iterate
$
INVERT IN AT ITERATION 36     NO. ETA COLS... . 35  ELEMENTS..... 115   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS..... 35  ELEMENTS..... 97    RECS..... 1
$
INVERT IN AT ITERATION 61     NO. ETA COLS..... 60  ELEMENTS..... 197   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS. ... 60  ELEMENTS... . 169   RECS..... 1
$$
INVERT IN AT ITERATION 98     NO. ETA COLS..... 97  ELEMENTS..... 338   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS..... 96  ELEMENTS..... 255   RECS..... 1
$$
INVERT IN AT ITERATION 144    NO. ETA COLS.... 139  ELEMENTS..... 532   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS. .. 137  ELEMENTS..... 358   RECS..... 1
$$
INVERT IN AT ITERATION 182    NO. ETA COLS.. . 174  ELEMENTS..... 735   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS.... 168  ELEMENTS..... 439   RECS.. .... 1
$$$$
INVERT IN AT ITERATION 226    NO. ETA COLS.... 212  ELEMENTS..... 889   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS.... 201  ELEMENTS..... 522   RECS..... 1
$$$$
INVERT IN AT ITERATION 275    NO. ETA COLS.... 250  ELEMENTS ... 1065   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS.... 237  ELEMENTS. ... 612   RECS..... 1
$$$$
```

```
INVERT IN AT ITERATION 326     NO. ETA COLS.... 286  ELEMENTS.... 1226   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 275  ELEMENTS .... 708   RECS..... 1
$$$$
INVERT IN AT ITERATION 359     NO. ETA COLS.... 307  ELEMENTS.... 1424   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 294  ELEMENTS..... 757   RECS..... 1
$$$$
INVERT IN AT ITERATION 403     NO. ETA COLS.... 336  ELEMENTS...  1545   RECS.... .. 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 322  ELEMENTS.. ...821   RECS..... 1
$$$$
INVERT IN AT ITERATION 439     NO. ETA COLS.... 358  ELEMENTS.... 1723   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 351  ELEMENTS..... 892   RECS..... 1
 $
INVERT IN AT ITERATION 460     NO. ETA COLS .... 372  ELEMENTS.... 1972   RECS.... .. 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS...  360  ELEMENTS..... 933   RECS..... 1
$$$$$$
INVERT IN AT ITERATION 536     NO. ETA COLS.... 445  ELEMENTS.... 1882   RECS..... 1
$INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 430  ELEMENTS.... 1086   RECS..... 1
$$$
INVERT IN AT ITERATION 564     NO. ETA COLS.... 460  ELEMENTS.... 2224   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 452  ELEMENTS.... 1132   RECS..... 1
$$$
INVERT IN AT ITERATION 589     NO. ETA COLS.... 477  ELEMENTS.... 2267   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS .... 467  ELEMENTS.... 1164   RECS..... 1
$$$$$
INVERT IN AT ITERATION 658     NO. ETA COLS.... 536  ELEMENTS.... 2333   RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 523  ELEMENTS.... 1292   RECS..... 1

$$
 FEASIBLE SOLUTION
$
INVERT IN AT ITERATION 682     NO. ETA COLS.... 546  ELEMENTS.... 2608   RECS... /. 2
$INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 527  ELEMENTS.... 1305   RECS..... 1
$
 OPTIMAL SOLUTION

SESAME COMMAND:  >print $funct $iterno
          164.00000
          603

SESAME COMMAND:  >call save name = start100
```

Example 4(b). Solution of the same model for a different set of bounds, starting from the previously attained optimum.

```
SESAME COMMAND:  >$decknm = dem105

SESAME COMMAND:  >run bounds change
*
1    REVERSED OR TRIVIAL BOUNDS,RANGES.  FIXED AT 11

SESAME COMMAND:  >call restore name = start100

SESAME COMMAND:  >$flag = 1

SESAME COMMAND:  >call iterate dual 500

INVERT IN AT ITERATION 0       NO. ETA COLS. .... 0   ELEMENTS..... 0    RECS..... 1
INVERT OUT: NO. ROWS = 531     NO. ETA COLS.... 527   ELEMENTS.... 1304   RECS..... 1

USING DUAL ALGORITHM.

 ITER.NO      SUM INF COL. IN   COL. OUT   INTERNL.DJ    FUNCTIONAL  PINFS CINFS DINFS ETA.CNT
    1       6.0000000    587      1724          .        172.00000     5     0     0     1337
    2       4.9999981   1513      1671          .        172.00000     5     0     0     1388
    3       2.0000000   1669      1691          .        172.00000     2     0     0     1443
    4       2.0000000   1668      1400     1.0000000     173.00000     2     0     0     1492
    5       1.0000000    620      1748          .        173.00000     1     0     0     1513
    6             .      711      1615          .        173.00000     0     0     0     1535

 FEASIBLE SOLUTION

 OPTIMAL SOLUTION

SESAME COMMAND:  >call save name = start105

SESAME COMMAND:  >quit
R;
C>
```

To suppress all log lines, type

```
$FLOG = 0
```

When an optimal solution is reached, you can print the value of the objective and the number of iterations performed, by typing

```
PRINT $FUNCT $ITERNO
```

§5.5 Saving and restoring bases

You can file the current basis at any point by typing

```
CALL SAVE NAME = basis-name
```

where *basis-name* is any identifier (8 characters or less) that you want to give to the basis. If you previously stored a basis under *basis-name* for the LP you are currently solving, the old basis is replaced by the new one. (Bases that you may have stored for other LP models are unaffected.)

To recall a basis and make it the current basis, type

```
CALL RESTORE NAME = basis-name
```

where *basis name* is the identifier you gave to the basis when you saved it. You may restore only bases that were saved for the model you are currently solving; bases saved for one model cannot be restored for another.

For the most part, you will save and restore optimal bases. A restored basis can serve as a starting point for a different optimization (see §5.6 below), or for an analysis of the tradeoff between two objectives (§6). Saved bases are also employed by programs that generate tables of results (§7).

All saved bases are stored in a special compact format in file MAPSFILE MPFILE on your A-disk.

§5.6 Minimization from a previously attained basis

For one model, there may be many combinations of bounds and objectives for which you want optimal solutions. It is not necessary, however, to start from an all-slack basis to find each optimum. Instead you can save

the optimal basis for one bound set and objective, and later use it as a starting point for iterating to optima for other bounds and objectives. Starting from a previously-attained optimum often involves far fewer iterations than starting from an all-slack basis, because optimal solutions of interest are usually fairly close to each other.

If you only want to optimize a new objective, for the same set of bounds, simply change your choice of objective (by $OBJ = *objective-name*) and invoke the simplex algorithm (by CALL ITERATE). It is not necessary to use CALL RESTORE if there is already an optimal current basis.

If you want to optimize the same objective for a different set of bounds, follow the instructions given above for RUN BOUNDS DEFINE (if you are beginning a new session) or RUN BOUNDS CHANGE (if you are continuing a session). Then use CALL RESTORE to recall some previously-found optimal basis. To find the new optimum, invoke the dual simplex algorithm by typing

CALL ITERATE DUAL *n*

where *n*, a limit on primal infeasibilities, is a large number (say, 500).

To optimize for a different objective *and* a different set of bounds, follow the instructions in the previous paragraph. The dual algorithm is not especially useful in this case, however; invoke the primal algorithm instead by typing CALL ITERATE as before.

§5.7 Leaving SESAME

To terminate a SESAME session and return to the CMS command environment, type QUIT.

§6 FINDING TRADEOFFS BETWEEN OBJECTIVES

SESAME's algorithm for parametric programming on the objective can be used to compute optima for "total cost" objectives of the form $Z_1 + \rho Z_2$, as described in Volume 1 of this report [2, §§2.3-2.5]. In general, there exist certain critical values of ratio ρ at which there is a tradeoff between Z_1 and Z_2: the former can increase at the optimal solution while the latter decreases. Commands and programs for finding the critical ratios and related optimal solutions are described in this section.

Example 5 shows part of a SESAME terminal session in which tradeoffs are computed.

§6.1 Initialization

Choose a set of bounds and an objective function, following the directions in §§5.2-5.3. Since the objective is chosen by typing $OBJ = *objective-name*, it is referred to here as $OBJ.

Iterate to or restore an optimal basis for $OBJ, as directed in §§5.4-5.6.

Choose another objective as the second component of total cost, by typing

$COBJ = *objective-name*2

This objective is referred to as $COBJ.

§6.2 Finding all critical ratios

To find all critical ratios ρ for $OBJ + ρ $COBJ, type

RUN TRADEOFF ALL

The output identifies the critical ratios, and prints the optimal values of $OBJ and $COBJ in the interval of ρ preceding each critical ratio, as shown in Example 5.

Example 5. Application of TRADEOFF command to objectives MINCARS ($OBJ) and MINMILES ($COBJ).

```
C>sesmodel basemod

SESAME V9.2

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  >$infile = nec1982  $deckpm = dem100

SESAME COMMAND:  >run bounds define
$
1    REVERSED OR TRIVIAL BOUND,RANGES.  FIXED AT 11

SESAME COMMAND:  >$obj = mincars

SESAME COMMAND:  >call restore name = start100

SESAME COMMAND:  >$cobj = minmiles

SESAME COMMAND:  >run tradeoff all
$$$$$$$$$$$$$$$PARAMETER HAS REACHED $PARAMAX -- TERMINATING

SOLUTION

         OPTIMAL SOLUTION AT ITERATION NUMBER 104

         ...NAME...    ...ACTIVITY...       DEFINED AS

         FUNCTIONAL        164.00001        MINCARS  + .1000E-09 * MINMILES
         RESTRAINTS                         RHS
         BOUNDS....                         BOUND
         RANGES....                         RANGE

ROWS SECTION

  NUMBER   ...ROW..  AT    ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY
       1  MINCARS   BS       164.00000       -164.00000        NONE            NONE          1.0000000
       2  MINMILES  BS       135978.00       -135978.00        NONE            NONE          .1000E-09
$PARAMETER HAS REACHED $PARAMAX -- TERMINATING
```

```
SOLUTION

        OPTIMAL SOLUTION AT ITERATION NUMBER 105

        ...NAME...     ...ACTIVITY...        DEFINED AS

        FUNCTIONAL        457.05605          MINCAPS  + .00215517 * MINMILES
        RESTRAINTS                           RHS
        BOUNDS...                            BOUND
        RANGES....                           RANGE

ROWS SECTION

  NUMBER  ...ROW..  AT    ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY
       1  MINCAPS   BS       164.00000      -164.00000         NONE            NONE          1.0000000
       2  MINMILES  BS       135978.00      -135978.00         NONE            NONE          .00215517
PARAMETER HAS REACHED $PARAMAX -- TERMINATING


SOLUTION

        OPTIMAL SOLUTION AT ITERATION NUMBER 113

        ...NAME. .     ...ACTIVITY...        DEFINED AS

        FUNCTIONAL        466.17335          MINCAPS  + .00222222 * MINMILES
        RESTRAINTS                           RHS
        BOUNDS....                           BOUND
        RANGES....                           RANGE

ROWS SECTION

  NUMBER  ...ROW..  AT    ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY
       1  MINCAPS   BS       167.00000      -167.00000         NONE            NONE          1.0000000
       2  MINMILES  BS       134628.00      -134628.00         NONE            NONE          .00222222
PARAMETER MAY INCREASE WITHOUT LIMIT -- TERMINATING
```

Example 5 (continued).

```
SOLUTION

         OPTIMAL SOLUTION AT ITERATION NUMBER 131

         ...NAME...     ...ACTIVITY...       DEFINED AS

         FUNCTIONAL        914.03335         MINCAPS  + .00555556 * MINMILES
         RESTRAINTS                          RHS
         BOUNDS ...                          BOUND
         RANGES....                          RANGE

ROWS SECTION

  NUMBER   ...ROW..  AT    ...ACTIVITY...  SLACK ACTIVITY  ..LOWER LIMIT.  ..UPPER LIMIT.  .DUAL ACTIVITY
       1  MINCAPS   BS        185.00000      -185.00000         NONE            NONE          1.0000000
       2  MINMILES  BS        331388.00      -331388.00         NONE            NONE          .00555556
PARAMETER MAY INCREASE WITHOUT LIMIT -- TERMINATING

SESAME COMMAND:  >call save name = minmiles

SESAME COMMAND:  >quit
R;

C>
```

In the first optimal solution printed, the objective values are min $OBJ and min $COBJ|$OBJ. In the last solution printed, the objective values are min $COBJ and min $OBJ|$COBJ. Other solution values are intermediate tradeoffs between $OBJ and $COBJ that are optimal at various values of ρ.

§6.3 Finding the next critical ratio

To find the first critical ratio only, initialize as described above and type

RUN TRADEOFF STEP

At this point you can use CALL SAVE (§5.5) to store the current basis -- which is optimal for min $OBJ and min $COBJ|$OBJ -- before continuing.

To find succeeding critical ratios one at a time, continue typing RUN TRADEOFF STEP. This allows you to save the optimal solution at each new ratio. You may also type RUN TRADEOFF ALL at any point to list all remaining critical ratios without stopping.

Both RUN TRADEOFF ALL and RUN TRADEOFF STEP execute programs of simple SESAME commands. These programs are stored in the file TRADEOFF DATA on your A-disk.

§6.4 Fixing one objective at its optimal value

To compute conditional optima involving all three objectives, you must fix one at its optimal value. You can then apply RUN TRADEOFF STEP or RUN TRADEOFF ALL to the other two objectives.

To fix an objective, first select the one to be fixed by typing

$RANGE = *objective-name* $XFSW = 3

Next, you must tell SESAME what the optimal value of the named objective is. Type

CALL VALUES CHANGE

Wait for a greater-than sign (>) to be typed at your terminal, then enter

objective-name = *value*

where *value* is the optimal value for the named objective. Wait for another greater-than sign, and respond

QUIT

This returns you to the regular SESAME environment (where you are prompted SESAME COMMAND:).

Once these steps are completed, you must execute RUN BOUNDS DEFINE or RUN BOUNDS CHANGE (§5.2) to set up the LP matrix with the fixed objective. Then you can work with the other two objectives as described above, with the difference that all optima that you find are conditional on the fixed objective. For example, if you fix MINCARS and then minimize MINMILES, the optimal basis is a solution to min MINMILES|MINCARS. If you then make MINTURN the $COBJ objective, and execute RUN TRADEOFF STEP, the resulting basis solves min MINTURN|MINMILES|MINCARS.

You can also follow the steps outlined above to fix an objective at a value other than its minimum. However, in such a case the existence of an optimal integral basic solution cannot be guaranteed.

§7 TABULATING RESULT VALUES

Various tabular summaries of the LP solution at any basis can be printed by use of SESAME and DATAMAT. A program for producing these summaries is described below.

The program assumes that *model-file*, the name of the file on which your model is stored (§4.1), is the same as *model*, the name you gave to the model on the file (§4.2). If the two are different, you must rename the model and table files by typing the following CMS commands:

RENAME *model-file* MPFILE A *model* MPFILE A
RENAME T*model-file* MPFILE A T*model* MPFILE A

In the sequel, the term *model-name* is used to refer to both the name of the model file and the name of the model on the file for which results are being tabulated.

Terminal output from the result-summary program is shown in Example 6.

§7.1 Requesting result tables

To print result tables for a particular model and basis, type

RESULTS *model-name basis-name table ...*

where *model-name* identifies the model, and *basis-name* is the name of some basis saved (§5.5) for that model. *Table* identifies a table that you want printed, as follows:

Table	Contents of printed table
OBJ	values of objectives MINCARS, MINMILES, and MINTURN
TRNSIZE	number of cars in each scheduled train, arranged by segment and time

Table	Contents of printed table
NSTRAINS	number of cars in each scheduled train, arranged as the trains appeared in the original schedule input to the demand program (§3.2)
TRNDIST	distribution of train sizes
IDLE	number of cars in storage at each city and time
LOADFACT	load factor of each train, load factor over each segment in each direction, and system load factor

You can request as many tables as you want in a single RESULTS command. It is more efficient to request numerous tables in one RESULTS command than to generate the tables by individual executions of the command.

The tables that RESULTS generates are sent as one file to your virtual printer.

The RESULTS command is implemented as a CMS EXEC program [5,10] that stacks SESAME and DATAMAT commands for execution. It is stored in file RESULTS EXEC on your A-disk. For each table that is printed, RESULTS calls a program written in the DATAMAT macro language; these programs are stored in file RESULTS DATARUN on your A-disk.

§7.2 Correcting load factors for scaled demands

RESULTS normally computes the LOADFACT table using the demand for each train as filed by the demand program (§3.4). Consequently, if you are using a set of bounds for other than 100% of annual patronage (§3.3), the load factors will be incorrect.

To remedy this problem, invoke RESULTS as follows:

RESULTS *model-name basis-name factor table ...*

where *factor* is the number by which demands were scaled in producing a set of bounds, and *basis-name* is a basis that was found using that set of bounds. (RESULTS automatically retrieves the proper set of bounds before calculating result values. It can do this because the RUN BOUNDS and RUN TRADEOFF programs store the names of the file and deck where the bounds are located, in such a way that these names are automatically saved with every basis.)

Example 6. Terminal output from RESULTS command. The tables produced by RESULTS are printed offline.

```
C>results basemod mc100 obj trnsize trndist

SESAME V9.2

SESAME COMMAND:
 OPTIMAL SOLUTION FOR: MODEL BASEMOD -- CASE MC100

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  >
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  $
SESAME COMMAND:
SESAME COMMAND:
1    REVERSED OR TRIVIAL BOUND,RANGES.  FIXED AT II

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:
INVERT IN AT ITERATION 0      NO. ETA COLS..... 0  ELEMENTS..... 0   RECS..... 1
INVERT OUT: NO. ROWS = 531    NO. ETA COLS.... 527  ELEMENTS.... 1308  RECS..... 1

SESAME COMMAND:
SESAME COMMAND:
SESAME COMMAND:  TERMINATING DATAMAT, RETURN TO SESAME
ENDATA
DATARUN FILE TERMINATED. REVERT TO TERMINAL

SESAME COMMAND:  TERMINATING DATAMAT, RETURN TO SESAME
ENDATA
DATARUN FILE TERMINATED. REVERT TO TERMINAL

SESAME COMMAND:  $TERMINATING DATAMAT, RETURN TO SESAME
ENDATA
DATARUN FILE TERMINATED. REVERT TO TERMINAL

SESAME COMMAND:  P;

C>
```

APPENDIX A

LIST OF REQUIRED FILES

The following CMS files are required for operation of the programs described in this report:

File-name	File-type	Contents
BOUNDS	DATA	programs run by SESAME commands RUN BOUNDS DEFINE and RUN BOUNDS CHANGE (§5.2)
TRADEOFF	DATA	programs run by SESAME commands RUN TRADEOFF ALL (§6.2) and RUN TRADEOFF STEP (§6.3)
TRAINS	DATAMAC	DATAMAT program that generates an LP matrix (§4.2)
RESULTS	DATARUN	DATAMAT programs that generate tables of results (§7.1)
DATAMAT	EXEC	program to initialize SESAME and DATAMAT prior to generation of an LP matrix (§4.1)
DEMAND	EXEC	CMS EXEC routine that runs the demand program (§3.3)
FORTCOMP	EXEC	CMS EXEC routine that compiles FORTRAN files comprising the demand program (§3.1)
GETD	EXEC	utility program that allocates temporary disk (D-disk) space; called by several other programs
MODAL	EXEC	CMS EXEC routine that runs the patronage program (§2.3)
PF	EXEC	utility program that sends a listing of a file to the virtual printer; called by several other programs
PLICOMP	EXEC	CMS EXEC routine that compiles PL/I files comprising the patronage program (§2.1)
PROFILE	EXEC	initialization commands for CMS (§1.1)

File-name	File-type	Contents
RESULTS	EXEC	program to print tables of result data (§§7.1-7.2)
SESMODEL	EXEC	program to initialize SESAME prior to solution or analysis of a model (§5.1)
TESTNUM	EXEC	utility program that determines whether an argument is a number; called by RESULTS EXEC
DEM	FORTRAN	
DEMAND	FORTRAN	
MAXLK	FORTRAN	source code for demand program (§3.1)
PRNT	FORTRAN	
SOUTH	FORTRAN	
PATR	FORTDATA	
SCHEDULE	FORTDATA	input files for demand program (§3.2)
TITLE	FORTDATA	
DMDOUT	PLI	
MODAL	PLI	source code for patronage program (§2.1)
SNOW	PLI	
FORE2A	PLIDATA	
FPOPINC	PLIDATA	
NECDICT	PLIDATA	
NECLINKS	PLIDATA	input files for patronage program (§2.2, Appendix B)
NEC1974B	PLIDATA	
PARMS	PLIDATA	

APPENDIX B

REQUIRED INPUT FILES FOR THE PATRONAGE PROGRAM

The patronage program invoked by the MODAL command (§2) requires six input files in various special formats. This appendix reproduces documentation for the program supplied by Peat, Marwick, Mitchell and Company, including detailed input specifications and summaries of the operation and output of the program.

All references to files in this appendix employ their logical file names, as assigned in the PL/I code. The MODAL command associates each actual input file with the appropriate logical file as follows:

Logical name	*Actual input file*
SYSIN	*PARMS PLIDATA*
CPCODES	*NECDICT PLIDATA*
LINK	*NECLINKS PLIDATA*
CITYPI	*FPOPINC PLIDATA*
CTYINF1	*NEC1974B PLIDATA*
CTYINF2	*FORE2A PLIDATA*

Output to logical file SYSPRINT appears at the terminal, while the contents of logical files PRINTER and LKPRINT are sent to the virtual printer. The contents of logical file PLIDUMP are discarded. (Changes in any of these assignments may be effected by modifications to the MODAL program, stored in file MODAL EXEC.)

INPUT FILES

The following data files are required to input data:

NAME	DESCRIPTION
SYSIN	Specifies options for program and parameters that apply to all city pairs.
CPCODES	Contains the city pair equivalence table indicating the city numbers associated with each city pair. These are in card images.
LINK	Contains the cities between which link information will be printed.
CITYPI	Contains the population and income information for each city.
CTYINF1	Contains city pair specific data. These are in card images with one record for each mode of each city pair. This file contains the base year or calibration data.
CTYINF2	Contains city pair specific data. This file is in the same format as CTYINF1, but contains forecast data.

OUTPUT FILES

The following files are output:

NAMES	DESCRIPTION
SYSPRINT	The System Printer contains a print of the SYSIN file, program error messages, and "debug" printing.
PLIDUMP	Contains a core dump when abnormal termination occurs. It also contains a storage management report, which may be of interest to the programmer. This is a print file.
PRINTER	Contains the detail results for all city pairs. This is a print file.
LKPRINT	Contains the link data summary and the corridor summary. This is a print file.

Input Data Specification

SYSIN

Data on the sysin file is in a free format consisting of the data name followed by "=" followed by a value followed by a comma (e.g. BASEYR = 1974,). Following the last value is a semi-colon instead of a comma. The card images may be packed with as many items as the user wishes. Array elements must be specified one element at a time (e.g. A1 (1,1) = 1, A1 (1,2)= 4,..). If the default value for an item is satisfactory, the item may be omitted.

VARIABLE	DESCRIPTION
MAX. LINKS	Maximum number of links to be processed. This must be at least as great as the number of links in the LINK file.
MAX. CTY_PAIRS	The largest city pair number. This is also the number of city pairs if they are numbered sequentially.
CALIBRATION	'YES' the base year data is calibrated. 'NO' the base year data is not calibrated. The default value is 'YES'.
PRINTSW	'DETAIL' prints city pair detail, link and link summary reports. 'SUMMARY' prints link and summary reports only.
BUSSW	1-one purpose modal split. 2-two purpose modal split with the percent of total for purpose one specified in the input data (see CTYINF1).
INCSW	This variable is used for forecasting income. 'LINEAR' results in a linear income growth factor. 'EXP' results in an exponential income growth factor.
POPSW	This variable is used for forecasting population. 'LINEAR' results in a linear population growth factor. 'EXP' results in an exponential growth factor.
INCN1	The value specified is a multiplicative constant for forecasting trips. The value used in the N.E. corridor is 1.77.

NAME	DESCRIPTION
INCN2	The value specified is a subtraction constant for forecasting trips. The value used is 203.
BASEYR	The year for the base data.
LOWYR	The first forecast year (LOWYR $\geq$ BASEYR).
HIGHYR	The last forecast year (HIGHYR $\geq$ LOWYR).
AUTOCOST	The auto cost per mile is specified in dollars. (e.g. 3¢/mile is coded: AUTOCOST = .03,)
OCC	This is a two value variable specifying the auto occupancy factor. OCC(1) is for purpose 1, OCC(2) is for purpose 2.
INDUCE-SUPPRESS	'YES' - the demand for a mode, city pair is factored by the ratio of the new omega to the omega exponentiated to the BETA power. Demand = Demand((new omega/old omega)**BETA) 'NO' - demand is not factored. 'NO' is assumed.
BETA	The value used when INDUCE-SUPPRESS is 'YES'. 0.6 is assumed

RN ?

XKA ?

The following variables are coefficients and factors for the modal split model. "MD" in the subscript indicates a value is required for each mode (1,2,3, 4). "PR" in the subscript indicates that a value is required for each purpose.

TOTA (md,pr)	This specifies an overall or total factor applied to the model. If CALIBRATION = 'YES', this variable has no effect.
A2 (md,pr)	This specifies the traveltime exponent.
A3 (md,pr)	This specifies the cost exponent.
A4 (md,pr)	This specifies the frequency exponent.
KAY (md,pr)	This specifies any exponential factor (i.e., 'KAY' factor).
MODE_FACTOR (md)	This specifies?
LHC_FACTOR (md)	This specifies the line haul cost factor.
LHT_FACTOR (md)	This specifies the line haul time factor.
FREQ_FACTOR (md)	This specifies the frequency factor.
ACC-FACTOR(md)	This specifies the access cost factor. 1 is assumed.
ACT-FACTOR(md)	This specifies the access time factor. 1 is assumed.

The following variables are available to assist the programmer in finding errors (bugs) in the program:

CPSW — When the city pair number is equal to the value of specified in CPSW various program variables are printed on SYSPRSNT. The default is 0 (no print).

ITERCNT — If ITERCNT is greater than the number of forecast years plus two then most variable and array values are printed at the end of execution. The default value is 0 (no print).

DUMPSW — 1-print PLI dump of all storage areas if an error occurs.

0-do not print the PLI dump on error. The default value is 0 (no print).

CPCODES — Data on the CPCODES file are in a fixed format. One 80 character record (such as punch card) is required for each city pair.

CHARACTER NUMBERS	VARIABLE	DESCRIPTION
1	FIL	filler-may be blank
2-4	E-CTYPR	city pair number
5-7	EQUIVI	origin city number
8-10	EQUIV2	destination city number
11-80	E_CTYPRNAME	The city pair name in characters. The first 40 characters will appear in the "detail" printing. If blank, the city pair name specified in "CTYINFI" will be used.

LINK

Data on the LINK file are in a fixed format. One 80 character record (such as punch card) is required for each link to be summarized. The number of link records should be the same as the value of MAX.LINKS.

CHARACTER NUMBERS	VARIABLE	DESCRIPTION
1-2	L_CODE	Link code or link number.
3-4	L_CTY1	Origin city number.
5-6	L_CTY2	Destination city number.
7-9	L_DISTI	Distance of link for mode 1.
10-12	L_DIST2	Distance of link for mode 2.
13-15	L_DIST3	Distance of link for mode 3.
16-18	L-DIST4	Distance of link for mode 4.
19-48	L-NAME	Link Name (30 characters).
49-80	FILL	Filler - may be blank.

CITYPI

Population and income data on the CITYPI file are in a fixed format on 80 character records. Three types of records may be supplied. Currently, no more than 6 years and 14 cities may be specified. National data - national data is must precede the city data. The year must be specified and the city code must be left blank. City identification - These are optional. These records identify the city numbers used. If not provided, the city numbers are determined from the city data records.

The year must be zero on these records.

City data - the city data records provide the city population and income for each forecast year and city. As many records as cities must be supplied. As many sets of city records as forecast years must be supplied.

National Data (Required)

CHARACTER NUMBER	VARIABLE	DESCRIPTION
1-4	PI_YR	Year of supplied data, must be specified.
5-6	PI_CTY	City number, must be zero.
7-12	PI_INC	National income level in dollars, must be specified.
13-21	PI_POP	National population level, must be specified.
22-80	PI_FIL1	May be left blank.

City Identification (Optional)

CHARACTER NUMBER	VARIABLE	DESCRIPTION
1-4	PI_YR	Year of supplied data, must be zero.
5-6	PI_CTY	City number, must be specified.
7-12	PI_INK	City income level, must be zero.
13-21	PI_POP	City population level, must be zero.
22-80	PI_FIL1	May be blank.

City Data (Required)

CHARACTER NUMBER	VARIABLE	DESCRIPTION
1-4	PI_YR	Year of specified data, must be specified.
5-6	PI_CTY	City number, must be specified.

City Data (Required) [continued]

CHARACTER NUMBER	VARIABLE	DESCRIPTION
7-12	PI_INC	City income level, must be specified.
13-21	PI_POP	City population level, must be specified.
22-80	PI_FIL1	May be blank.

CIYINF1, CTYINF2

City pair data on the CTYINF files are in a fixed format. Four 80 character records are required for each city pair. Each of four modes requires one of these records. The highest city pair number should not exceed the value specified in MAX.CTY_PRS. The input record characteristics is given following the description:

CHARACTER NUMBER	VARIABLE	DESCRIPTION
1	IC_MODE	Transportation Mode Code 1 - auto 2 - air 3 - bus 4 or 5 - rail
2-4	IC_CITY_PAIR_CODE	City pair code - this number must not exceed the value specified in MAX.CTY_PRS.
5-7	IC_PERCENT_BUSINESS_TRIPS	This indicates the share of total trips are in purpose 1 (business trips), if BUSSW= 2 otherwise this field is ignored. An implied decimal point is between columns 5 and 6.

CHARACTER NUMBER	VARIABLE	DESCRIPTION
8-10	IC_FIL1	This may be left blank.
11-18	IC_PERSON_TRIPS	The number of person trips using this mode for the specified city pair.
19-21	IC_ACCESS_TIME	The access time in minutes for each trip.
22-24	IC_LINE_HAUL_TIME	The line haul time in minutes for each trip.
25-28	IC_ACCESS_COST	The access cost per passenger for each trip in cents. The auto access cost is for each vehicle.
29-33	IC_FIL2	This may be left blank.
34-36	IC_TOLLS	The total tolls for the city pair for the auto mode only, in cents.
37-40	IC_LINE_HAUL_COST	The line haul cost per person for each trip except for auto which is the line haul cost per vehicle. The cost
41-44	IC_FIL2A	This may be left blank.
45-47	IC_ROUND_TRIP_FACTOR	The round trip factor. This variable is currently unused.
48-51	IC_FIL3	This may be left blank.

CHARACTER NUMBER	VARIABLE	DESCRIPTION
52-53	IC_VEHICLE_TRIPS	The number of one-way vehicle trips per day for each none=auto mode.
54-55	IC_FIL4	This may be left blank.
56-59	IC_DISTANCE	The distance in miles between the two cities in the city pairs for the specified mode.
60-80	IC_NAME	The city pair name is 21 characters or less. This is coded on the mode =1 record only. This may be omitted if the city pair name is coded on the CPCODES record.

PROGRAM OPERATION

After the program parameters are read from SYSIN, the dynamic tables are allocated. If link data is to be accumulated, file LINK is read. The city pair equivalence table is read from CPCODES, followed by the population and income date from CITYPI.

The base year city pair date (CTYINF1) is the input for all city pairs. Modal weights are computed for each city pair and if CALBRATION = 'YES' then the 'A1' coefficient for model is calibrated. The travel demands and revenue is then distributed over the modes (allowing for 2 purposes if indicated by BUSSW=2).

If PRINTSW = 'DETAIL', the detailed city pair report is produced, otherwise, only the link and summary reports are produced.

The forecast city pair data is then input (CITINF2). The modal weights are calculated without calibration and the process repeats as described above.

APPENDIX C

REPORT OF INVENTIONS

Work performed under this contract included the development of innovative techniques for modeling railroad passenger-car allocation under certain assumptions. In addition, the contract required creation of specialized computer programs, not previously available, to apply these techniques.

No inventions were made in performance of this contract.