

**GRAPHICS SOFTWARE
FOR MATHEMATICAL OPTIMIZATION:**

**A Guide to Mathematical Programming
on Project Athena at MIT**

DRAFT

Harilaos N. Psaraftis

**Massachusetts Institute of Technology
Cambridge, Ma 02139**

September 1988

This is a DRAFT version and therefore subject to revision.

Any comments should be addressed to the author.

Address: H.N. Psaraftis

Room 5-211, MIT,

Cambridge, MA. 02139

Tel: (617) 253-7639

E/MAIL: hnpсар@athena.mit.edu

TABLE OF CONTENTS

Foreword

Table of Contents

Chapter 0: Introduction: Getting Started

Chapter 1: Linear Programming Using Simplex

Chapter 2: Nonlinear Programming

Chapter 3: Integer Programming

Chapter 4: The Transportation Problem

Chapter 5: The Shortest Path Problem

Chapter 6: The Minimum Cost Flow Problem

Chapter 7: The Maximum Flow Problem

Chapter 8: The Traveling Salesman Problem

Chapter 9: The Vehicle Routing Problem

Chapter 10: Conclusions and Recommendations

References

FOREWORD

This publication describes the software packages developed in conjunction with the project "Graphics Software for Mathematical Optimization", sponsored by Project Athena at MIT. The purpose of this project has been to develop educational graphics-based software, to be used in mathematical optimization courses at MIT. This development effort started in January 1987, and as of September 1988 has produced nine (9) packages, covering many of the basic mathematical programming problems and methods. Most of these packages have been used in two courses, starting in the fall of 1987.

There are obviously several distinct perspectives from which such a work can be viewed, including those of (a) the student user, (b) the faculty user, (c) the faculty developer, and (d) the software programmer, to name just a few. Although these perspectives are interrelated, this specific report focuses on the perspective of the optimization student using this software in class or in homework assignments. It is my intention to further elaborate on the other perspectives, especially on (b) and (c), in a future report, and as soon as sufficient experience and feedback on the use of this software has been acquired and assessed.

The main motivation that led to this project (other than the fact that I like drawing and hence graphics) was that until recently there was virtually no attempt to exploit the computer for educational purposes within the MIT optimization curriculum. Thus, for most of the approximately 15 optimization-related courses at MIT (spanning the entire spectrum from undergraduate to doctoral-level education, and commanding a total enrollment of over 600 students a year), the educational potential of the computer has been,

and still is, largely untapped. With the possible exception of occasional use of commercial (non-graphics) packages such as LINDO and MINOS, "doing things by hand" has been usually the norm in all of these subjects. This problem has been particularly acute in introductory optimization courses (graduate and undergraduate) which put more emphasis on the student understanding the basics (and hence being inevitably involved in number-crunching) and less emphasis on mastering rigorous analytical proofs, theorems, and derivations (typical in more advanced courses).

With the massive influx of computers on campus since Project Athena started a few years ago, an obvious opportunity became apparent. The basic question that arose was: "Can the vast resources of the Athena system, and the practically unlimited talent and ingenuity of MIT students who could work on a software development project, be put together for the benefit of the optimization curriculum at MIT?"

(I note here parenthetically that the deliberate choice to consider the UNIX-based workstation environment of Athena rather than an alternative configuration - such as a PC or Macintosh-based system - was dictated by the sheer dominance of the Athena system in terms of resources - hardware, software, and human - and infrastructure).

Today, and after almost two years of fun in what I regard as perhaps the most exciting project I have worked on during my 14 years at MIT, and in spite of some occasional feelings of despair by those student users who rightly felt as guineapigs, I can say without hesitation that the answer to the previous question is "yes!".

Notwithstanding several natural caveats ((a) yes, there might be alternative, or better ways to use the computer in class, (b) yes, it may still

be OK to do a few things by hand, and (c) yes, there are still some bugs waiting to be discovered!), I believe that this has been a very fruitful exercise, for the following reasons:

- 1) At a minimum, we now have the beginning of a library of basic optimization software that is operational and easily available to everybody. So far, such software either did not exist, or was widely scattered or user-unfriendly. Of course, this toolkit is specifically designed for educational purposes and hence may not be suitable for "production" runs of large-scale problems. I hope that a similar "research-oriented" toolkit will be developed and installed at some point in the near future.
- 2) The sheer visual impact of the graphical representation of an optimization problem on a powerful, high-resolution computer workstation can greatly boost one's ability (and appetite!) to understand the structure of the problem. The same is true regarding key algorithmic steps. This impact becomes even more pronounced for those applications in which the screen display is "animated" in real-time.
- 3) Using the computer as an "advanced calculator" can free up valuable time which the student can use for other, more rewarding endeavors, such as playing "what if" with various algorithmic techniques, seeing what happens if an erroneous step is taken, observing the result if a required assumption is relaxed, finding the optimum interactively, or generally putting his/her mind to work for productive purposes.
- 4) Getting the student's feet wet with such software (and hardware) can usually stimulate the development of additional software for such purposes as theses, research projects, or further curriculum development.
- 5) Synergies with research: Although there is a fundamental difference between

educational and research-oriented software (in terms of relative emphasis on pedagogical value versus computational efficiency, for instance), there can also be significant synergies between the two, especially in such areas as logistics and transportation in which the role of graphics is increasingly important.

One of our basic premises when this project started was the idea that the software to be developed would be based on existing non-graphics optimization codes, suitably linked to a graphics interface. However, and except for the code implementing Newton's method which was generously donated by Dimitri Bertsekas, all other optimization codes were developed from scratch. In spite of this additional effort, we managed to surpass the goals we had originally set for this project, at least in terms of variety of optimization problems that we worked on (see below).

One area we did not get into was the development of computer-assisted tutorials. By this I mean the student taking a tutorial in a specific optimization area, with the computer being the expert tutor. Although something along these lines could be developed (and has been developed in other Athena projects), in this project the expert tutor is assumed to be the human instructor, who uses the computer as one of his/her instruction tools.

We emphasize here that all of our packages allow the user to input his/her own problem and then solve it. An alternative mode (which was not pursued and which would probably be more suited for computer-driven tutorials) would be for the user to be allowed to work only on standardized problems, "hard-wired" into the software. Of course, our mode of operation does not preclude the human instructor to specify which would be the problems of study, or even have them available in a data file.

So far, the mode of use of this software has been two-fold. Either in

lab sessions (usually taking place in the "electronic classroom" in room E37-312), or in homework assignments. In lab sessions, students get exposed to the capabilities of the software by working on standard problems. In homework, students use the software as a companion toolkit to solve the problems that are assigned.

The nine graphics packages that were developed and are now operational on the Athena system are the following:

- . lpprog, for linear programming using Simplex
- . nonlinear, for unconstrained nonlinear programming
- . intprog, for integer programming
- . transp, for the transportation problem
- . spm, for the shortest path problem
- . mcost, for the minimum cost flow problem
- . maxfl, for the maximum flow problem
- . travel, for the traveling salesman problem
- . vrp, for the vehicle routing problem.

In addition, the standard (non-graphics) linear programming package LINDO is also available on Athena.

By design, and as of September 1987, these packages have been primarily intended for two MIT courses:

- (a) the undergraduate course "Optimal Design of Engineering Systems" (1.02/6.490/13.671), offered as an operations research schoolwide elective in the School of Engineering, and
- (b) the graduate course "Logistical and Transportation Planning Methods" (1.203J/6.281J/11.526J/13.665J/15.078J/16.76J), offered as an elective in several departments and in the programs of the Operations Research Center and of

the Center for Transportation Studies.

So far, all packages except travel and vrp are intended for (a), and packages spm, travel, and vrp are intended for (b).

In addition to students of the above two courses, this software is available to and easily accessible by any member of the MIT mathematical programming community who wishes to use it. Instructions on getting access to this software are in Chapter 0.

All of these packages are written in Fortran-77, and use the package BLOX for the graphics interface. BLOX is one of the main graphics packages supported by Athena. In terms of hardware, these programs have been so far implemented on the Vaxstation II or 2000, although running them on the IBM RT workstation should be straightforward.

Acknowledgements

Funds for this project have been provided by Project Athena and the Dean of Engineering, with matching support being provided by the Department of Ocean Engineering. The administrative home of this project has been the MIT Operations Research Center.

This project would not be possible without the help of several individuals, whom I feel obliged to thank. First and foremost, my thanks go to David Hwang of Project Athena, who served the dual role of software developer and technical consultant to the rest of the team. Equal gratitude goes to the developer team, graduate students Lambros Anagnostopoulos, Christopher Hrut, Jihong Ou, and Peter Vranas. Steve Ellis and Jackie Stewart of Athena provided excellent technical and administrative support. My OR Center colleagues Dimitri Bertsekas, Rob Freund, Tom Magnanti, Jim Orlin, and Yossi Sheffi provided ideas

and advice. And last, but not least, my appreciation goes to all the graduates and undergraduates who used this software, for their patience, feedback, and many helpful suggestions.

CHAPTER 0

INTRODUCTION: GETTING STARTED

Using this software is very easy. Although it definitely requires familiarity with the particular optimization problem and/or techniques to be worked on, it requires very little knowledge of the intricacies of the Athena system, or of Unix, the operating system. Very little knowledge of X, the workstation windowing system is also required.

Of course, some familiarity with Athena would definitely help, particularly in unforeseen situations (e.g. if you are stuck and you want to kill the program). The following Athena publications are recommended for those who want to become more familiar with the Athena system:

- Essential Unix
- Essential Workstation

These and other Athena publications are available in public clusters (see Figure 0.1) or in Athena headquarters (3rd floor of building E40).

To run this software, you first need an Athena login id. Most undergraduates should already have one. If you do not have an Athena id, please contact Carla Fermann at x3-1325, or the course instructor.

Go to any Vaxstation cluster, and, after you login, issue the following commands, one by one (wait for prompt before proceeding):

```
attach opt2
```

```
attach beta  
source /mit/beta/vsblox/bgbsetup  
cd /mit/opt2
```

Now you should be ready to run the software. Before you proceed, we recommend that you set up a second window at the right hand side of the screen (using command `xterm&`). A second window has many uses, including screen dumps (hardcopies), killing the program if you are stuck, etc. Once you do that, move the cursor to the primary window, and issue the following command:

```
optimize
```

After a brief wait, you are next presented with a screen, displaying a menu of available packages, and some introductory messages (see Figure 0.2). Read the introduction and select the package you wish by clicking the mouse cursor into the appropriate menu box. Details about the graphics packages are presented in the remaining chapters. To terminate, select "exit".

Note: All of these packages have specific code names (lpprog, intprog, spm, etc.). However, the user is not necessary to know or remember these names, since the packages are automatically called once the appropriate menu items are selected.

Linear Programming Using LINDO

In addition to the various graphics packages developed in this project, the widely known commercial LP package LINDO is also available on the Athena system. LINDO (for Linear, Interactive and Discrete Optimizer) is a system for

| Athena Mathematical Optimization Project | | |
|--|----------------------------|---------------------------|
| Linear Programming (Simplex) | Nonlinear Programming | Integer Programming |
| Transportation Problem | Shortest Path Problem | Minimum Cost Flow Problem |
| Maximum Flow Problem | Traveling Salesman Problem | Vehicle Routing Problem |
| LINDO (non-graphics) | Comments (mail) | Help |
| Introduction | Quit | Hardcopy |

INTRODUCTION OF MATHEMATICAL OPTIMIZATION PROJECT

Welcome to the Athena Mathematical Optimization Project! The purpose of this project has been to develop graphics-based software to help the teaching of optimization courses at M.I.T. For further information and any questions please contact Professor H.N. Psaraftis, Room 3-211. Tel (617) 253-7639, or e-mail hnpsar@athena.mit.edu (menu item: Comments)

At this point, there are ten packages available for your use:

- Linear Programming using Simplex
- Nonlinear Programming
- Integer Programming
- Transportation Programming
- Shortest Path Problem
- Minimum Cost Flow Problem
- Maximum Flow Problem
- Traveling Salesman Problem
- Vehicle Routing Problem
- LINDO (non-graphics) program

Detailed instructions on how to use these packages are described in the following technical report: [GRAPHICS SOFTWARE FOR MATHEMATICAL OPTIMIZATION] --- A guide to Mathematical Programming on Project Athena at M.I.T. by H.N. Psaraftis, M.I.T., September 1988. Although all packages are pretty much self-explanatory and user-friendly, we suggest obtaining this report as a companion manual.

To proceed, select the package you want from menu, and wait for the program to be loaded.

Figure 0.2: Initial menu

solving linear, integer, and quadratic programming problems, developed by Linus Schrage. To run LINDO, select appropriate item from menu. LINDO is self-documenting (type HELP for a summary of available options). A user's manual is also on sale in educational bookstores.

Making Hardcopies

Making hardcopies of screen displays can be a tricky proposition on the Athena system. All packages have a "hardcopy" option available, but as of September 1988, it is not yet clear whether this option can be implemented by the system.

However, there is also an indirect way to make hardcopies. For this to work, you have to be in a cluster that is equipped with an LN03plus laser printer. A simple LN03 printer may not have adequate memory for the dump, and a LN03 postscript printer will not do the trick.

To make the screen dump, issue the following command in the second window:

```
xwd | xpr | lpr
```

When the cursor becomes a cross, click the mouse with the cross inside the graphics display you want to copy. The hardcopy is available from the printer in a few moments.

CHAPTER 1

LINEAR PROGRAMMING USING SIMPLEX

Developer: Jihong Ou

Introduction

Program lpprog allows the user to visualize on the screen a linear programming problem with two (main) variables and at most 9 constraints. Problems with more variables are not easily amenable to graphical representation. Problems with more constraints produce Simplex tableaux that cannot be neatly displayed on the screen. Such (larger scale) problems can be solved by other packages, such as LINDO (see Chapter 0 on how to run LINDO).

Summary of User Options

- 1) Enter an LP, either from the terminal, or read from a file.
- 2) Edit the problem before solving.
- 3) Show LP feasible region and Simplex tableaux.
- 4) See sequence of Simplex pivots (executed by the program or selected by the user).
- 5) Post-optimize after altering the problem (change an objective function coefficient, change a right-hand-side, add or delete a constraint).
- 6) Perform a dual pivot in case of primal infeasibility.
- 7) Store the problem in a file.

Instructions

Your first screen after evoking lpprog (see Chapter 0 on how to do this) displays an introductory message and a menu with two main options. To enter an LP problem from the terminal, select TYPE IN A NEW LP PROBLEM. To read an LP problem from a file, select RECALL A STORED PROBLEM. If you do the latter, specify the name of the file to be read, when prompted.

Entering an LP problem from the terminal is straightforward. Just type the objective function and constraints, one by one, when prompted. Valid objective functions are: $\max 4x + 2y$, $\min 2.5s + 7r$, etc. Valid constraints are $x + y < 10$, $2x + 5y > 2$, etc. Invalid forms are: $\max 4*x + 2*y$ (* is not recognized as a multiplication sign), $x + y < = 10$ (typing < or > implies \leq or \geq respectively), $x + 2y = 5$ (equality constraints are not allowed). Note that non-negativity constraints are implicitly assumed, so there is no need to enter them.

Figure 1.1 shows the user having typed $y < 8$ when prompted by: CONSTRAINT 3). In this example, the user has already entered an objective function and two constraints, shown at the top of the screen.

A return after $y < 8$ is typed prompts the user with CONSTRAINT 4). If no more constraints exist, just hit return. At this point, the program asks you if you want to store this problem in a file. If yes, give the name of that file.

The next screen is similar to the one shown in Figure 1.2, and displays the LP feasible region, the LP formulation, the initial Simplex tableau, and a menu.

The picture of the LP feasible region includes an arrow to indicate the

ATHENA MATHEMATICAL OPTIMIZATION PROJECT

The Objective
 Max xy
 Subject to
 (1) $xy \geq 10$
 (2) $x \geq 7$

CONSTRAINT 3) $y < 8$

Menu Selection
 ** MENU SELECTION **

TYPE IN A NEW LP PROBLEM

RECALL A STORED PROBLEM

HELP

HARDCOPY

TO INTRODUCTION

TERMINATE PROGRAM

Figure 1.1: Entering an LP from the terminal.

ATHENA MATHEMATICAL OPTIMIZATION PROJECT

The Objective
 Max xy
 Subject to
 (1) $xy \geq 10$
 (2) $x \geq 7$
 (3) $y \leq 8$

CONSTRAINT 3) $y < 8$

Menu Selection
 ** MENU SELECTION **

EDIT THE PROBLEM

S-CR SIMPLEX PLOTS

BIG FIGURE

BIG TABLEAU

BACK TO START

HELP

HARDCOPY

TO INTRODUCTION

TERMINATE PROGRAM

| BASIS | RHS | x | y | S1 | S2 | S3 |
|-------|-------|------|------|------|------|------|
| 1- S1 | 10.00 | 1.00 | 1.00 | 1.00 | 0. | 0. |
| 2- S2 | 7.00 | 1.00 | 0. | 0. | 1.00 | 0. |
| 3- S3 | 8.00 | 0. | 1.00 | 0. | 0. | 1.00 |
| -Z | 0. | 1.00 | 2.00 | 0. | 0. | 0. |

Figure 1.2: Basic screen layout

current solution, and an isoquant of the objective function at that point. The feasible region is shaded (if non-empty). (Note: to make the display less boring, the shade pattern depends on the number of constraints - compare with Fig. 1.10).

The Simplex tableau shows, in addition to the original two variables, the slacks corresponding to the problem constraints (in the example of Fig. 1.2, these are S1, S2, and S3).

The major menu options are explained as follows:

EDIT THE PROBLEM: Gives you the opportunity to correct typos or make other changes before you further proceed. Selecting this option takes you to another menu, on possible changes, such as CHANGE THE OBJECTIVE, DELETE ONE CONSTRAINT, CHANGE ONE CONSTRAINT, and ADD NEW CONSTRAINT. Make changes when prompted. When done, select DONE, and you get back to the previous menu.

BIG FIGURE magnifies the LP feasible region, and BIG TABLEAU magnifies the Simplex tableau (see Figs. 1.3 and 1.4 respectively - these correspond to the infeasible basis of Fig. 1.10, obtained at a later iteration - see below).

SHOW SIMPLEX PIVOTS: This menu item allows you to proceed from one basic solution to another, by performing a Simplex pivot. As Fig. 1.5 shows, there are three options once this menu item is selected: DO IT YOURSELF, LET ME PROCEED ONE STEP, and DO NOTHING → RETURN.

THE DO IT YOURSELF option allows you to interactively choose your own pivots as you wish. You are even allowed to choose the wrong pivot, and see what happens. This option also asks you if the current basis is feasible, or optimal (Fig. 1.5), and verifies if your answer is correct or incorrect. You choose pivots by first entering the pivot column (variable to enter the basis). Once this is done, the corresponding ratios are displayed so you can perform the

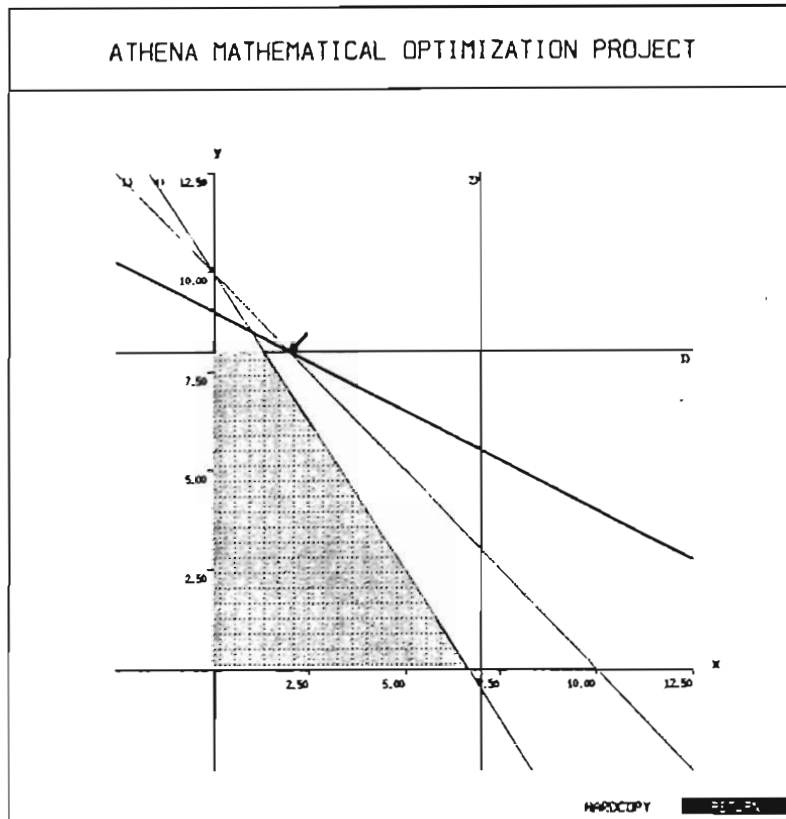


Figure 1.3: BIG FIGURE: LP region magnification

ATHENA MATHEMATICAL OPTIMIZATION PROJECT

| BASIS | RHS | x | y | s1 | s2 | s3 | s4 | |
|-------|--------|------|------|-------|------|-------|------|--|
| 1. y | 8.00 | 0. | 1.00 | 0. | 0. | 1.00 | 0. | |
| 2. x | 2.00 | 1.00 | 0. | 1.00 | 0. | -1.00 | 0. | |
| 3. s2 | 5.00 | 0. | 0. | -1.00 | 1.00 | 1.00 | 0. | |
| 4. s4 | -2.00 | 0. | 0. | -3.00 | 0. | 1.00 | 1.00 | |
| -Z | -18.00 | 0. | 0. | -1.00 | 0. | -1.00 | 0. | |

HARD COPY

Figure 1.4: BIG TABLEAU: Simplex tableau magnification

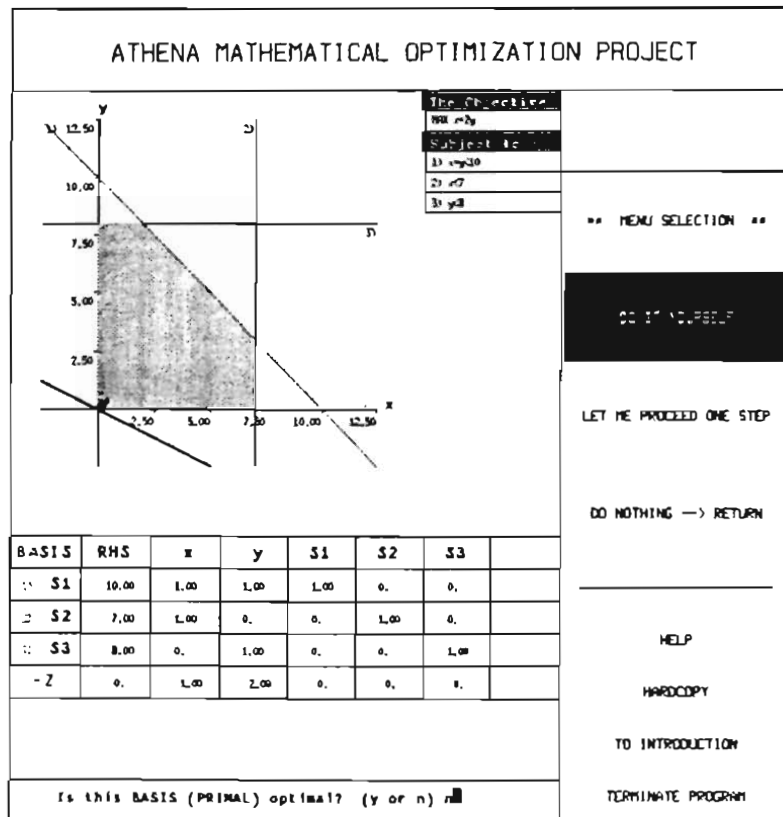


Figure 1.5: DO IT YOURSELF Option: User chooses own pivots

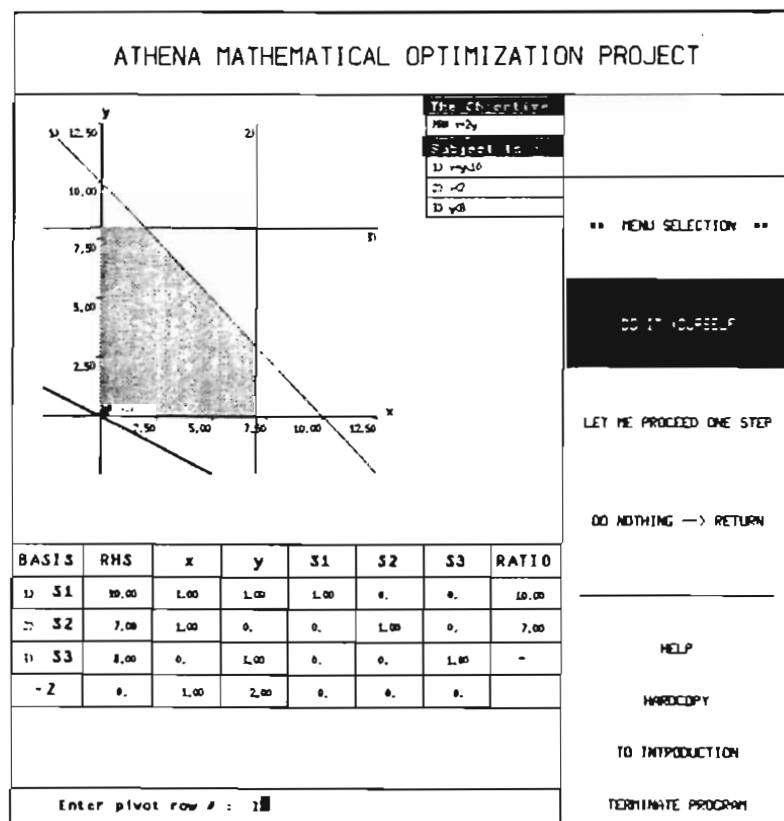


Figure 1.6: User chooses pivot row after column (x) has been chosen (in this example, choosing row 1 is a mistake)

(primal) ratio test. In the example of Fig. 1.6, the user has already chosen column 1 (variable x) as pivot column, and the ratios 10.00 and 7.00 are displayed. (Note: Ratios are displayed even if negative. If all ratios are zero or negative, the problem is unbounded).

You are next asked to enter the pivot row (variable to drop from the basis). In Fig. 1.6, the user mistakenly chooses row 1 (variable S1) as the pivot row. The result is displayed in Fig. 1.7, which shows that the next basic solution is infeasible. You have a chance to correct your mistake and choose a different pivot. (see bottom of Fig. 1.7).

The LET ME PROCEED ONE STEP option performs one valid pivot for you automatically. The pivot element is highlighted (see Fig. 1.8).

The user can switch between the DO IT YOURSELF and the LET ME PROCEED ONE STEP options between successive pivots. Even if you are lost at an infeasible point because of poor pivot selection, you can always use the latter option to get back on track. Moving from an infeasible basis to a feasible one (if one exists) uses the dual Simplex method (see also below).

Solving the problem to optimality or choosing the DO NOTHING → RETURN option takes you to a menu similar to the previous one, except EDIT THE PROBLEM is now replaced by ALTER THE PROBLEM. Selecting the latter option allows you to post-optimize without solving the problem from scratch. Fig. 1.9 is an example. Here the user adds a constraint to the problem. The result is the (infeasible) solution shown in Fig. 1.10.

Moving to restore feasibility involves performing one or more dual Simplex pivots. Again, this can be done by either the DO IT YOURSELF, or by the LET ME PROCEED ONE STEP options. Fig. 1.11 shows one such dual pivot using the second option. All dual ratios are displayed, and the pivot element is

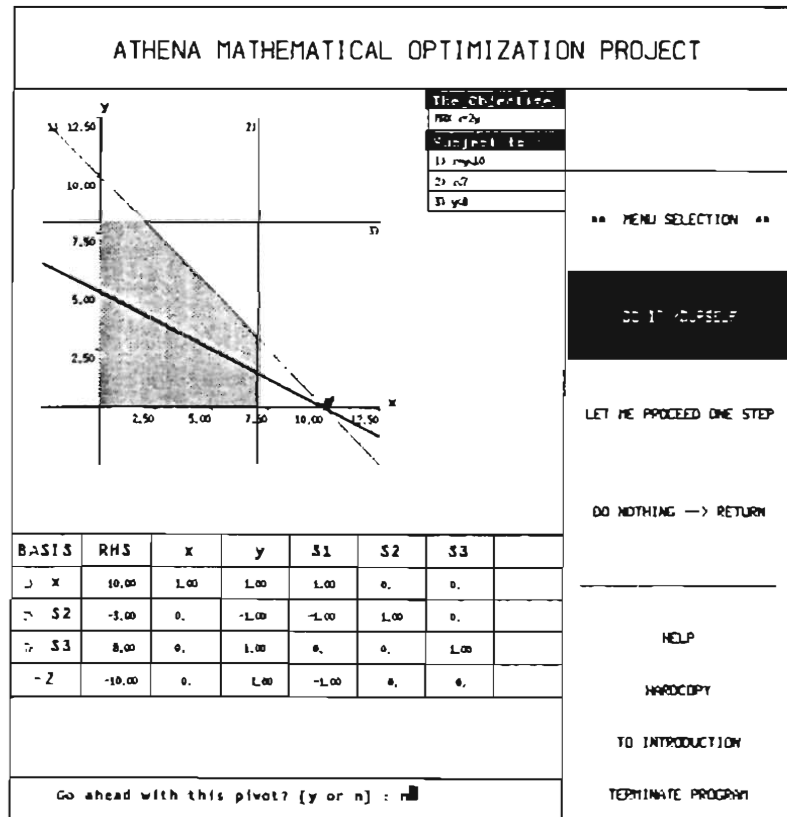


Figure 1.7: User observes what happens if wrong pivot is chosen, then has a chance to correct mistake.

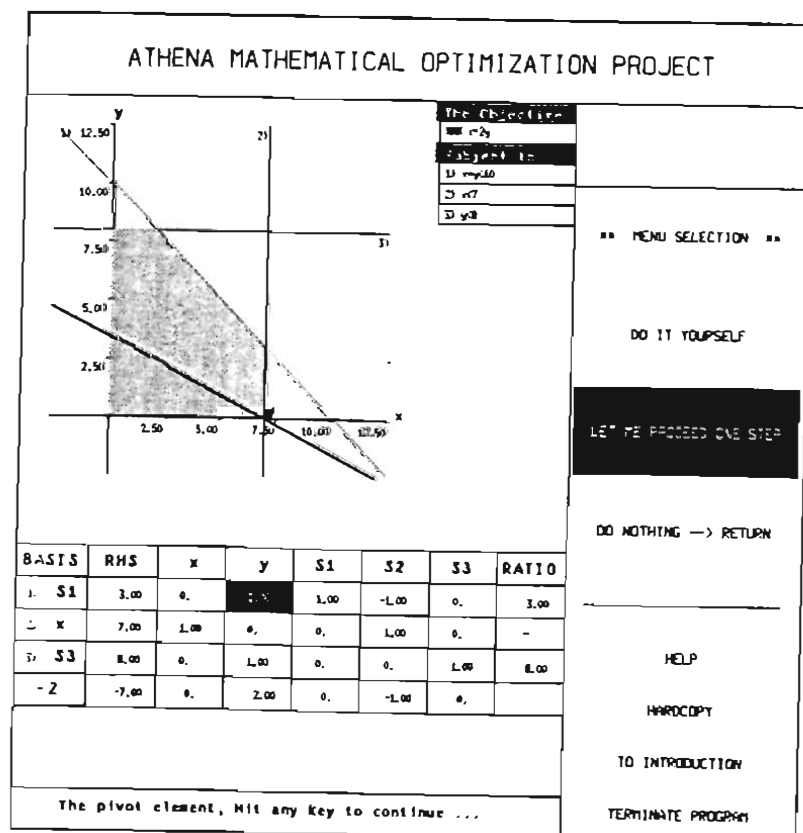


Figure 1.8: LET ME PROCEED ONE STEP option: Pivot element is highlighted

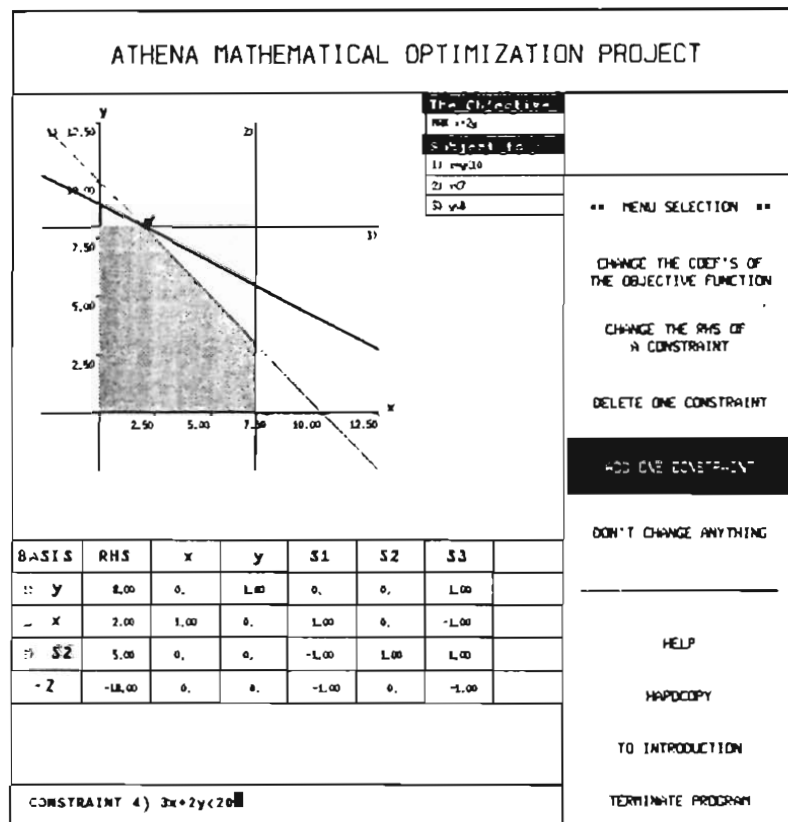


Figure 1.9: Menu of options for altering (post-optimizing) the problem. Here the user adds one constraint.

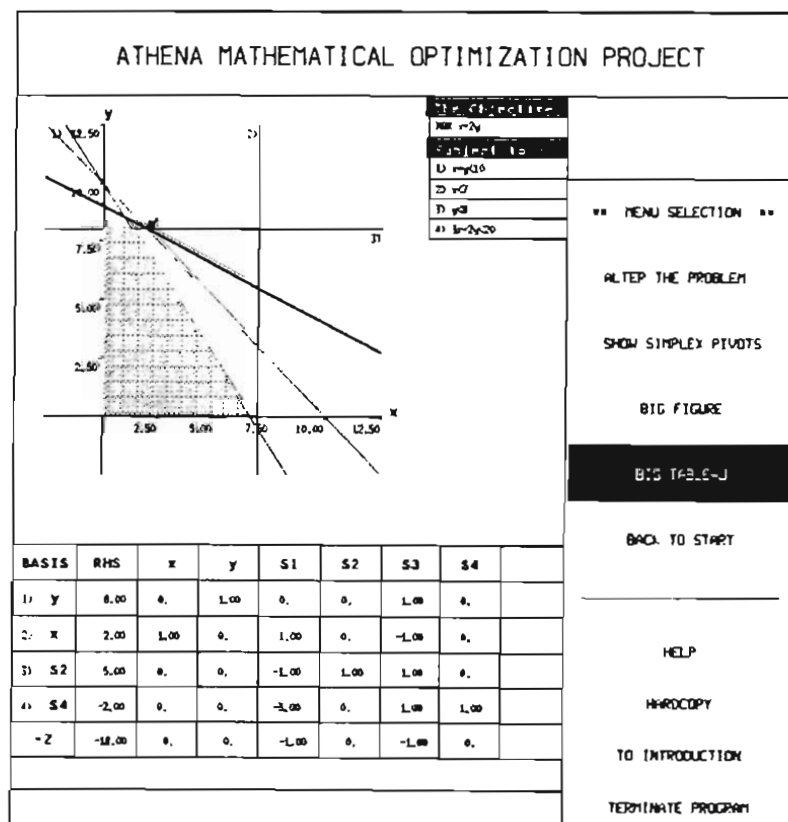


Figure 1.10: Infeasible Basis

highlighted. Fig. 1.12 shows the new optimal solution.

Dual pivoting is also used (instead of Phase I) if the original problem cannot be readily put into canonical form, and the initial basis is infeasible.

Other post-optimization options are: CHANGE THE COEF'S OF THE OBJECTIVE FUNCTION, CHANGE THE RHS OF A CONSTRAINT, DELETE ONE CONSTRAINT, and DON'T CHANGE ANYTHING. See Figs. 1.12 and 1.13 for an example on what happens if the coefficient of x in the objective function is changed to -1 .

Possible Future Work

The software currently does not handle degeneracy. This means that the LET ME PROCEED ONE STEP option does not guarantee anti-cycling. It's up to the user to get out of a degenerate situation, by an appropriate selection of pivots. It should not be that difficult to add an anticycling module.

It should also be easy to add a Phase-I routine, as a substitute for dual Simplex if the original problem is not in canonical form.

Further streamlining the user interface is also a worthwhile goal. Although the interface is currently very user-friendly and flexible, increased use of the mouse (e.g. for picking constraints, pivots, etc.) might be desirable.

As of 8/88, the HELP and HARDCOPY menu options are not yet enabled on this package. Hard copies can be made using the procedure outlined in Chapter 0.

Reference

Any decent book on linear programming, e.g. Bradley, Hax, Magnanti (1977), Chapters 2 to 4.

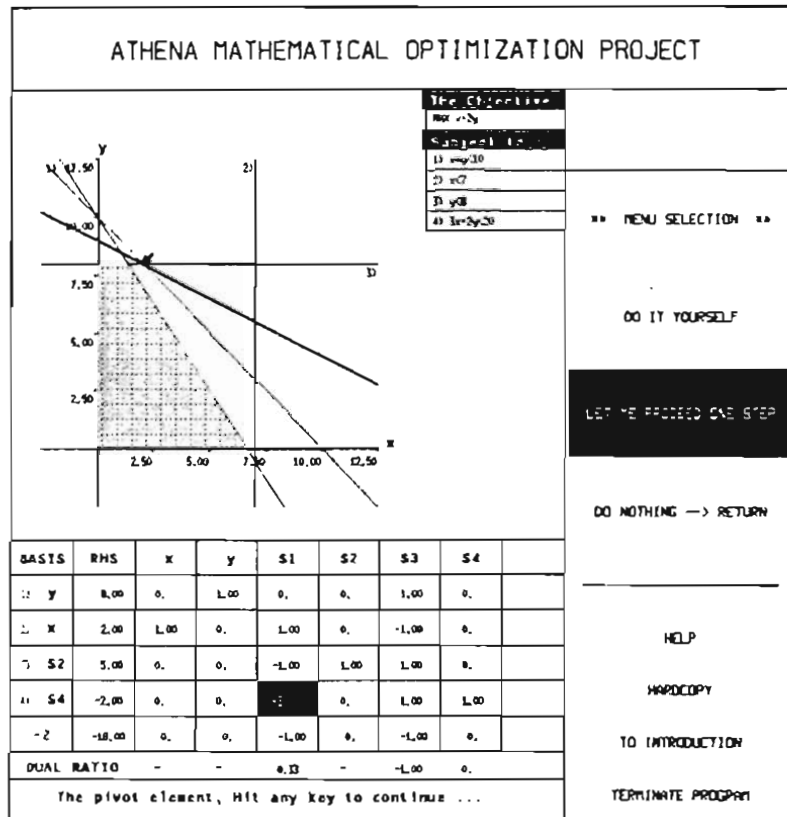


Figure 1.11: Performing a dual pivot

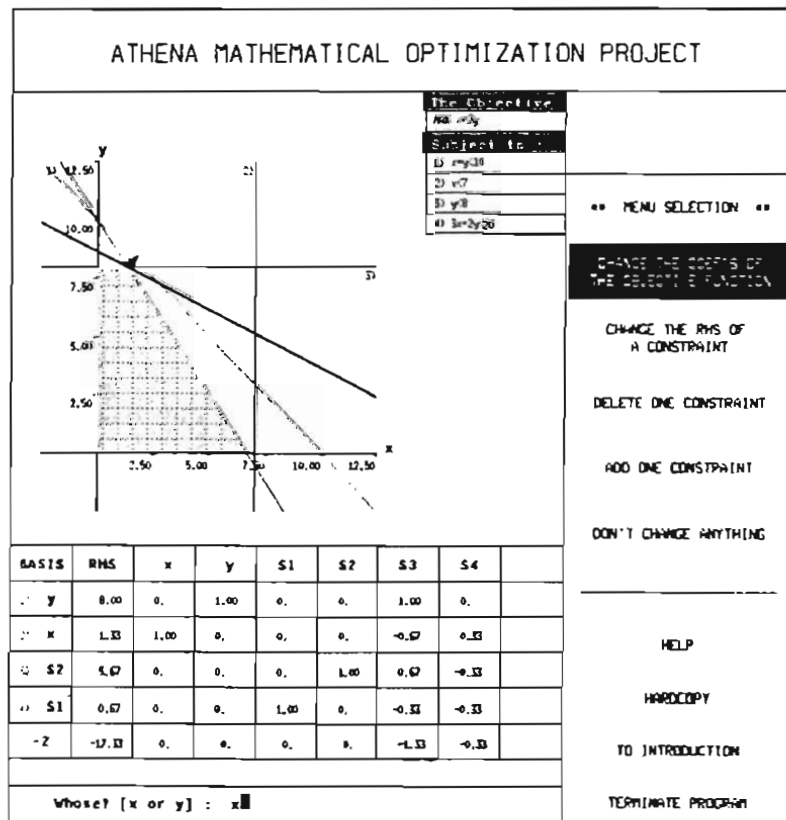


Figure 1.12: Changing the coefficients of the objective function, New coefficient of x is -1.

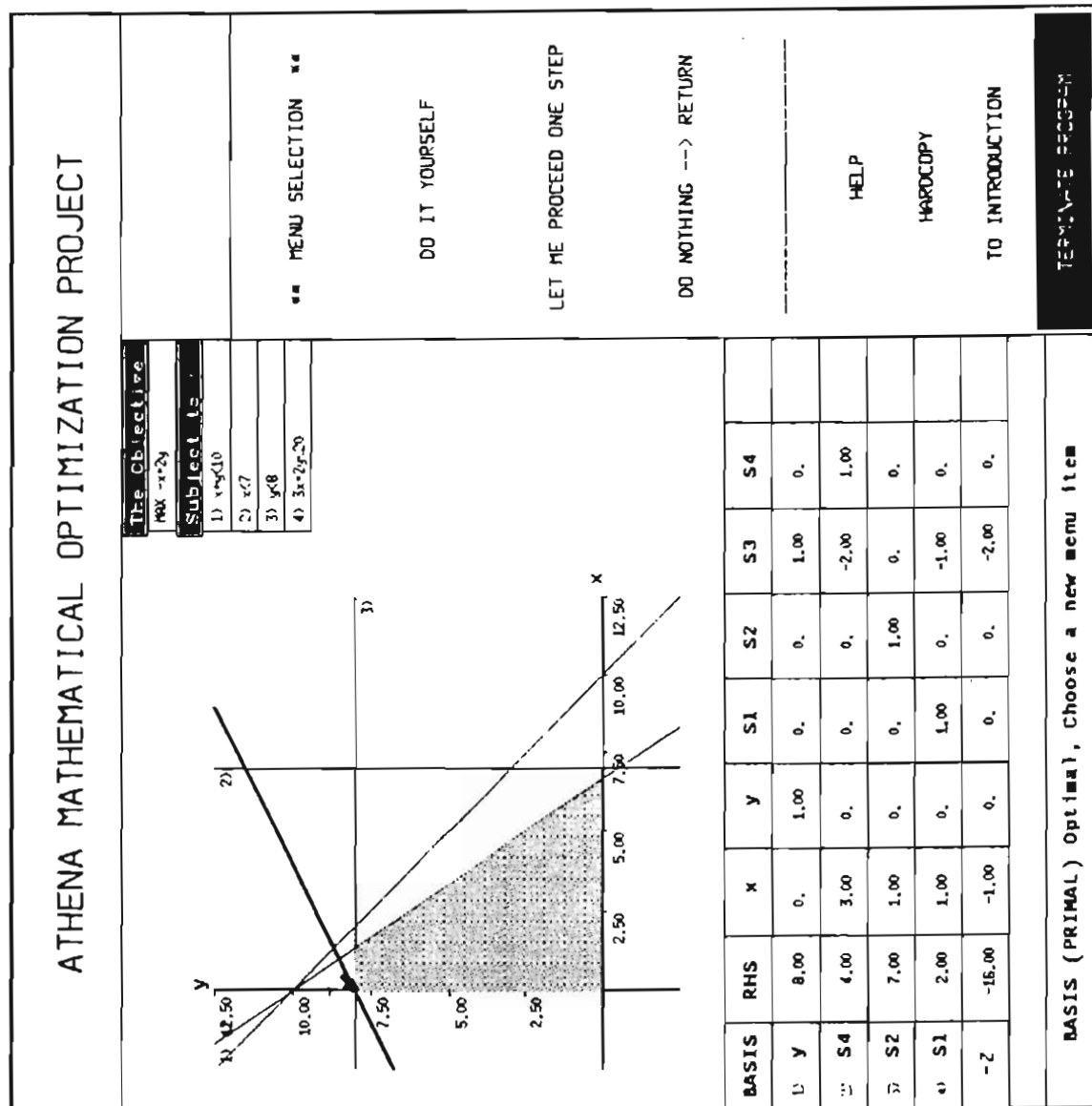


Figure 1.13: New Optimal Solution

CHAPTER 2
NONLINEAR PROGRAMMING

Developer: David Hwang

Introduction

The purpose of program nonlinear is to interactively solve unconstrained nonlinear optimization problems. The basic idea is to help the user visualize on the screen the form of a selected nonlinear function, and then optimize the function using a manual or an automated procedure.

Use of the program is (currently) limited to functions $f(x,y)$ of two decision variables, x and y . Problems of higher dimension are not easily amenable to graphical representation. The program allows the user to look at the contours of $f(x,y) = \text{constant}$, and examine also other information, such as first and second order derivatives, gradient, Hessian, etc. Newton's method can be used to identify a local optimum.

Summary of User Options

- 1) Select from a menu of pre-specified functional forms.
- 2) Select parameters for the function.
- 3) Draw contours of the function.
- 4) Examine information on the function such as: function values, first and second derivatives, gradient, and Hessian.
- 5) Find a local optimum manually.
- 6) Find a local optimum using Newton's method.

Instructions

After this program is called (see Chapter 0) you are presented with the first screen. Read the introduction and select CONTINUE from the menu. This will take you to the screen shown in Fig. 2.1, displaying a list of several (currently 6) families of functions. In all of these functions, x and y are the decision variables, and a, b, c, d, e , and f are parameters to be specified by the user. To select a function, choose FUNCTION SELECTION from menu, and when prompted, type function number (e.g., 3 for quadratic 2-D).

You will next be prompted to enter function parameters (if any), one by one. Hit return each time you enter a parameter.

Fig. 2.2 shows what happens if you select function 5 (Rosenbrock's 2-D banana function) with parameter $a = 1$. You are presented with a screen which displays the functional form at the top, a working screen in the middle (initially blank), an "initialization" display at the bottom left corner, a "bulletin board" at the center bottom, and a menu on the right.

The default window for all problems is such that you can only see what happens between $x_{\min} = -10$, $x_{\max} = 10$, $y_{\min} = -10$, and $y_{\max} = 10$. To change these default values, type y (yes) in the bulletin board. You will then be prompted to enter the new values for x_{\min} , x_{\max} , y_{\min} , y_{\max} . (Note: doing this may require some experimentation. There is no a priori good way to choose "perfect" values, especially for an unknown function).

For this example, boundary values were reset to $x_{\min} = -6$, $x_{\max} = 6$, $y_{\min} = -6$, $y_{\max} = 12$. Doing this will present you with a menu. Choosing DRAW CONTOUR from the menu gets you to a screen like the one shown in Figure 2.3. Initially, the working screen, as well as the boxes showing the values of x , y ,

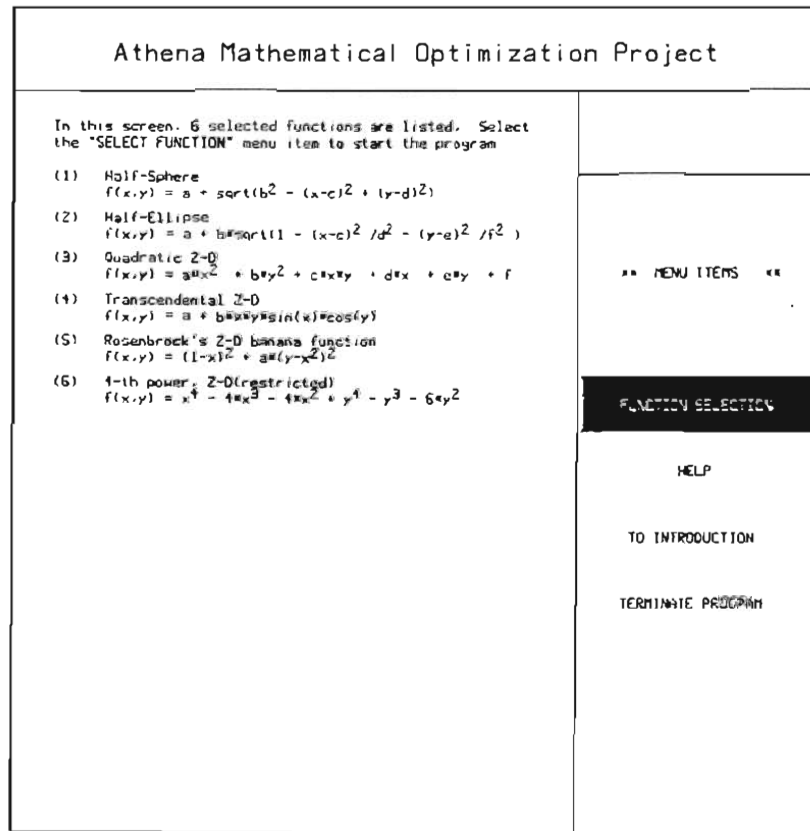


Figure 2.1: Function Selection. Basic families of functions.

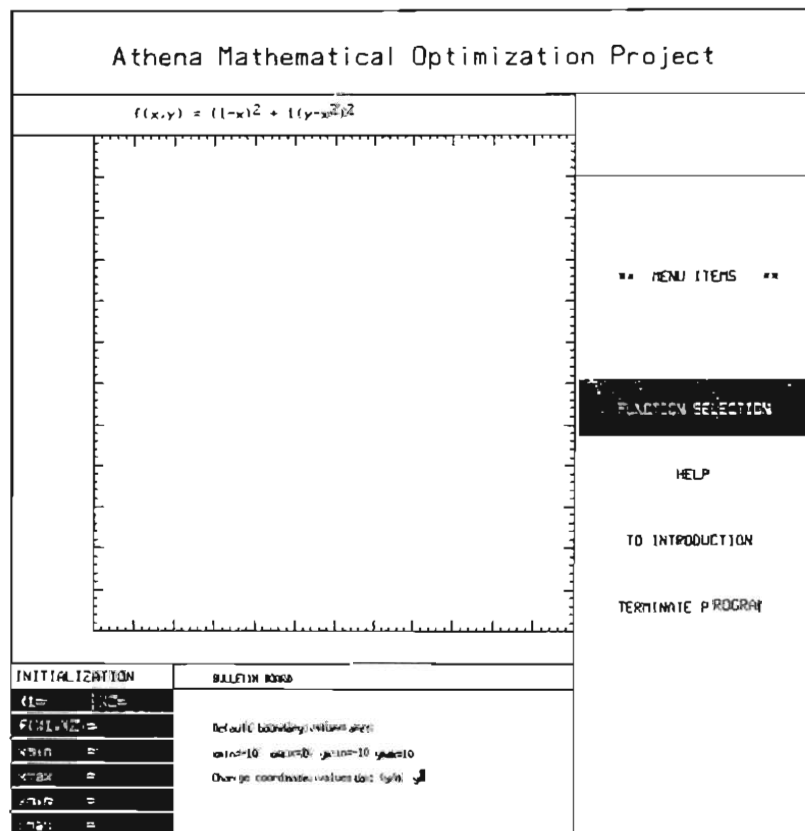


Figure 2.2: Defining the boundary values of x and y.

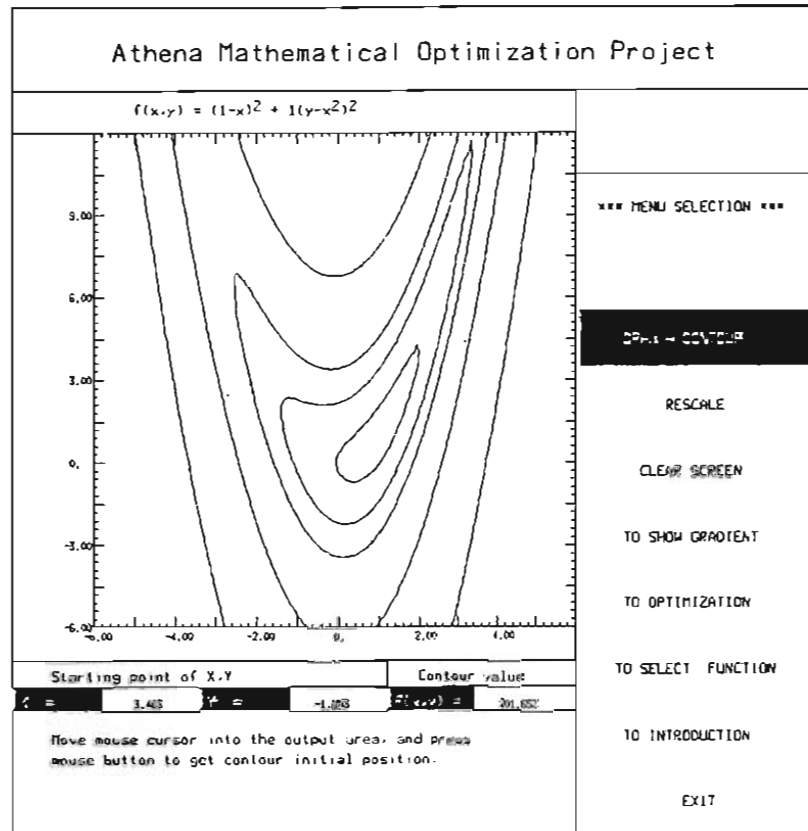


Figure 2.3: Drawing Contours

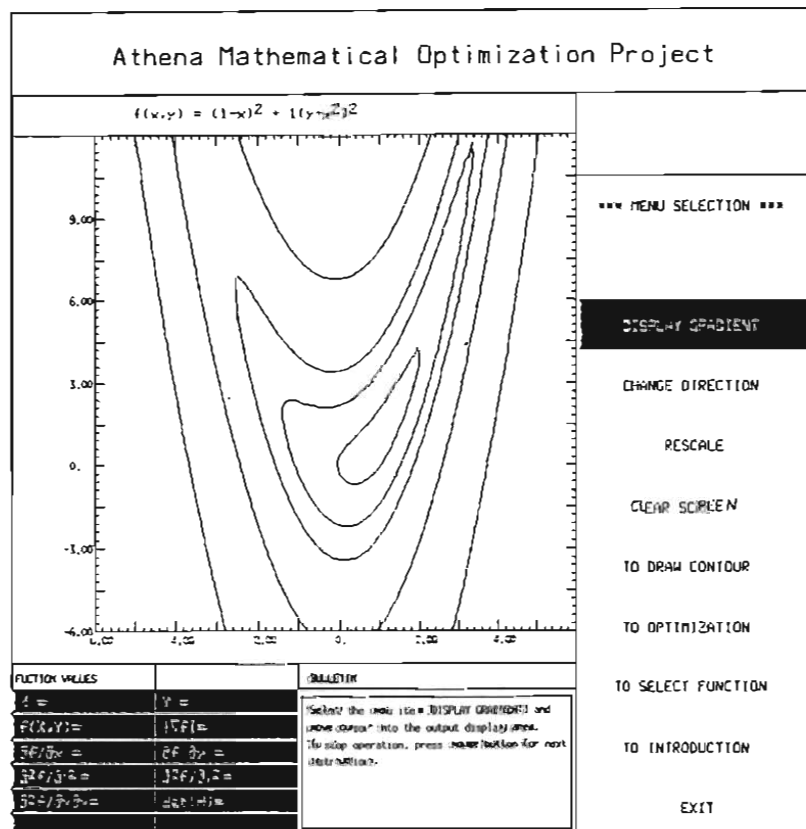


Figure 2.4: Screen layout for DISPLAY GRADIENT option

and $f(x, y)$, will be blank.

To draw a contour, first select DRAW A CONTOUR from menu (Note: this is different from the previous menu DRAW CONTOUR you had selected). Then move the mouse cursor at a (x, y) point on the working screen. Once you click the mouse, a contour passing from this point is drawn. Also shown are the values of x , y , and $f(x, y)$ (which is constant along the contour). You can draw as many contours as you want, by repeating this procedure (click DRAW A CONTOUR each time). In Figure 2.3, five contours are shown.

If you are not satisfied with the way the contours are displayed, you may resize the window by selecting RESCALE from the menu. CLEAR SCREEN gives you a new screen to work on. Suggest clearing the screen only if you rescale.

Selecting TO SHOW GRADIENT brings you into another display, shown in Figure 2.4. The working area of the screen stays the same. However, now there are two new areas (labeled "function values" and "bulletin" at the bottom of the screen), and a new menu.

Selecting DISPLAY GRADIENT and then moving the mouse cursor to any point inside the work area of the screen shows you in real time several things (see Figure 2.5): First, an arrow at the mouse cursor shows you the direction of steepest descent. The magnitude of the arrow is proportional to the norm of the gradient of the function. Second, as the mouse cursor moves, the values of the function, its derivatives, and its Hessian corresponding to the (x, y) coordinates of the mouse cursor are shown and dynamically updated at the "white-on-black" bottom left part of the display. Move the mouse cursor freely around in the work area, and observe what happens. Can you find a point (x, y) for which all first derivatives (and, hence, $|\nabla f|$) vanish? Is it a local minimum, a local maximum, or a saddle point? How can you tell?

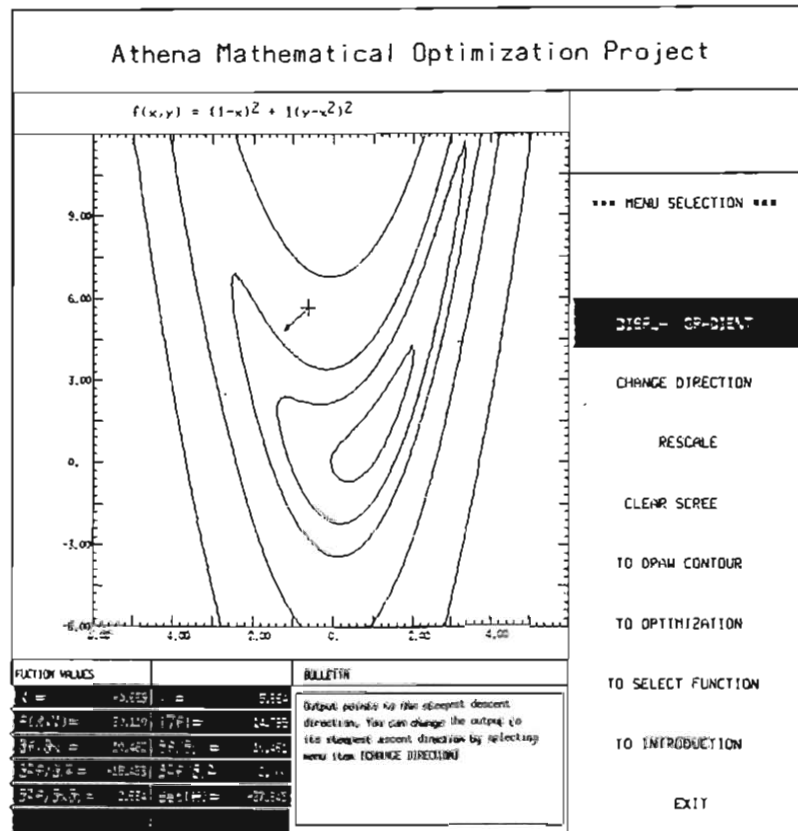


Figure 2.5: Continuous updating of gradient and other function information, as mouse cursor (+) is moved on the (x, y) plane.

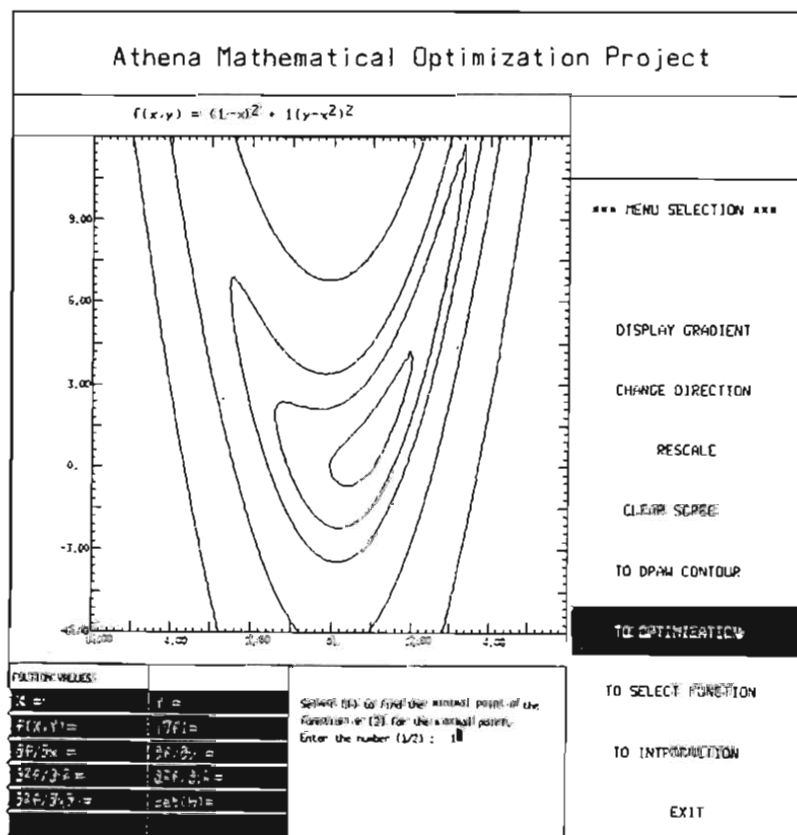


Figure 2.6: Choose whether to maximize or to minimize

You can reverse the direction of the arrow from steepest descent to steepest ascent (or vice versa) by selecting CHANGE DIRECTION from menu. (Note: You have to click the mouse and hit return to get out of the continuous update mode into the menu selection mode).

Selecting TO OPTIMIZATION (see Figure 2.6) prompts you to specify whether you want to maximize or minimize the function. If you specify the former and the function has no maximum, the program will display a message and you will not be allowed to proceed. Similarly with a minimum, or a saddle point. If your request is legitimate, you get a display similar to the one shown in Figure 2.7.

Selecting NEWTON'S METHOD is the main option here. To do this, you have to select an initial starting point. Do this by clicking the mouse at the appropriate (x, y) point in the work screen. This is shown in Figure 2.8. Observe that the display now includes a smaller window at the bottom right, displaying values for x_{\min} , x_{\max} , y_{\min} , y_{\max} , together with the initial point. The purpose of this smaller window is to track down the current values of (x, y) at each step of Newton's algorithm, particularly if these go beyond the initially assumed values of x_{\min} , x_{\max} , y_{\min} , and y_{\max} , that is, outside the working screen showing the contours. If this is the case, the small window boundaries are updated.

Figure 2.9 shows the second iteration of Newton's method, and Figure 2.10 the last iteration, with the local optimum (in this case, the minimum) shown with an * (the asterisk blinks when the optimum is found). Notice in Figure 2.10 the path of (x,y) values at each iteration.

Figure 2.11 shows contours and gradient for function 4, $f(x, y) = xysinx\cos y$. Ordinarily, the user would have little or no idea how a function

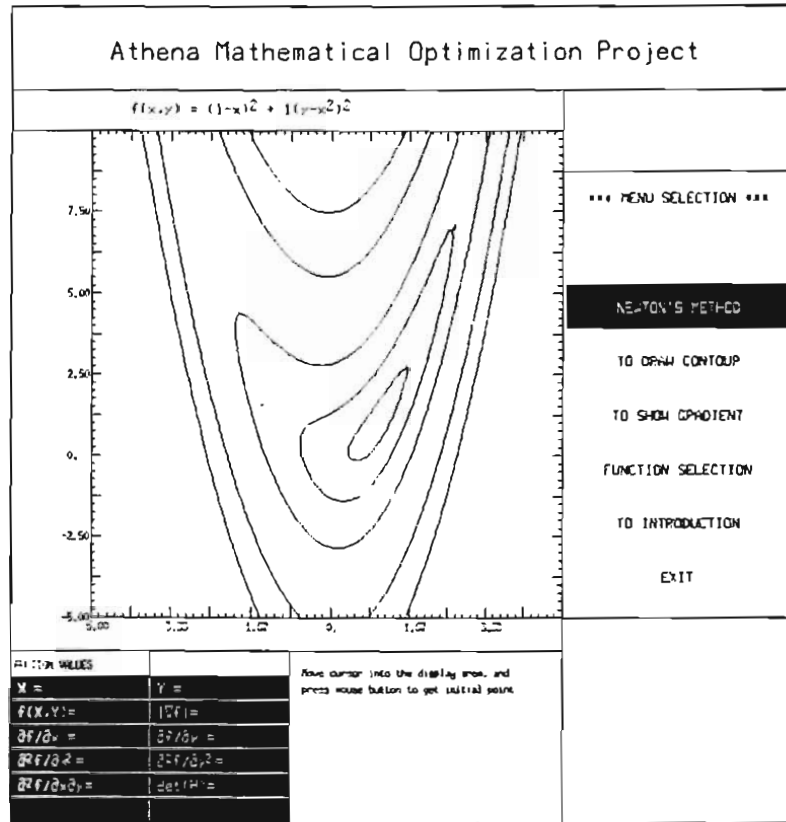


Figure 2.7: Screen layout for Newton's method.

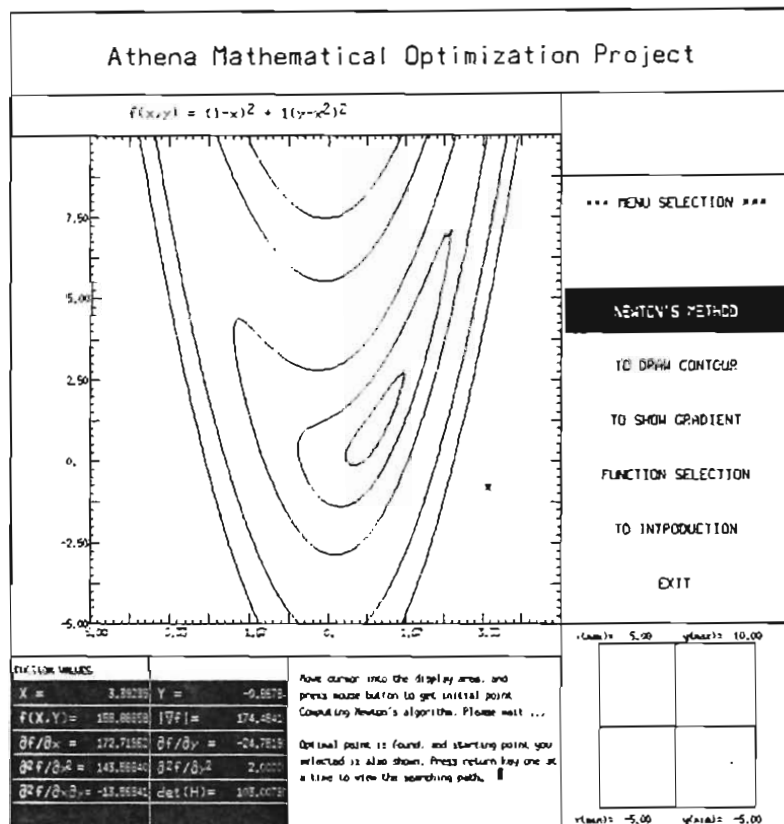


Figure 2.8: Initial point in Newton's method

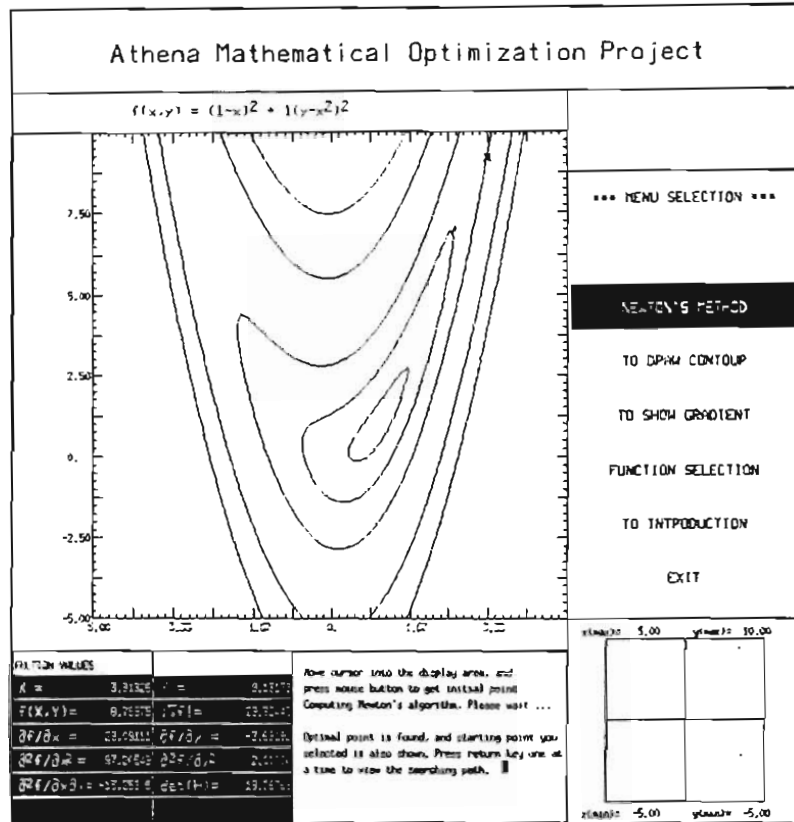


Figure 2.9: Second iteration of Newton's method.

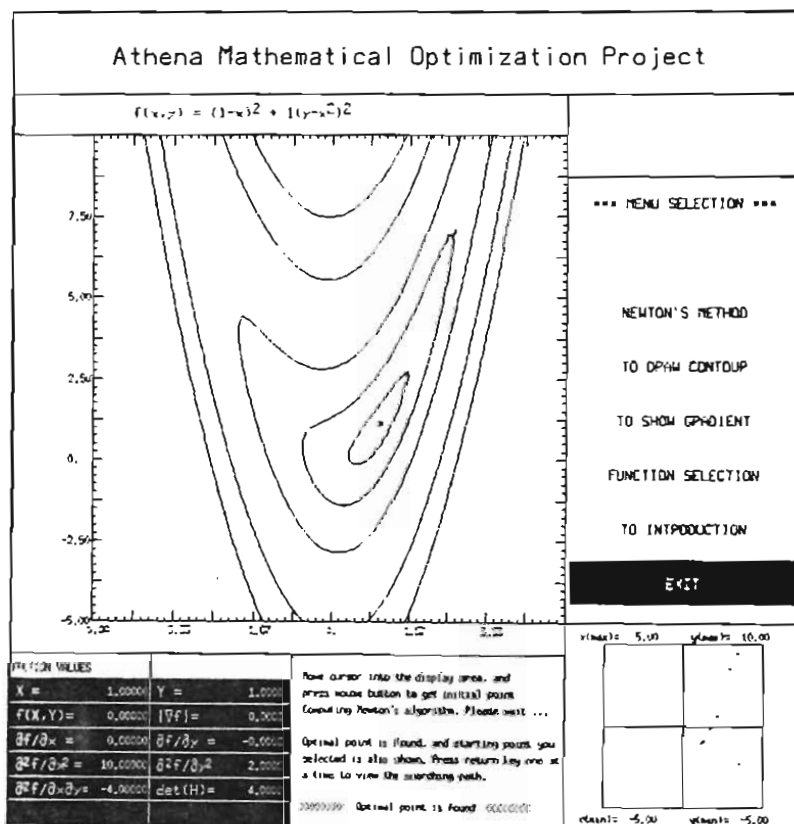


Figure 2.10: Final iteration of Newton's method. Optimum is at *. Small window on the bottom left shows the path followed to the optimum.

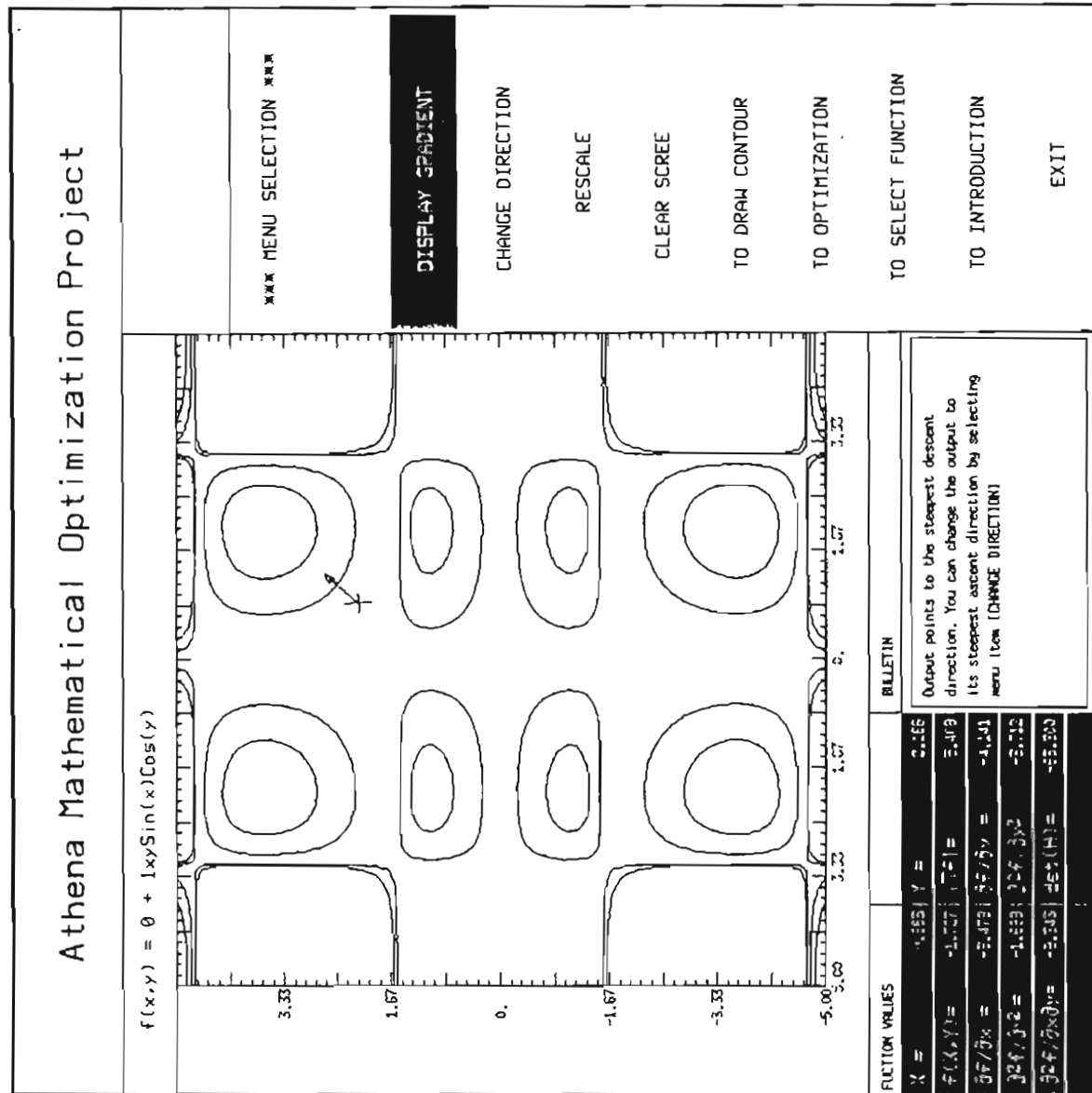


Figure 2.11: Contours and gradient of $f(x, y) = xysinx\cos y$.

like this would behave. Looking at the contours produced by this program, the user obtains some insights on the behavior of the function in question.

Possible future work

There are many directions for further work. Obviously, one is to enrich the library of available functions. This would involve adding a few more Fortran subroutines that define this function, its derivatives and the Hessian. Another direction is to extend this to constrained optimization. Showing the feasible region on the (x, y) plane, together with the contours would assist the user in finding an optimum. Finally, linking with other optimization algorithms would be a worthwhile direction.

Reference

Any decent book on nonlinear programming, e.g. Luenberger (1973).

CHAPTER 3
INTEGER PROGRAMMING

Developer: Jihong Ou

Introduction

Program intprog can be considered a variant of the linear programming package lpprog (see Chapter 1). It allows the user to interactively solve an integer programming problem of two (main) variables and up to 9 (initial) constraints. The user can solve the problem either by branch-and-bound or by cutting planes, and visualize on the screen all steps of either method.

Summary of User Options

- 1) Enter an integer program (IP), either from the terminal, or read from a file.
- 2) Edit the problem before solving.
- 3) Show feasible regions of LP subproblems (in branch-and-bound).
- 4) Show branch-and-bound tree of LP subproblems.
- 5) Allow user to select variable to branch on, and control search mode on the branch-and-bound tree.
- 6) Display all possible (Gomory) cuts and allow user to choose.
- 7) Store the problem in a file.

Instructions

The procedure for entering an IP is much the same as the one described

for LP's (Chapter 1). See Chapter 0 on how to start-up program intprog. Then, select TYPE IN A NEW IP PROBLEM to enter a problem from the terminal, or RECALL A STORED PROBLEM to read one from a file. Entering an IP problem from the terminal is done as in lpprog, except that integrality of variables is implicitly assumed.

Figure 3.1 shows the first screen after an IP is entered. From this point, there are two major menu options, BRANCH AND BOUND and CUTTING PLANE, as far as the solution method is concerned.

Selecting BRANCH AND BOUND presents you with a screen display as the one shown in Figure 3.2. The display consists of a large area (initially showing only an empty box at the top) that is reserved for the branch-and-bound tree, three smaller areas at the bottom, and a menu. The three smaller areas display the following (from left to right): The original IP formulation ; a bulletin board; and a geometric representation of the feasible region of the LP relaxation.

An alternative mode of display is also available by choosing BIG FIGURE AND SMALL TREE from the menu. Figure 3.3 shows this display. In it, the large area of the screen is reserved for the LP feasible region (integer solutions are also shown), and the tree is shown in the smaller sub-area at the bottom. One can go back to the original display by choosing BIG TREE AND SMALL FIGURE from the menu. It is recommended that the first mode of display be chosen, because it gives more details on the branch-and-bound tree. However, one can flip to the other mode if more details on the various LP subproblems are desired.

Choosing FIND AND OPTIMUM allows you to find the optimum of an LP subproblem. To do this, click the mouse either inside the shaded region of the LP subproblem, or inside the box corresponding to this subproblem on the

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | | | | | | | | | | | | | | | | | |
|--|--|--|--|-----------|--|--|-------------|--|--|---------|--|--|-------------|--|--|---|--|
| <p>< USAGE ></p> <ol style="list-style-type: none"> 1. Use only alphanumerical characters, either in lowercases or uppercases for all input. The length of any input string can not exceed 95. all extra input characters will be truncated. 2. Only simple real type data are acceptable, for example: min -12.23x+100y, but not 1.1e3 nor 2+1.3y. And the absolute value of data must between 0.1 and 999.0, otherwise the tableau is not readable 3. When inputting a equality constraints, split it into two inequality constraints of any form 4. If any error occurred, press [Return] key then enter another input to proceed | <p style="text-align: center;">>>> MENU SELECTION <<<</p> <p style="text-align: center;">CHOOSE A METHOD :</p> <div style="background-color: black; color: white; text-align: center; padding: 5px; margin: 5px;">BRANCH & BOUND</div> <p style="text-align: center; margin: 5px;">CUTTING PLANE</p> | | | | | | | | | | | | | | | | |
| <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; padding: 2px;">The Objective:</td> <td style="width: 35%;"></td> <td style="width: 35%;"></td> </tr> <tr> <td style="padding: 2px;">MAX S=s1g</td> <td></td> <td></td> </tr> <tr> <td style="padding: 2px;">Subject to:</td> <td></td> <td></td> </tr> <tr> <td style="padding: 2px;">1) maxS</td> <td></td> <td></td> </tr> <tr> <td style="padding: 2px;">2) S=s1g/45</td> <td></td> <td></td> </tr> </table> | The Objective: | | | MAX S=s1g | | | Subject to: | | | 1) maxS | | | 2) S=s1g/45 | | | <p style="text-align: center; margin: 5px;">HELP</p> <p style="text-align: center; margin: 5px;">HARDCOPY</p> <p style="text-align: center; margin: 5px;">TO INTRODUCTION</p> <p style="text-align: center; margin: 5px;">TERMINATE PROGRAM</p> | |
| The Objective: | | | | | | | | | | | | | | | | | |
| MAX S=s1g | | | | | | | | | | | | | | | | | |
| Subject to: | | | | | | | | | | | | | | | | | |
| 1) maxS | | | | | | | | | | | | | | | | | |
| 2) S=s1g/45 | | | | | | | | | | | | | | | | | |

Figure 3.1: Selecting between branch-and-bound and cutting planes.

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | | | | | | | | | | | | | | | | | |
|---|---|---------------|--|-----------|---|--|-------------|--------------------------|--|---------|-------------------|--|-------------|-----------------------------|--|---|--|
| <div style="border: 1px solid black; width: 100px; height: 100px; margin: 0 auto;"></div> | <p style="text-align: center;">>>> MENU SELECTION <<<</p> <p style="text-align: center;">FIND THE OPTIMUM</p> <p style="text-align: center;">BRANCH</p> <p style="text-align: center;">PROCEED BY THE PROGRAM</p> <div style="background-color: black; color: white; text-align: center; padding: 5px; margin: 5px;">BIG FIGURE/SMALL TREE</div> <p style="text-align: center; margin: 5px;">BIG TREE/SMALL FIGURE</p> <p style="text-align: center; margin: 5px;">RESTART</p> <p style="text-align: center; margin: 5px;">EDIT THE PROBLEM</p> <p style="text-align: center; margin: 5px;">ENTER A NEW PROBLEM</p> | | | | | | | | | | | | | | | | |
| <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%; padding: 2px;">The Objective:</td> <td style="width: 35%; padding: 2px;">-- Remarks --</td> <td style="width: 35%;"></td> </tr> <tr> <td style="padding: 2px;">max S=s1g</td> <td style="padding: 2px;">Each node displays values of 3 parameters as follows:</td> <td></td> </tr> <tr> <td style="padding: 2px;">Subject to:</td> <td style="padding: 2px;">(x) - the coordinates of</td> <td></td> </tr> <tr> <td style="padding: 2px;">1) maxS</td> <td style="padding: 2px;">(y) - the optimum</td> <td></td> </tr> <tr> <td style="padding: 2px;">2) S=s1g/45</td> <td style="padding: 2px;">(equal) - the optimal value</td> <td></td> </tr> </table> <div style="margin-top: 10px;"> <div style="display: inline-block; width: 100px; height: 100px; border: 1px solid black; position: relative;"> <div style="position: absolute; top: 0; left: 0; width: 100%; height: 100%; background: linear-gradient(to top right, transparent 49%, black 49%, black 51%, transparent 51%);"></div> </div> <div style="display: inline-block; vertical-align: top; margin-left: 10px;"> <p>x -> The coordinates of the optimum</p> <p>y -> The optimum</p> <p>equal -> The optimal value</p> </div> </div> <div style="margin-top: 10px;"> </div> | The Objective: | -- Remarks -- | | max S=s1g | Each node displays values of 3 parameters as follows: | | Subject to: | (x) - the coordinates of | | 1) maxS | (y) - the optimum | | 2) S=s1g/45 | (equal) - the optimal value | | <p style="text-align: center; margin: 5px;">HELP</p> <p style="text-align: center; margin: 5px;">HARDCOPY</p> <p style="text-align: center; margin: 5px;">TO INTRODUCTION</p> <p style="text-align: center; margin: 5px;">TERMINATE PROGRAM</p> | |
| The Objective: | -- Remarks -- | | | | | | | | | | | | | | | | |
| max S=s1g | Each node displays values of 3 parameters as follows: | | | | | | | | | | | | | | | | |
| Subject to: | (x) - the coordinates of | | | | | | | | | | | | | | | | |
| 1) maxS | (y) - the optimum | | | | | | | | | | | | | | | | |
| 2) S=s1g/45 | (equal) - the optimal value | | | | | | | | | | | | | | | | |

Figure 3.2: Branch-and-bound: main display mode (I)

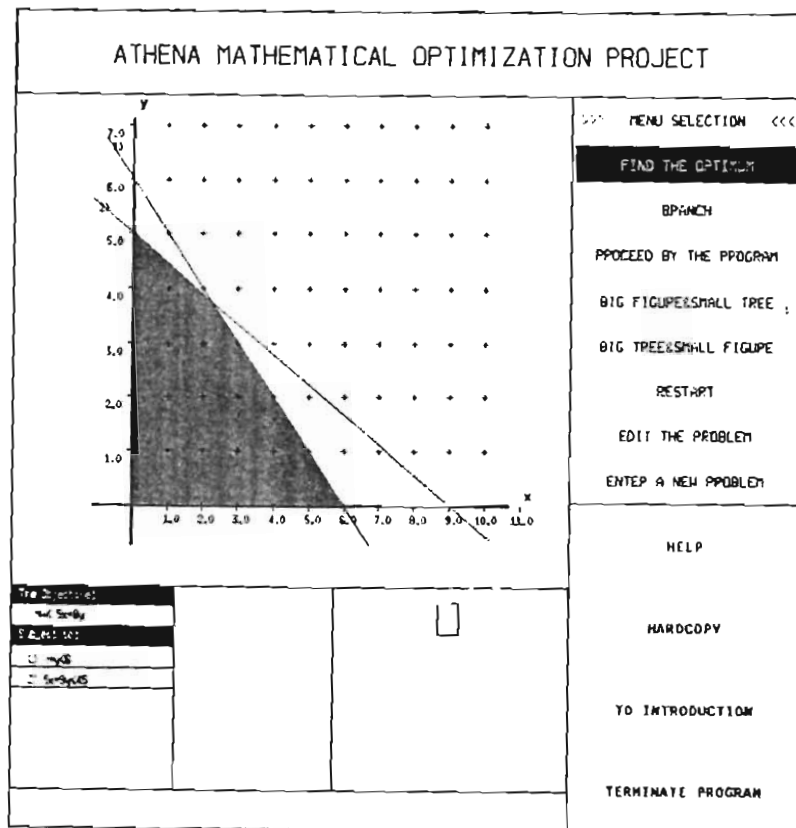


Figure 3.3: Branch-and-bound: alternative display mode (II)

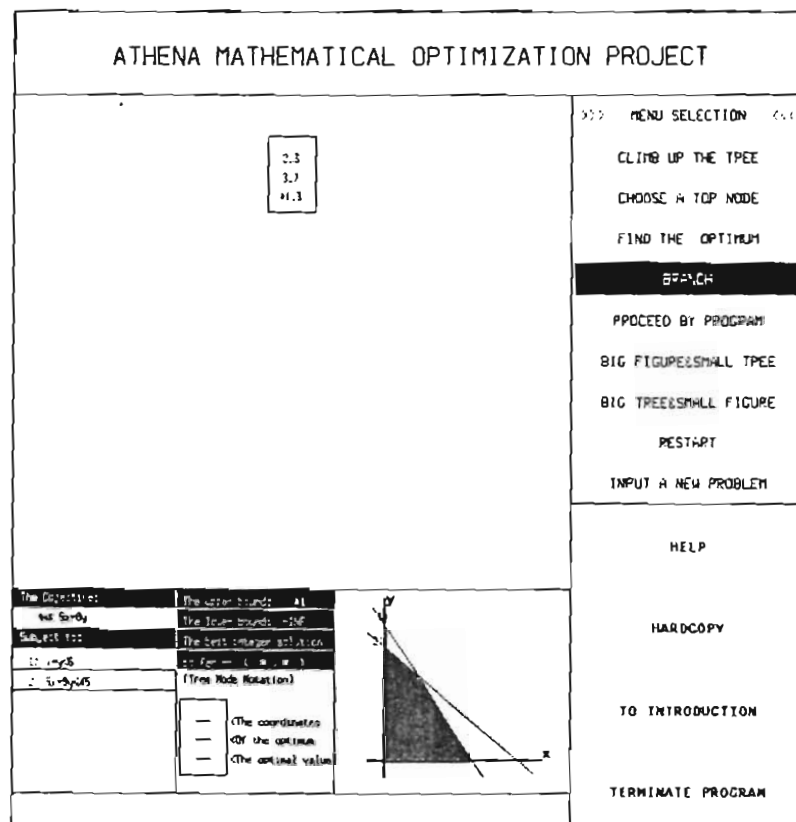


Figure 3.4: Branching

branch-and-bound tree (there is only one such region or box when this procedure is started).

Figure 3.4 shows the result of such an operation for the first (root) LP subproblem. The three numbers in the box show the values of x, y , and the objective function (here they are equal to 2.3, 3.7, and 41.3 respectively). Note also that the bulletin board at the bottom middle of the screen displays the upper bound (here 41), the lower bound (here -) of the optimal value of the problem, as well as the (x, y) values of the best integer solution found so far (asterisks are displayed if none is found).

Selecting BRANCH prompts the user to choose the variable to branch on. Figure 3.5 shows what happens if x is chosen: First, the LP feasible region is separated to two LP subproblems, and second, two new branches are created on the branch-and-bound tree, corresponding to the two alternative constraints (here $x \leq 2$ or $x \geq 3$) associated with the branching.

Figure 3.6 shows the alternative mode of display (BIG FIGURE AND SMALL TREE) corresponding to Figure 3.5.

One can proceed this way by using the FIND THE OPTIMUM and BRANCH commands in succession, and build up the branch-and-bound tree. Figure 3.7 shows a display of the procedure at a later stage. Branches that are fathomed (either by integrality, or by infeasibility, or by bound) are shown with an "F" (notice one such branch in Figure 3.7, and the corresponding values in the bulletin board). The final (optimal) display for this problem is shown in Figure 3.8 (notice that upper and lower bounds are equal). Obviously, it makes no sense to further branch on a branch that has been fathomed, however this option is not forbidden by the program, so that the user can observe why this does not make sense.

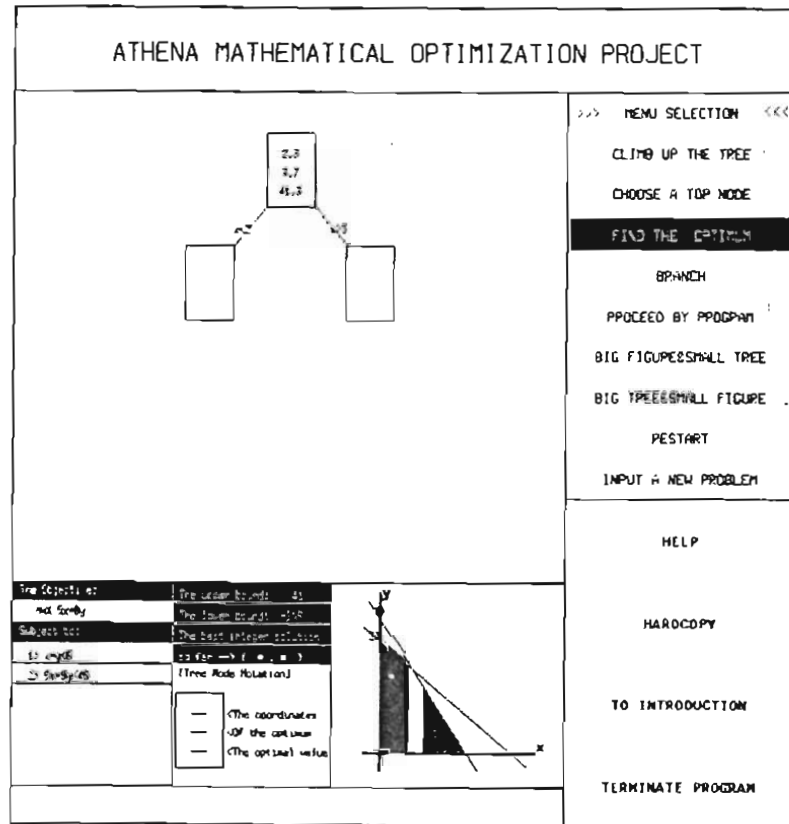


Figure 3.5: Branching and LP subproblems I

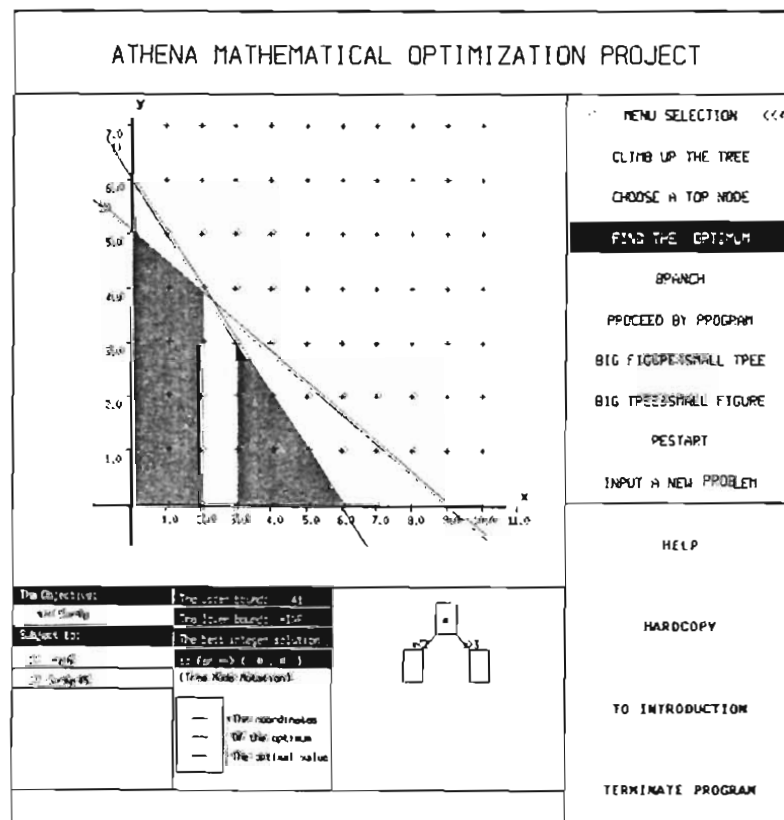


Figure 3.6: Branching and LP subproblems II

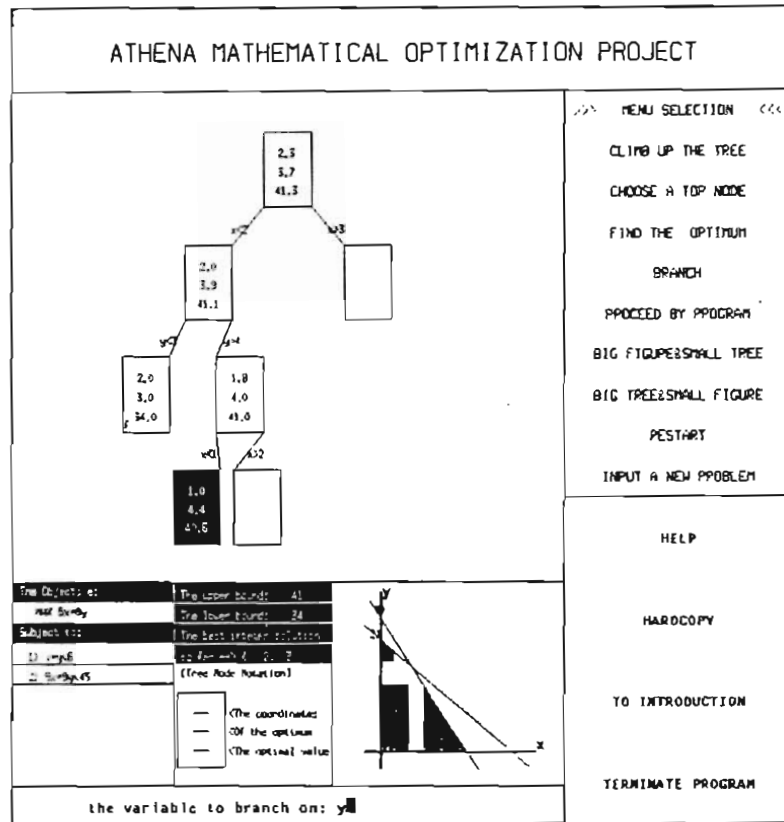


Figure 3.7: An intermediate step in the procedure

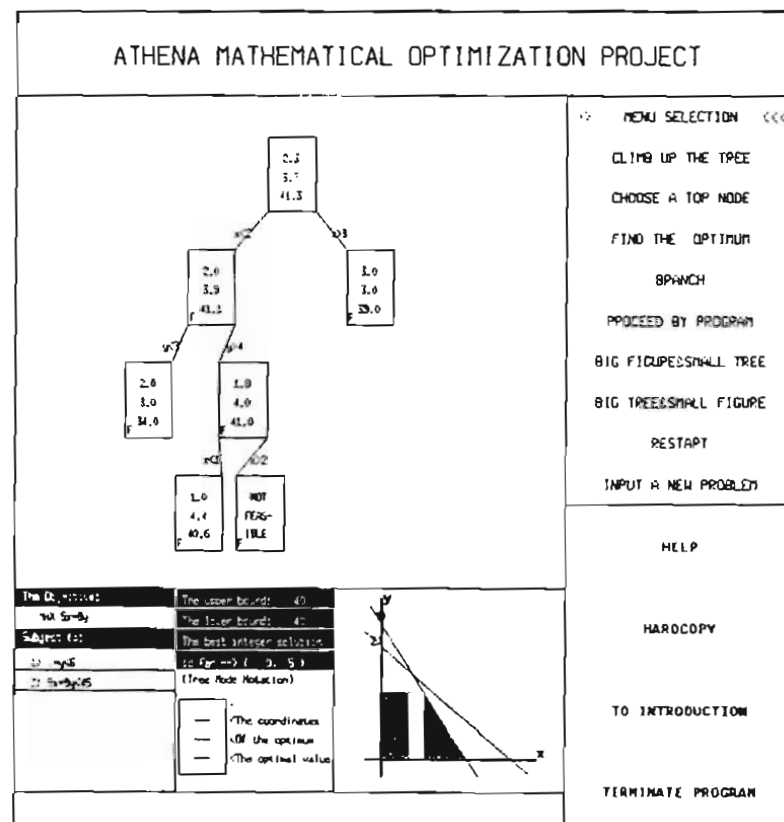


Figure 3.8: Branch-and-bound: final (optimal) display

In exploring the tree, the user has complete freedom on what LP subproblems to solve. Depth-first-search is the "standard" recommended search mode, but other modes can be explored as well.

The PROCEED BY THE PROGRAM option is the "automatic" mode of the package. It proceeds toward the optimum using a depth-first search mode. The user can switch to (and from) that option at any point during the execution of the algorithm.

The screen layout can accommodate trees that have depth of at most 4 (i.e. as many as 15 LP subproblems).. In case the tree depth is more than 4, a new screen is shown, displaying only the bottom part of the tree. The user can move up and down the tree by using the CLIMB UP THE TREE and CHOOSE A TOP NODE commands.

Other menu items for the branch-and-bound case are:

RESTART brings you back to the menu of Figure 3.1, allowing you to solve the same problem in a different way.

EDIT THE PROBLEM brings you to make changes before the problem is solved. This option is similar to the one in lpprog (see Chapter 1).

INPUT A NEW PROBLEM brings you back to the very first menu, so you can input a new problem.

Solving an integer program by cutting planes (using Gomory cuts) can be done by choosing CUTTING PLANE from the menu of Figure 3.1. Doing this presents you with a display such as the one shown in Figure 3.9, where the LP feasible region and the problem formulation are displayed.

To proceed, select SOLVE THE PROBLEM, and the result appears in Figure 3.10, where the Gomory cuts corresponding to the (fractional) LP solution are listed. At this point, the user can one among the possible cuts. To do this,

the CHOOSE THE CUT item must be selected, and then the program prompts the user for the cut to be chosen (in Figure 3.10), the first cut, i.e. $4x + 7y \leq 35$, is chosen).

The user has a chance to observe the effect of this cut before it is actually taken. The dashed line in Figure 3.1 shows a geometric representation of the cut. To zoom in on the cut, select DETAILS ABOUT THIS CUT. This brings you to Figure 3.12, which not only shows a magnification of the cut, but also the equations representing the cut in the Simplex tableau.

The user may or may not wish to proceed with a cut. To proceed, select TAKE THIS CUT, otherwise select DO NOT TAKE IT → RETURN.

Figure 3.13 shows the optimal solution of the problem, after repeated cuts have been applied. The cuts themselves are also shown, both geometrically, and appended as constraints to the original problem.

Possible Future Work

Adding an "implicit enumeration" module is a straightforward and probably worthwhile task. This would complete the basic integer programming toolkit.

The branch-and-bound package could also be extended to the case of more variables. In this case, the display would be limited just to the representation of the tree, and, possibly, to a plot of the upper and lower bounds versus iteration number. The educational value of such an extension would be to let the user explore various tree search modes in order to find the optimum as quickly as possible.

References

Any decent book on integer programming, e.g. Bradley, Hax, Magnanti (1975) (Chapter 9), Garfinkel and Nemhauser (1973).

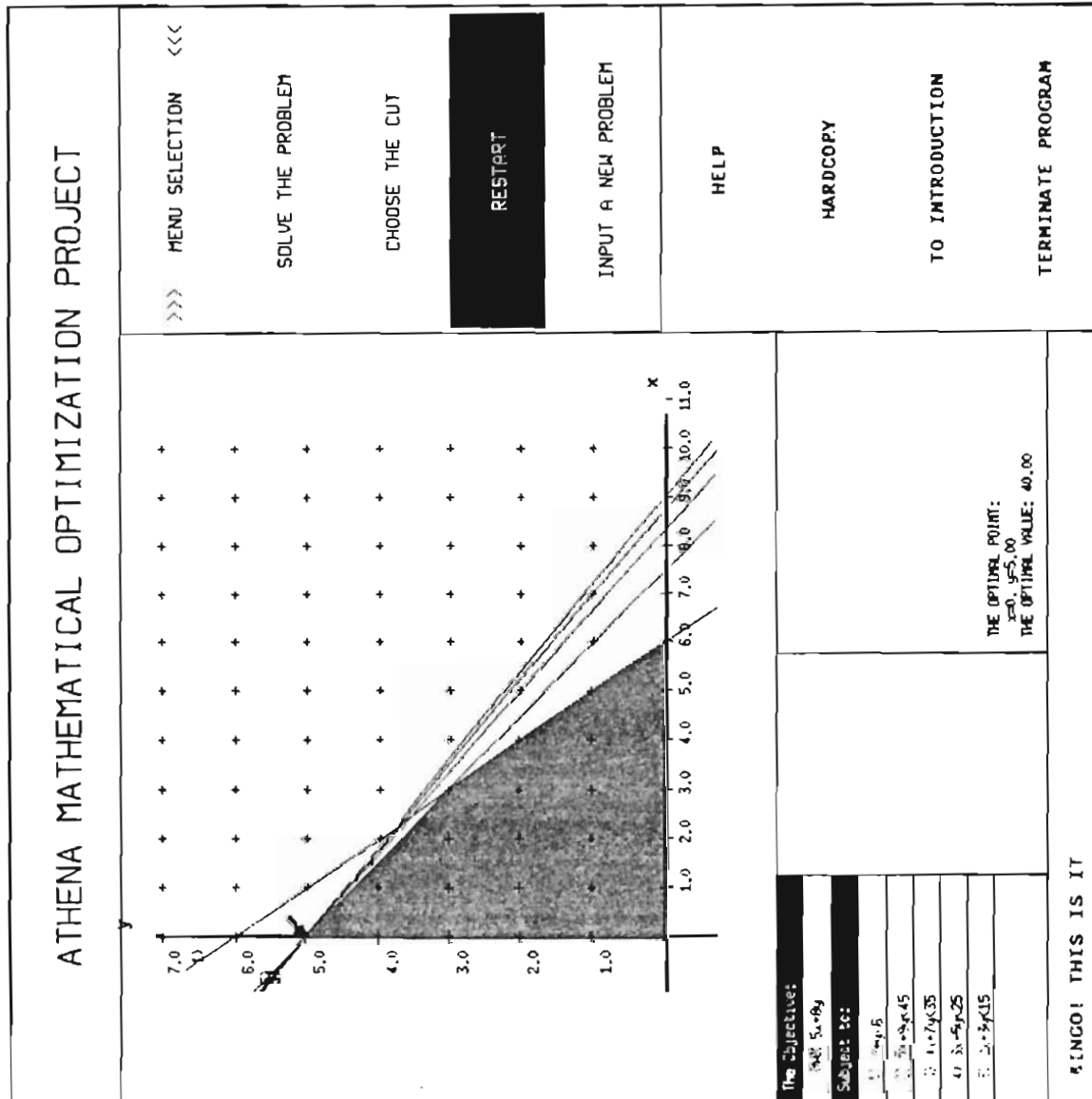


Figure 3.13: Cutting planes: final (optimal) display

CHAPTER 4

THE TRANSPORTATION PROBLEM

This page should
appear on the
right side

Developer: Christopher Hrut

Introduction

The purpose of program transp is to allow the user to interactively solve a minimum cost "transportation" problem. Maximum problem size (limited by the resolution of the screen) is about 10 supply nodes and 10 demand nodes. Larger problems can be solved (non-graphically) by other LP packages such as LINDO. All steps of the solution are displayed on the screen.

Summary of User Options

- 1) Enter a transportation problem from the terminal
- 2) Edit the problem before solving
- 3) Select an initial solution by a variety of methods, including arbitrarily.
- 4) Pivot and improve the solution step by step.
- 5) At each step, observe flows, dual prices, and other relevant information.

Instructions

Throughout this application, menu items preceded by the >> symbol typically correspond to taking the next step toward solving the problem. If you are lost and do not know how to proceed, choose a >> menu item and you will move through the algorithm step by step. Other menu items typically provide auxiliary functions, such as displaying certain information, editing the

problem, etc.

Before you proceed, select HELP and TABLE NOTATION for important information on how things are displayed in table form (see Figure 4.1). Tables are used throughout this application (as a complement to network geometric representations) to display information such as costs, reduced costs, flows, dual prices (shadow prices), etc.

To enter a transportation problem, go back to main menu and select >> DATA ENTRY. You will then be prompted to enter the following information, in this order:

- . number of supply nodes (suggest maximum 10)
- . number of demand nodes (suggest maximum 10)
- . name and amount of supply for each supply node (see Figures 4.2 and 4.3)
- . name and amount of demand for each demand node
- . unit transport costs for each supply-demand pair (see Figure 4.4)

Note that total demand should not necessarily equal total supply. If these are not equal, a dummy node will be added later (as in this example).

The next display is as in Figure 4.5. Note that the menu has three >> items. Of these, OPTIMAL SOLUTION *table* and OPTIMAL SOLUTION *network* are "automated" modes, to be described later. DISPLAY NETWORK switches to a display that shows the network corresponding to the problem at hand (this menu item is present in all subsequent menus). DISPLAY TABLE switches back to the table representation, as in Figure 4.5. DATA CHANGES allows you to edit the problem before it is solved, by modifying any of the input data.

To proceed, select INITIAL SOLUTION from menu. This allows you to specify an initial feasible solution for the problem. Options here are the

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|---|--|
| <p style="text-align: center; margin-bottom: 20px;">EXPLANATION OF THE TABLE FORMULATION</p> <div style="text-align: center;"> </div> | <p style="text-align: center; margin-top: 20px;">** HELP **</p> <p style="text-align: center; margin-top: 20px;">INTRODUCTION</p> <p style="text-align: center; margin-top: 20px;">TABLE NOTATION</p> <p style="text-align: center; margin-top: 20px;">GO TO PREVIOUS MENU</p> <p style="text-align: center; margin-top: 20px;">EXIT</p> |

Figure 4.1: Table notation (explanation of display)

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|----------------|--|--|--|----------------|----------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| <p style="text-align: center; margin-bottom: 10px;">TYPE NAME OF A SUPPLY NODE</p> <p style="text-align: center; margin-bottom: 10px;">TYPE quit TO QUIT</p> <div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: 80%;"> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td style="width: 10%; font-size: small;">Supply node</td> <td style="width: 20%; height: 20px;"></td> <td style="width: 20%; height: 20px;"></td> <td style="width: 20%; height: 20px;"></td> <td style="width: 20%; height: 20px;"></td> <td style="width: 10%; font-size: small;">Demand node</td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> </table> </div> | Supply node | | | | | Demand node | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | <p style="text-align: center; margin-top: 20px;">** DATA ENTRY **</p> <p style="text-align: center; margin-top: 20px;">HELP</p> <div style="border: 1px solid black; padding: 5px; text-align: center; margin-top: 20px;"> <p>>> DATA ENTRY</p> </div> <p style="text-align: center; margin-top: 20px;">HARDCOPY</p> <p style="text-align: center; margin-top: 20px;">EXIT</p> |
| Supply node | | | | | Demand node | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 4.2: Data entry: Name of supply node

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | | | | | | | | | |
|--|--|--|--|---|---|--|--|--|--|
| TO PROCEED, PRESS "INITIAL SOLUTION" | | | | | | | | | |
| <div style="border: 1px solid black; padding: 2px;"> 10 20 35 25 30 </div> | <div style="border: 1px solid black; padding: 2px;">10</div> <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">15</div> <div style="border: 1px solid black; padding: 2px;">25</div> <div style="border: 1px solid black; padding: 2px;">30</div> | <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">30</div> <div style="border: 1px solid black; padding: 2px;">35</div> <div style="border: 1px solid black; padding: 2px;">15</div> <div style="border: 1px solid black; padding: 2px;">20</div> | <div style="border: 1px solid black; padding: 2px;">10</div> <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">25</div> <div style="border: 1px solid black; padding: 2px;">35</div> <div style="border: 1px solid black; padding: 2px;">15</div> | <div style="border: 1px solid black; padding: 2px;"> Athens 20 30 Paris 10 London 10 </div> | <div style="border: 1px solid black; padding: 2px;"> New York 30 Boston 20 Washington 15 </div> | <div style="border: 1px solid black; padding: 5px;"> ** TRANSPORTATION ** PROBLEM HELP DISPLAY NETWORK DISPLAY TABLE <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> INITIAL SOLUTION </div> DATA CHANGES >> OPTIMAL SOLUTION #table# >> OPTIMAL SOLUTION #network# GO TO MAIN MENU EXIT </div> | | | |

Figure 4.5: First menu after all data are in

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | | | | | | | | | |
|--|--|--|--|---|---|---|--|--|--|
| SELECT INITIAL SOLUTION | | | | | | | | | |
| <div style="border: 1px solid black; padding: 2px;"> 10 20 15 25 30 </div> | <div style="border: 1px solid black; padding: 2px;">10</div> <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">15</div> <div style="border: 1px solid black; padding: 2px;">25</div> <div style="border: 1px solid black; padding: 2px;">30</div> | <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">30</div> <div style="border: 1px solid black; padding: 2px;">35</div> <div style="border: 1px solid black; padding: 2px;">15</div> <div style="border: 1px solid black; padding: 2px;">20</div> | <div style="border: 1px solid black; padding: 2px;">10</div> <div style="border: 1px solid black; padding: 2px;">20</div> <div style="border: 1px solid black; padding: 2px;">25</div> <div style="border: 1px solid black; padding: 2px;">35</div> <div style="border: 1px solid black; padding: 2px;">15</div> | <div style="border: 1px solid black; padding: 2px;"> Athens 20 30 Paris 10 London 10 </div> | <div style="border: 1px solid black; padding: 2px;"> New York 30 Boston 20 Washington 15 </div> | <div style="border: 1px solid black; padding: 5px;"> ** INITIAL SOLUTION ** HELP <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> NORTH-WEST CORNER METHOD </div> <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> MINIMUM COLUMN METHOD </div> <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> MINIMUM ROW METHOD </div> <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> MINIMUM MATRIX METHOD </div> <div style="border: 1px solid black; padding: 2px; text-align: center;"> >> USER SUPPLIED SOLUTION </div> DISPLAY TABLE DISPLAY NETWORK GO TO PREVIOUS MENU EXIT </div> | | | |

Figure 4.6: Menu of possible initial solution methods

Northwest Corner, Minimum Column, Minimum Row, and Minimum Matrix Methods, as well as a user-supplied solution (see Figure 4.6). Chapter 8 of Bradley, Hax, Magnanti (1977) describes these methods. In our example, the user has selected the minimum matrix method, and the result is shown in Figure 4.7. All nonzero flows are shown in reverse video (white on black). Notice the appearance of the dummy node that balances supply with demand.

One can see the resulting initial solution by selecting DISPLAY NETWORK. Note that the basis corresponding to this solution may be degenerate. This will be the case if the network shown is not a spanning tree. To show the basis, select DISPLAY SPANNING TREE. If basis is degenerate, zero flow (degenerate) arcs are shown in thin lines (see Figure 4.13 later on for an example of a degenerate basis).

Once an initial solution has been chosen, select IMPROVING SOLUTION. You will be then presented with a series of menus, for which the >> entries are (in cyclical sequence): CALCULATE SHADOW PRICES, CALCULATE REDUCED COSTS, SHOW ENTERING BASIS ELEMENT, DISPLAY LOOP, DISPLAY EXISTING BASIS ELEMENT, DISPLAY NEW FLOWS NETWORK, and DISPLAY NEW SPANNING TREE. These steps correspond to a typical pivot operation. You can display table or network at any of the above steps. The entering basis element is chosen by the program to be the one with the most negative reduced cost, unless it is overridden by the user, who can choose his/her own entering variable, and see what happens.

Figures 4.8 to 4.12 show a sample of screens in a typical pivot operation. Figure 4.8 shows the table display after the CALCULATE SHADOW PRICES and CALCULATE REDUCED COSTS menu items have been selected. Negative reduced costs are shown inside small boxes, and correspond to candidate entering basis elements. Figure 4.9 shows the equivalent network display, showing flow and

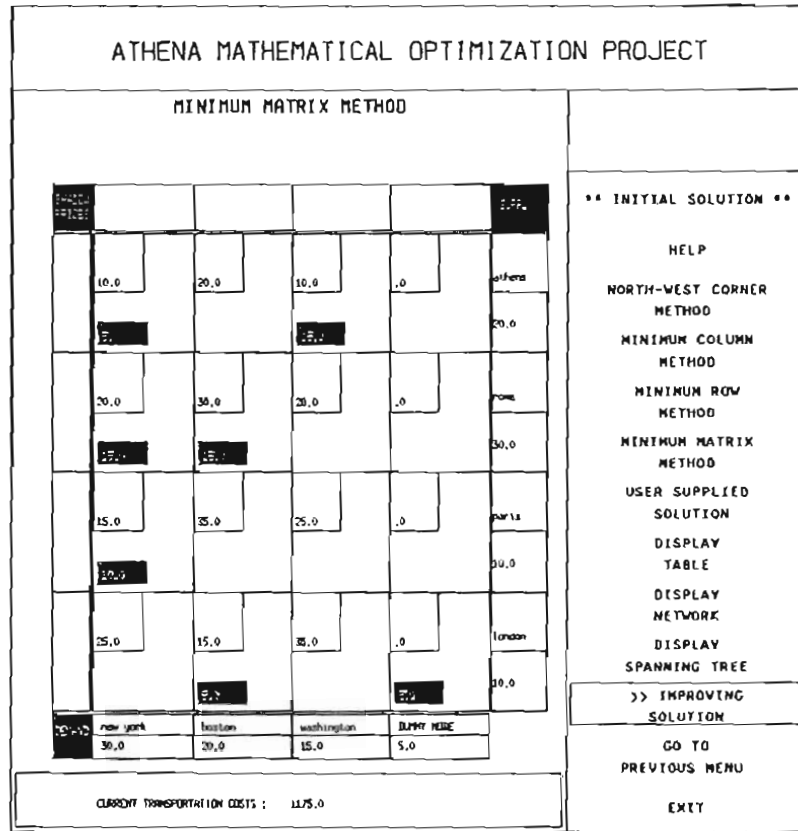


Figure 4.7: Nonzero flows are shown in reverse video

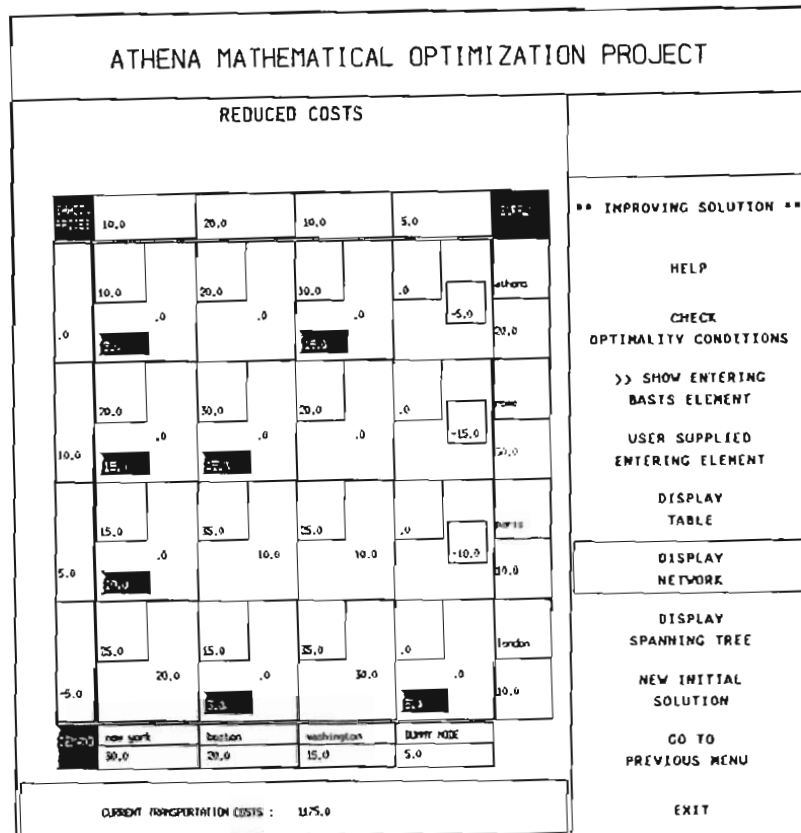


Figure 4.8: Updated table display showing shadow price information

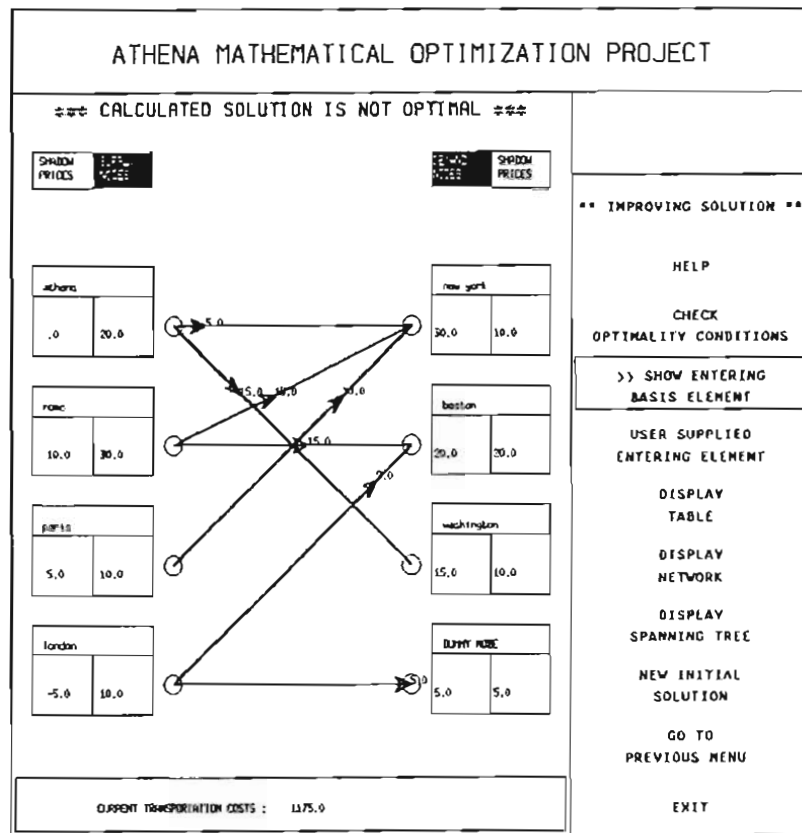


Figure 4.9: Equivalent network display: current flows are not optimal

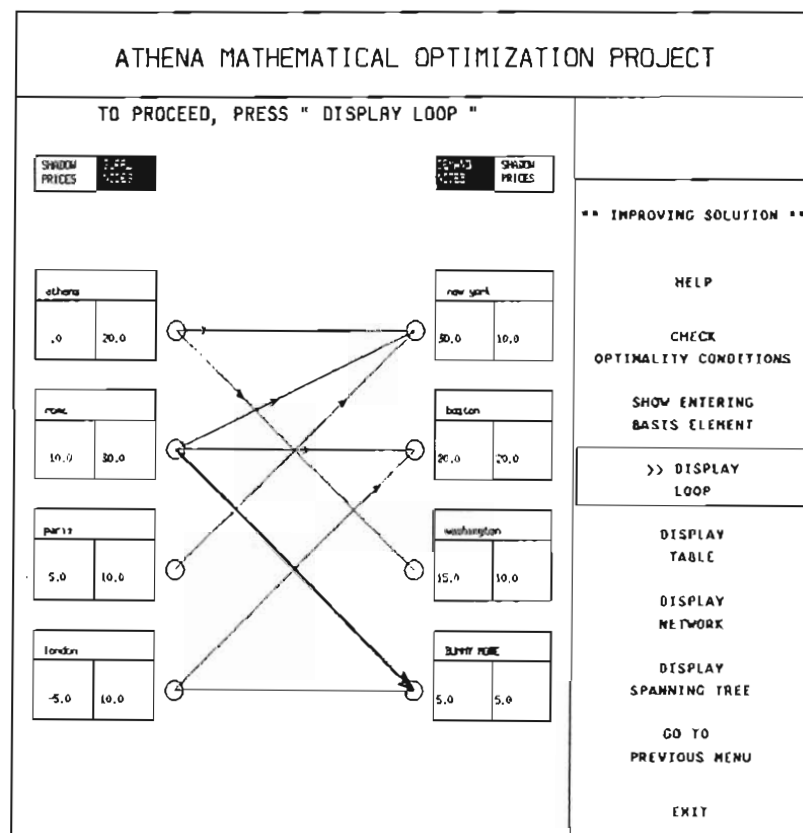


Figure 4.10: Arc entering the basis

dual price information (this display, is shown after the CHECK OPTIMALITY CONDITIONS item is selected. Notice the message at the top of the screen).

Figure 4.10 identifies the arc that will enter the basis (here, from "rome" to the "DUMMY NODE", corresponding to the most negative reduced cost of -15). Figure 4.11 shows the unique loop that is formed by adding this arc into the basis (together with the flow in this loop), and Figure 4.12 shows the arc that drops from the basis.

Repeated application of the steps outlined above will eventually lead to an optimal solution, with a maximum amount of information available at each step of the procedure.

In addition to the above "manual" mode, and as already mentioned earlier, there is an "automated" mode for this package. This is executed by selecting the OPTIMAL SOLUTION *network* or the OPTIMAL SOLUTION *table* commands. Both options run you through the steps of the procedure, showing you the problem in a network or table form (respectively) for each iteration. These options are only recommended if you are in a rush, or have a good excuse. Both options require that you specify an initial solution along the lines already described.

Figure 4.13 shows a typical display at the final iteration of the OPTIMAL SOLUTION *network* mode. (note that the arc from "rome" to "new york" is degenerate).

Directions for further work

As with lpprog, anticycling is not guaranteed in this package. Degeneracy should present no problem in most of the cases, however there are some problematic cases where cycling might occur. Therefore, adding an

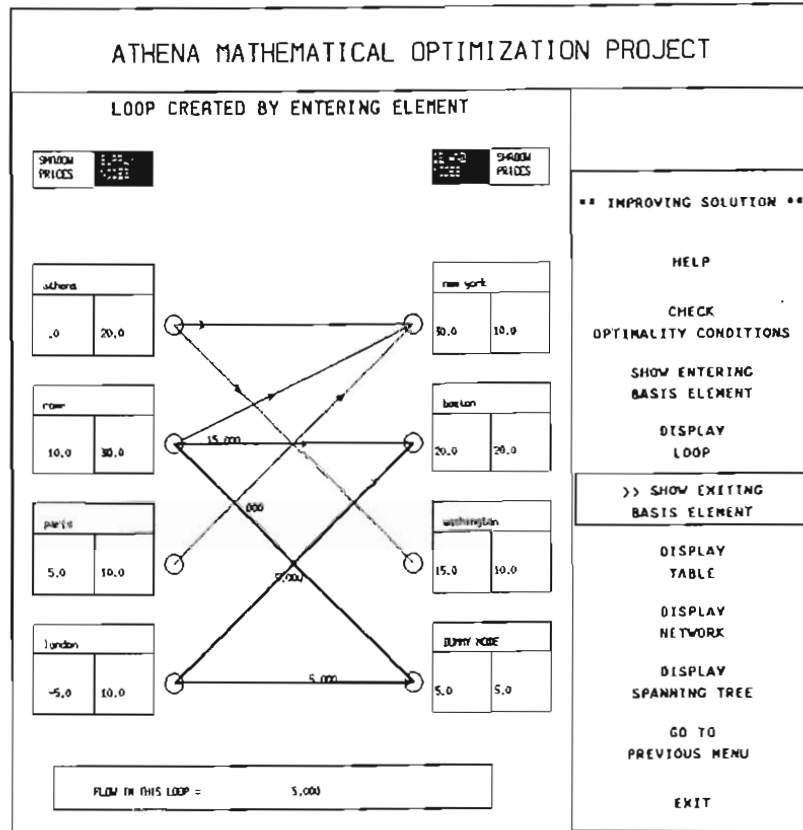


Figure 4.11: Loop created by adding entering arc

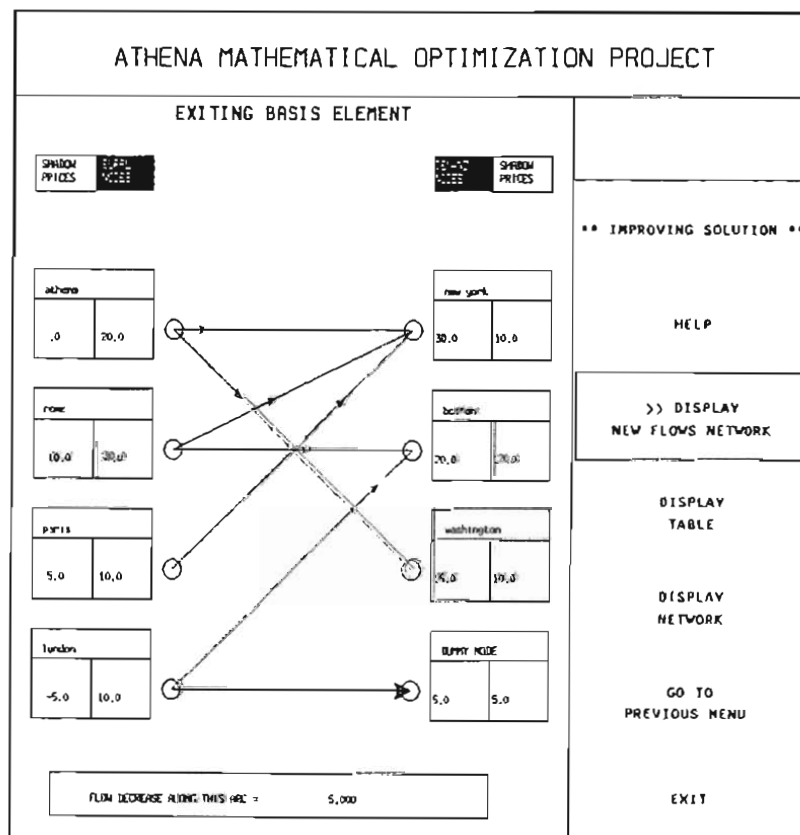


Figure 4.12: Arc dropping from basis

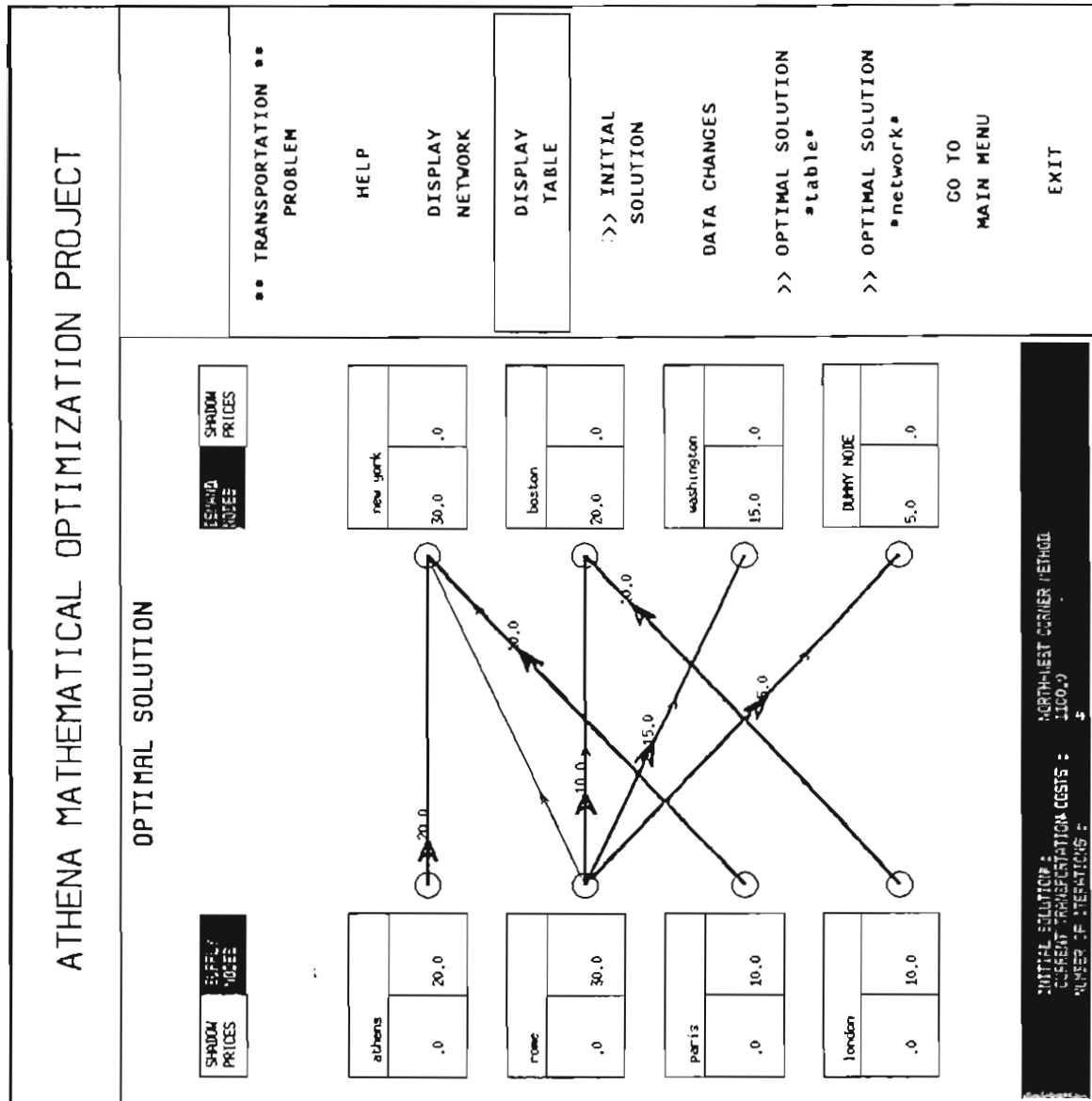


Figure 4.13: Final iteration of the OPTIMAL SOLUTION *network* mode.

anticycling module is a worthwhile task.

The use of table displays was chosen so as to conform with the approach used in Bradley, Hax Magnanti (1977), which is the textbook for academic subjects in which this software is used. Network displays can also be chosen by the user, however most of the vital information (such as reduced costs) is shown only in the table display. An alternative way of displaying such information in a network display mode is described in conjunction with the minimum cost flow problem in Chapter 6.

Reference

Ch. 8 of Bradley, Hax, Magnanti (1977).

CHAPTER 5

THE SHORTEST PATH PROBLEM

Developer: Christopher Hrut

Introduction

The purpose of package spm is to allow the user to interactively solve the shortest path problem on a given network. A "network editor" allows for interactive input and modification of nodes, arcs, links, and costs. Algorithmic options include Dijkstra's algorithm (for all shortest paths from a given node) and Floyd's algorithm (for shortest paths among all pairs of nodes). All steps of either algorithm are displayed on the screen.

Summary of User Options

- 1) Input a given network, either from the terminal or from a file.
- 2) Edit the network on the screen.
- 3) Test the network for connectivity.
- 4) Apply Dijkstra's algorithm for all shortest paths from a given node.
- 5) Apply Floyd's algorithm for shortest paths among all pairs of nodes.
- 6) Display algorithmic and network information along all iterations.
- 7) Store the network in a file.

Instructions

As with the transportation problem (Chapter 4), menu items preceded by the >> symbol typically correspond to taking the next step toward solving the

problem. Other menu items provide other auxiliary functions.

After transp is evoked (see Chapter 0), you get a menu listing HELP, DATA ENTRY, GO TO MAIN MENU, HARDCOPY, EXIT AND DATA FILES as selections. Of those, GO TO MAIN MENU and HARDCOPY are currently disabled. HELP will get you into the help menu which provides information on the main functions of this package. EXIT AND DATA FILES will get you into a menu that allows you to terminate the program (YES, TERMINATE OUR SESSION), WRITE DATA INTO A FILE, or READ DATA FROM A FILE. If a network has been previously stored into a file, provide the name of that file and the network will be read by the program (see also later on).

DATA ENTRY gets you into a "network editor" menu, as shown in Figure 5.1. This will allow you to define (and edit) a new network, by specifying its nodes, links (or arcs, as appropriate), and costs. Start with DRAW NODES, select node size (see Figure 5.2), and then click the mouse cursor in the display work area to define nodes. In Figure 5.3, three nodes are shown. Two of them (labeled 1 and 2) have already been created, and the user is just entering the label (name) of the third node (in this example 34). Node "labels" here just refer to node names (1 to 99), not to Dijkstra labels. It is not necessary to number nodes sequentially.

Select DONE when all nodes have been defined (Figure 5.4) and proceed by choosing DELETE NODES, DRAW LINKS, DELETE LINKS, DRAW ARCS, DELETE ARCS, or MOVE NODE, as appropriate. To draw a link between two nodes, select DRAW LINKS, click the mouse inside one of the nodes, move the cursor inside the other node, and click the mouse again. An example is shown in Figure 5.5, where the user has just defined a link between nodes 1 and 34. After a link is defined, the user is prompted to enter the cost (or length) of the link (in this example, it

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|--|---|
| SELECT MENU | |
| | <div>>> DRAW NODES</div> <div>Delete NODES</div> <div>DRAW LINKS</div> <div>Delete LINKS</div> <div>DRAW ARCS</div> <div>Delete ARCS</div> <div>MOVE NODE</div> <div>DISPLAY NETWORK</div> <div>Delete NETWORK / DATA FILES</div> <div>>> DATA TESTING</div> <div>>> DONE: GO TO ALGORITHMS</div> <div>GO TO MAIN MENU</div> <div>HARDCOPY</div> <div>EXIT AND DATA FILES</div> |

Figure 5.1: Basic network editor menu

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|--|--|
| PRESS TWICE IN THE NODE CENTER OR 'DONE' | |
| | <div>HELP</div> <div>SMALL WHITE CIRCLE</div> <div>MEDIUM WHITE CIRCLE</div> <div>LARGE WHITE CIRCLE</div> <div>>> DONE</div> <div>EXIT AND DATA FILES</div> |

Figure 5.2: Node size selection

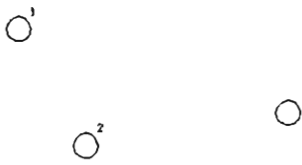
| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|---|---------------------|
|  | HELP |
| | SMALL WHITE CIRCLE |
| | MEDIUM WHITE CIRCLE |
| | LARGE WHITE CIRCLE |
| ===== | |
| TYPE THE LABEL OF THIS NODE : 1..99 [input field] | >> DONE |
| | EXIT AND DATA FILES |

Figure 5.3: Node location and number (label) definition

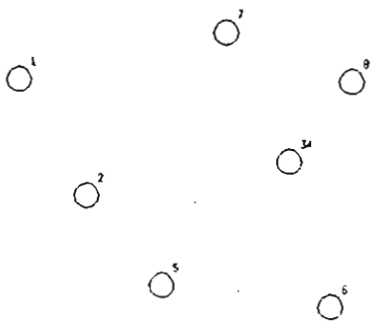
| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|---|---------------------|
| PRESS IN THE NEXT NODE CENTER OR SELECT MENU | |
|  | HELP |
| | SMALL WHITE CIRCLE |
| | MEDIUM WHITE CIRCLE |
| | LARGE WHITE CIRCLE |
| ===== | |
| | >> DONE |
| | EXIT AND DATA FILES |

Figure 5.4: After all nodes are defined, select DONE

is 25). Similar considerations apply for arcs, with the additional point that direction is now important. MOVE NODE allows you to reconfigure the location of a node on the screen (and all of its incident edges) for a better display (no other changes in input data are effected). DELETE NODE deletes a node and all of its incident edges. To change the cost of an edge (link or arc) you have to delete that edge (using DELETE LINK or DELETE ARC) and then reenter it.

Figure 5.6 shows a (mixed) network read from a file, and Figure 5.7 shows the same network after node 5 has been moved to a new location.

Before you proceed to the algorithms menu (DONE: GO TO ALGORITHMS), you should test the data (DATA TESTING) to check whether the defined graph is a legitimate one, in terms of connectivity. Several tests are available, as shown in Figure 5.8. REACHABILITY FROM SELECTED NODE shows you every node that is reachable from a selected node via some path. GENERAL REACHABILITY identifies nodes that are (or are not) reachable from other nodes of the network. You may also want to identify NODES WITHOUT EXITING ARCS or NODES WITHOUT ENTERING ARCS. The REACHABILITY MATRIX has entries of one or zero depending on whether or not there is a path between a specified pair of nodes (see Figure 5.9). Figure 5.10 shows the cost (distance) matrix for this network (and is displayed after DISPLAY COST TABLE is selected).

The algorithm menu (Figure 5.11) includes INTERACTIVE DIJKSTRA, NONINTERACTIVE DIJKSTRA, and FLOYD as its main options (ALGORITHM 3 is currently disabled). The difference between the two Dijkstra options is that the former presents all steps of Dijkstra's algorithm in a detailed, maximum-information mode that allows the user to inquire about almost anything at every step, whereas the latter displays only the tree of shortest paths at intermediate steps. The INTERACTIVE option is recommended for beginners.

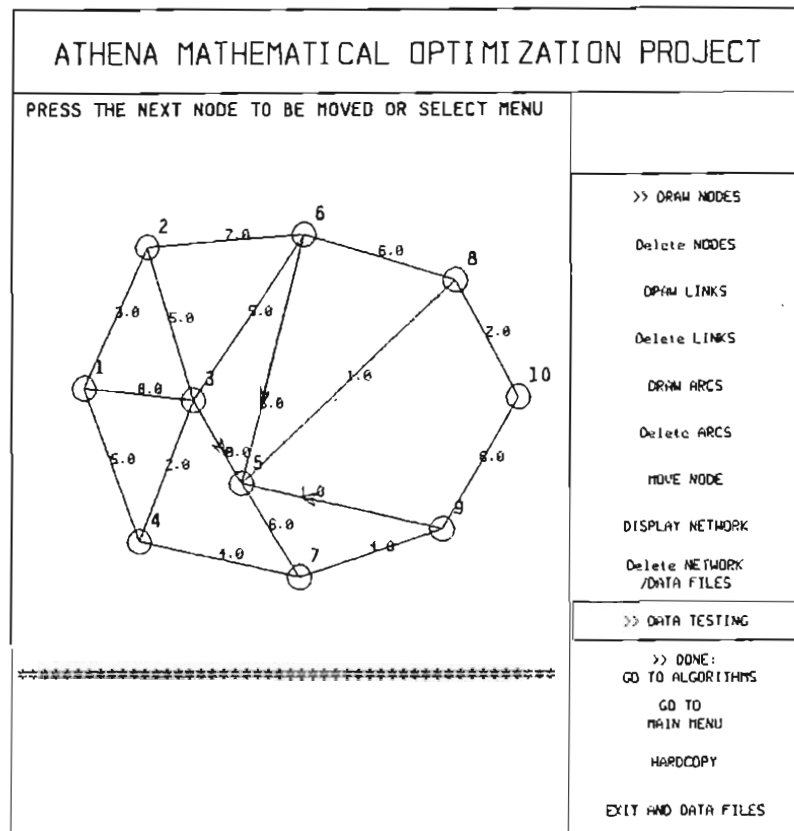


Figure 5.7: Move node 5 to new location, and proceed with data testing.

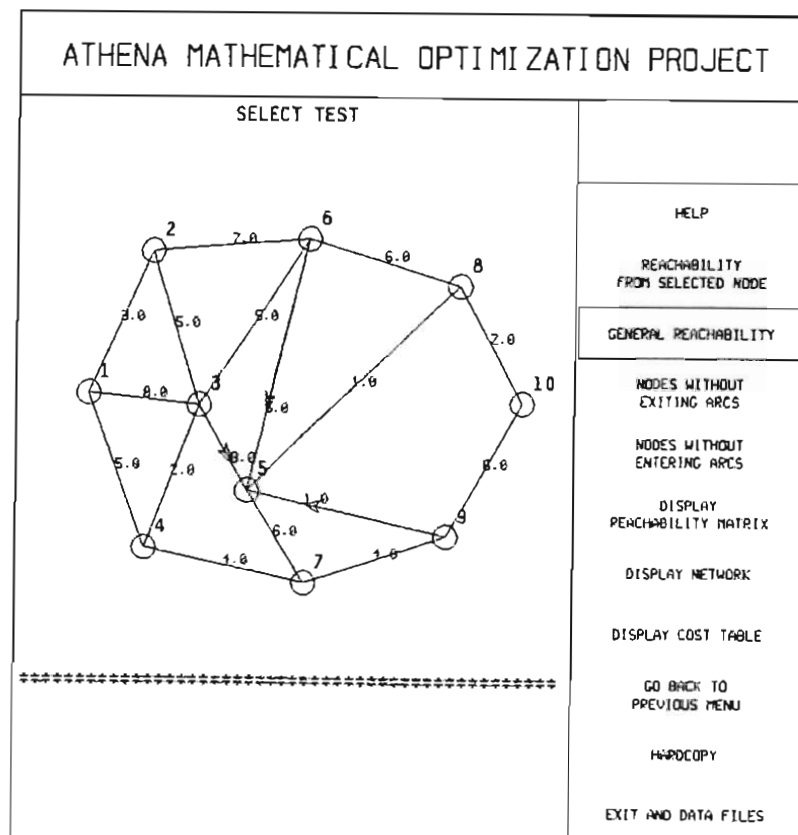


Figure 5.8: Data testing menu

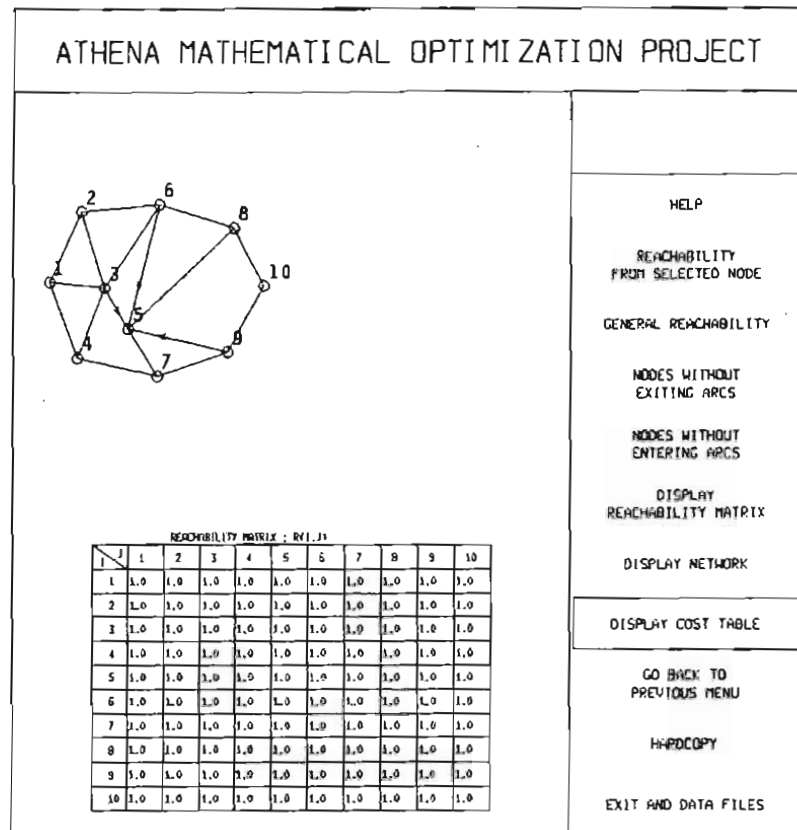


Figure 5.9: Reachability matrix: An entry of 1.0 means there is a path between a specified (ordered) pair of nodes.

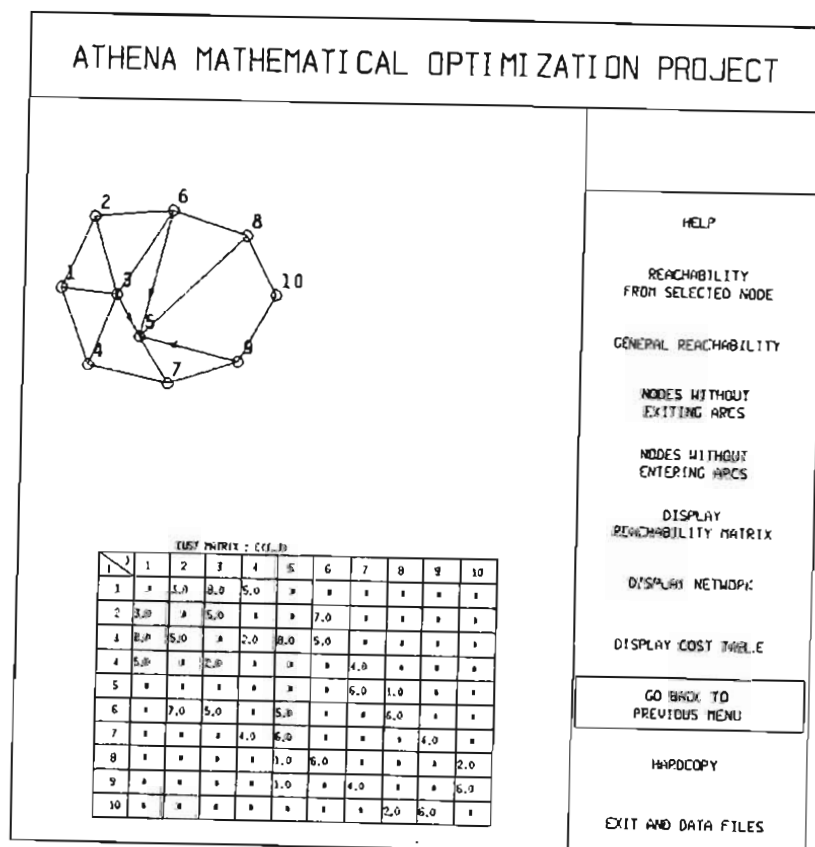


Figure 5.10: Distance (cost) matrix

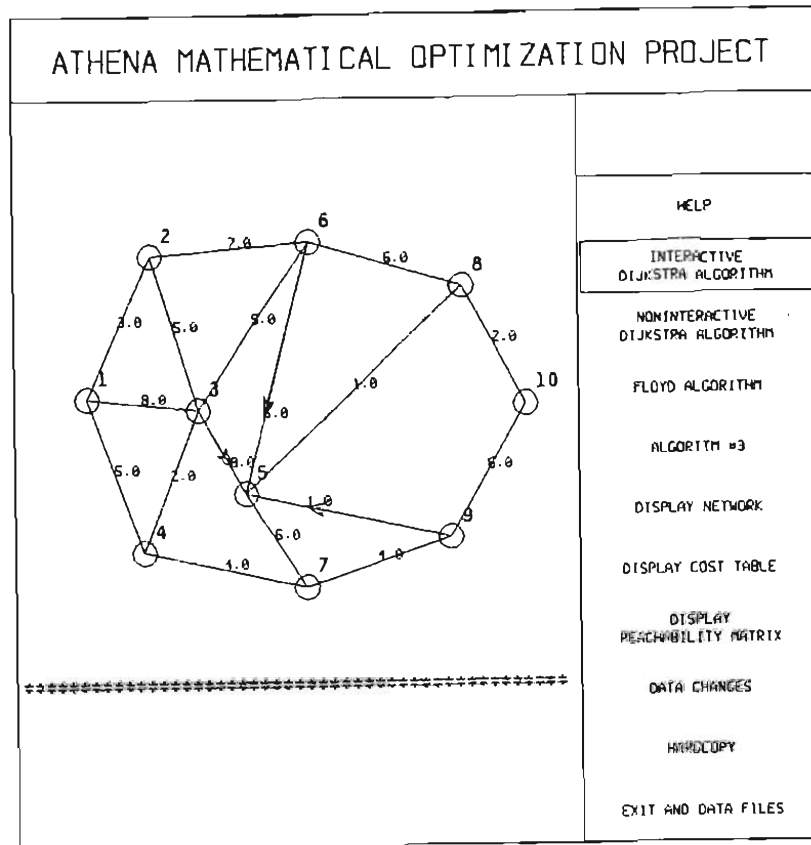


Figure 5.11: Basic algorithm menu

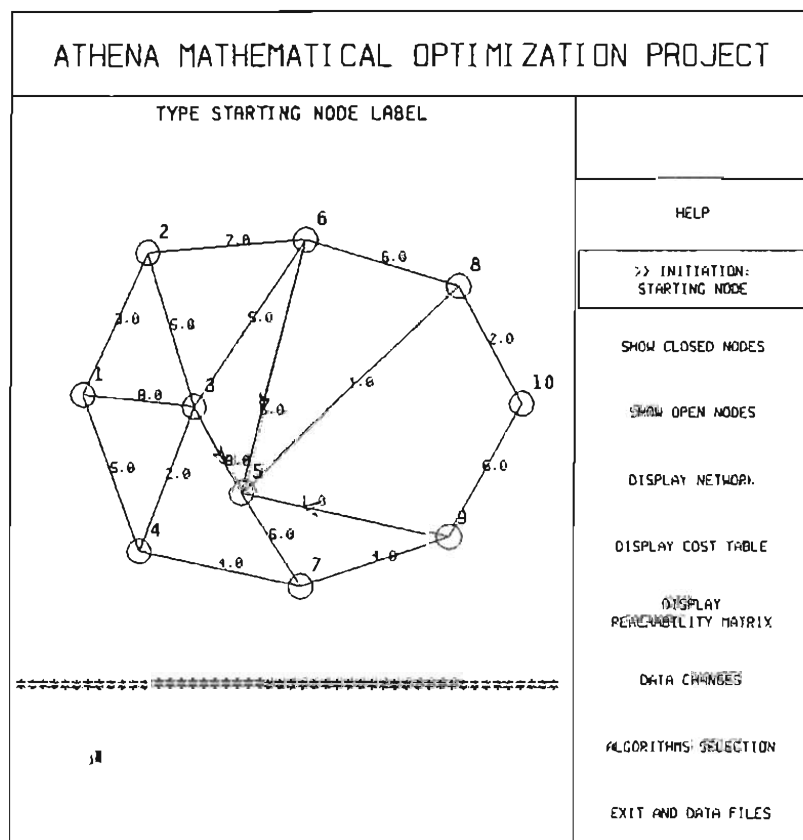


Figure 5.12: Interactive Dijkstra algorithm: Enter starting node

Selecting INTERACTIVE DIJKSTRA first asks you for INITIATION AND STARTING NODE (Figure 5.12 - you are prompted to enter the node from which shortest paths emanate), and then runs you through a series of four (4) menus, for which the >>items are (in cyclical sequence):

FIND: TEMPORARY LABELS

FIND: BEST TEMPORARY LABEL

FIND: NEW PERMANENT LABEL, and

FIND: IF END OF ALGORITHM

After the starting node is specified, the display shows the first node to close (in Figure 5.13, node 1). Then the user proceeds to FIND: TEMPORARY LABELS.

Figures 5.14 to 5.17 show this cyclical sequence of commands at an intermediate iteration of Dijkstra. In Figure 5.14, nodes 1, 2, 3, and 4 are already closed, and open nodes 5, 6, and 7 are candidate nodes to be closed next. The table in the figure shows the path lengths and predecessors from some closed node to each of these candidate nodes to be closed. The user then proceeds with FIND: BEST TEMPORARY LABEL, and the result is shown in Figure 5.15.

Figure 5.15 assigns temporary labels to each candidate node to be closed next. The user then proceeds with FIND: NEW PERMANENT LABEL, which identifies the node to be closed next. This is the node whose path length label entry among candidate nodes is the smallest (here node 7), and the result is shown in Figure 5.16, which presents an updated list of permanent labels.

The user then proceeds with FIND: IF END OF ALGORITHM, that is, to find if Dijkstra has terminated at this iteration. For this particular example, and as Figure 5.17 shows, the algorithm has not terminated, and this cyclical

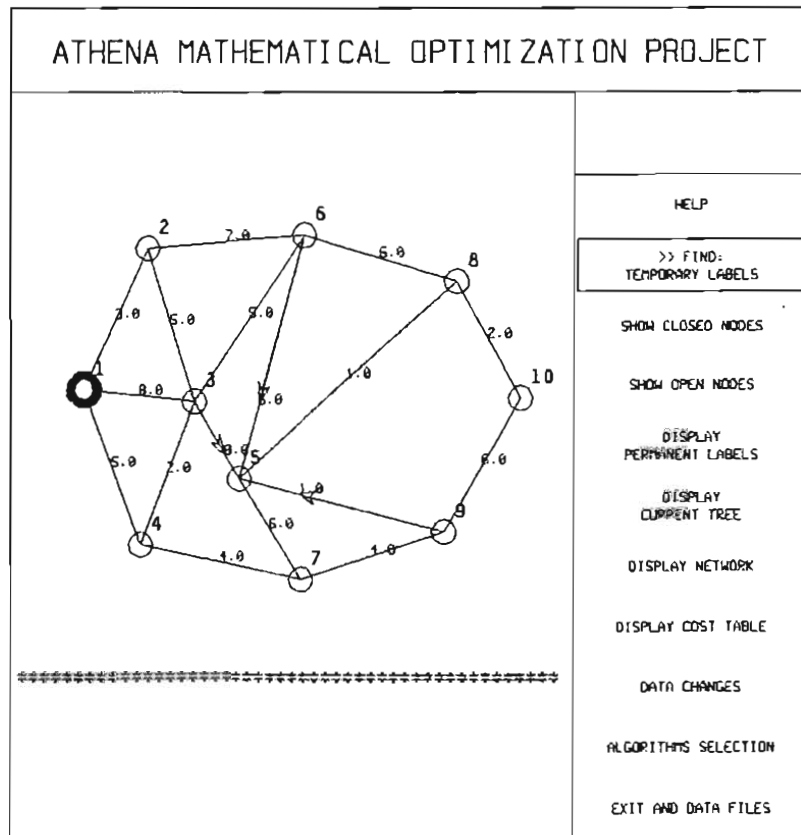


Figure 5.13: Dijkstra: first node to close, proceed to find temporary labels

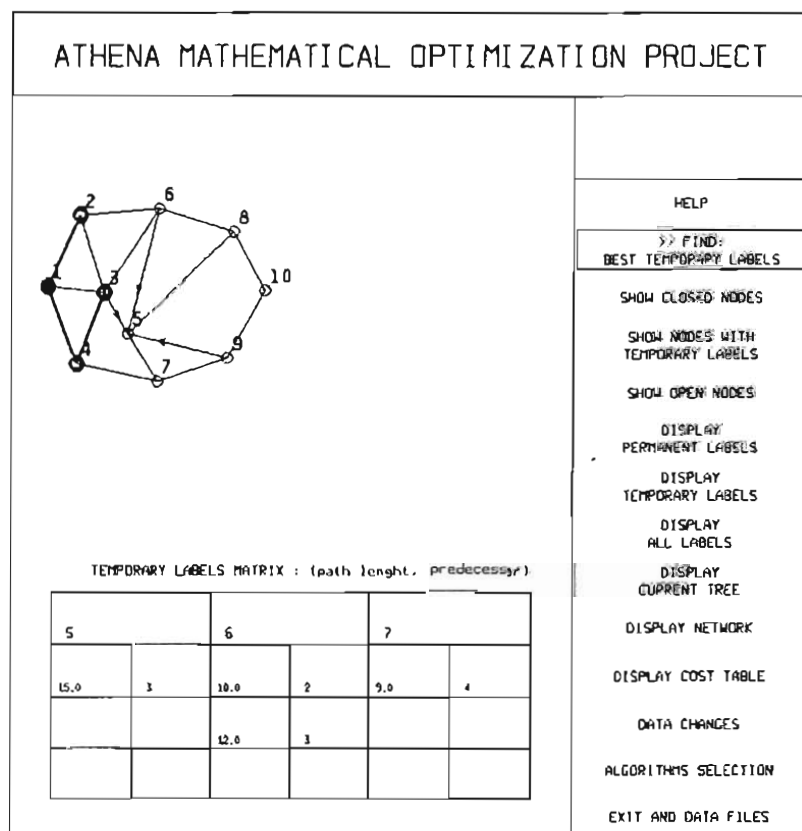


Figure 5.14: Dijkstra: Temporary labels auxiliary information

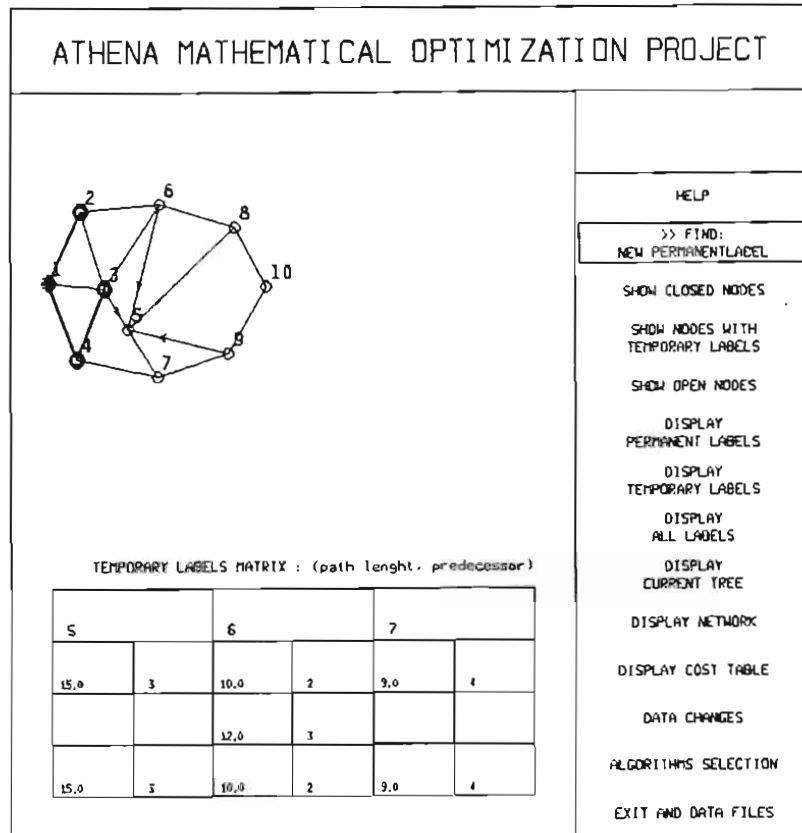


Figure 5.15: Dijkstra: Best temporary labels, proceed to final new permanent label (next node to close)

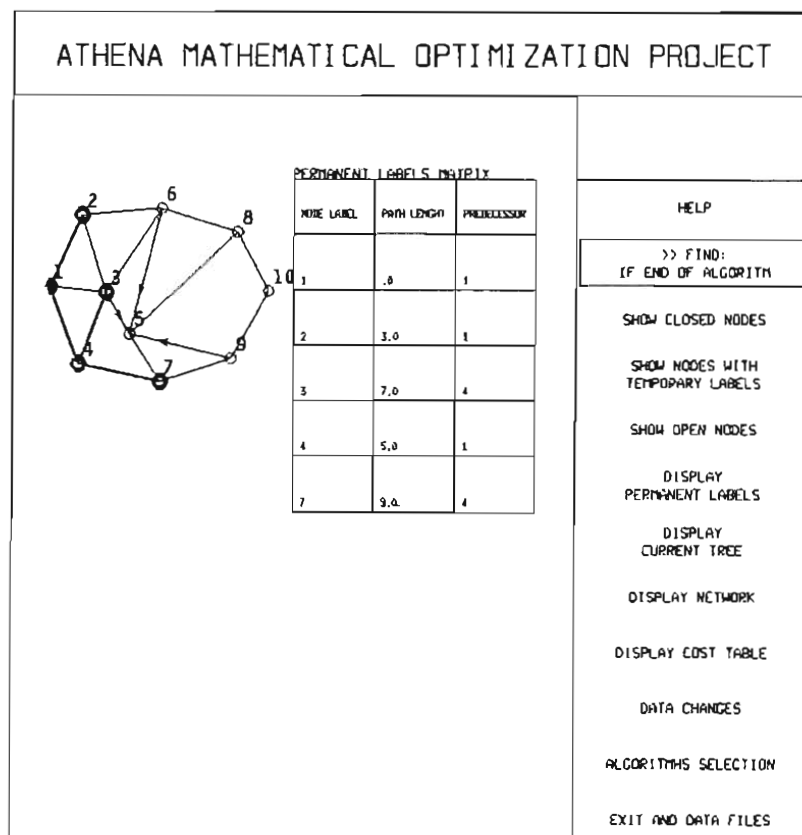


Figure 5.16: Dijkstra: Node 7 closes next. Proceed to check if there are more open nodes.

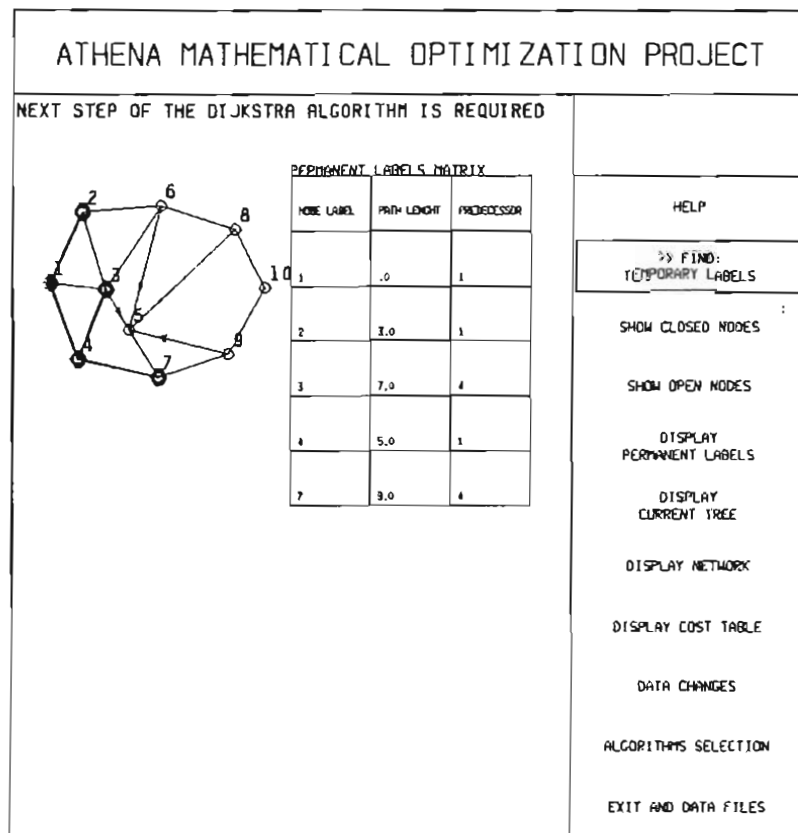


Figure 5.17: Dijkstra: Next iteration required

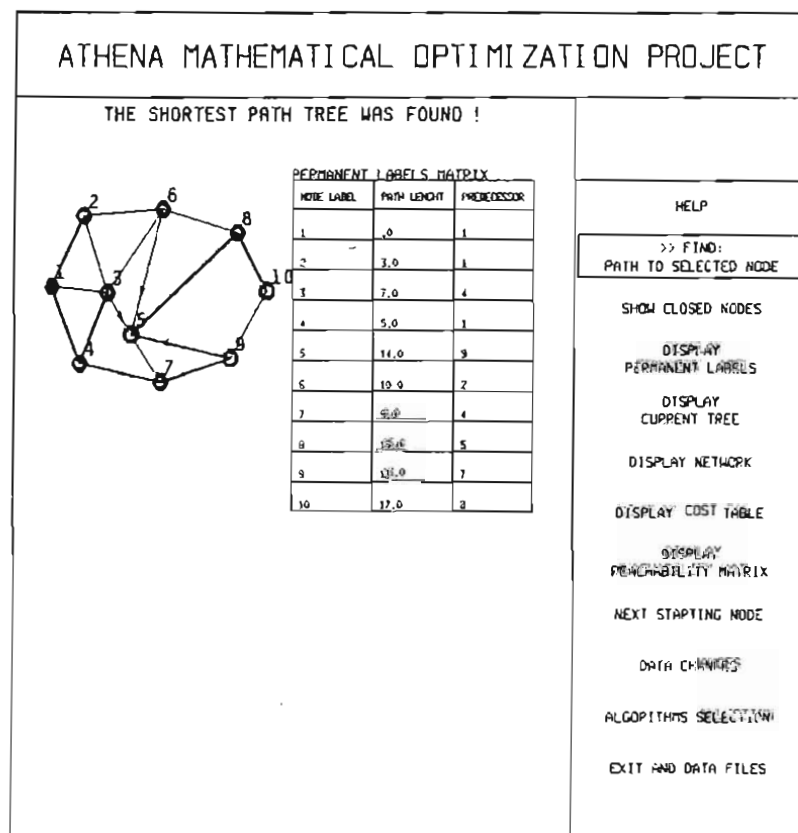


Figure 5.18: Dijkstra: End of algorithm

sequence is repeated.

Once Dijkstra terminates (see Figure 5.18), you are presented with a menu to obtain additional information about the solution.

Throughout this run of Dijkstra's algorithm, there are some auxiliary menu items that provide additional information (SHOW CLOSED NODES, SHOW OPEN NODES, DISPLAY PERMANENT LABELS, DISPLAY CURRENT TREE, etc). After you are done, selecting FIND: PATH TO SELECTED NODE prompts you to specify the selected ending node (node 9 in Figure 5.19).

NONINTERACTIVE DIJKSTRA just asks you for INITIATION: STARTING NODE, and then closes one node at a time by invoking DO THE NEXT STEP. At each step, the partial tree of shortest paths is displayed, and closed nodes are highlighted (see Figure 5.20). Only at the very end of this procedure can you get additional information about the solution.

Note: Dijkstra's algorithm is good for undirected, directed, and mixed graphs, but assumes that all edges have nonnegative costs. However, the program allows you to enter a negative cost link, and see what happens.

The FLOYD option presents you with a menu like the one in Figure 5.21, which has DO THE NEXT STEP as its option. If you select this menu item until the end, you will find all shortest paths from every node to every other node. Auxiliary intermediate information includes the "distance" and "predecessor" matrices. If you are at iteration k , invoking DISTANCE MATRIX displays the status of this matrix after the k^{th} node (in the order by which they were entered) is included. Here "distance" between two nodes is the length of the shortest path between these two nodes, possibly using the 1st, 2nd, 3rd, or k^{th} (entered) nodes as intermediate nodes. The same is true for the PREDECESSOR MATRIX. Invoking A PREVIOUS ONE in both cases shows the matrix in the previous

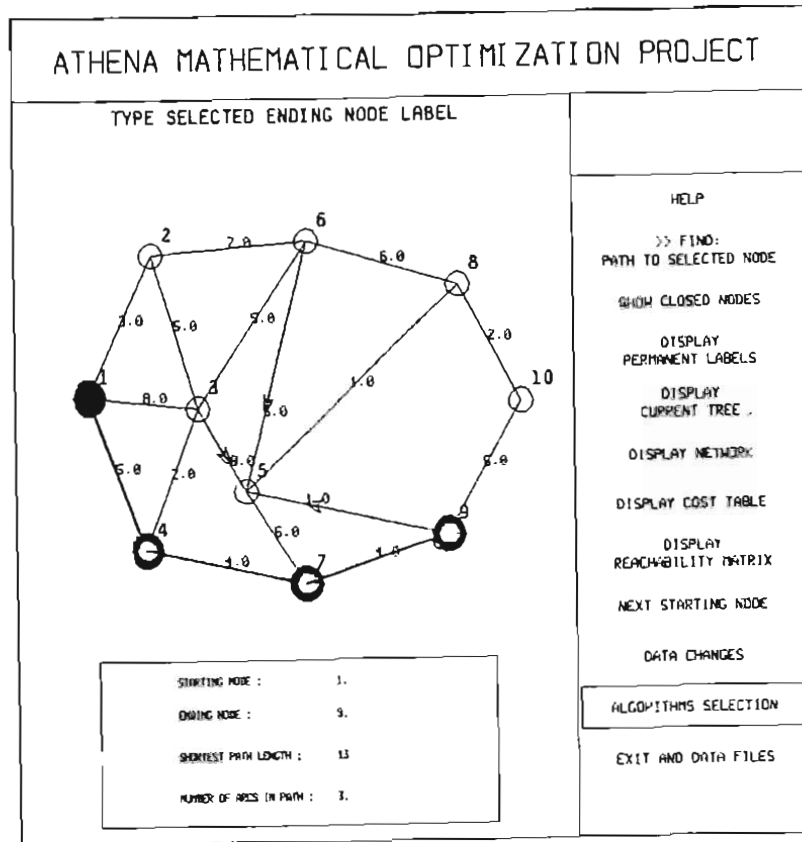


Figure 5.19: Dijkstra: Find path to selected node (here node 9)

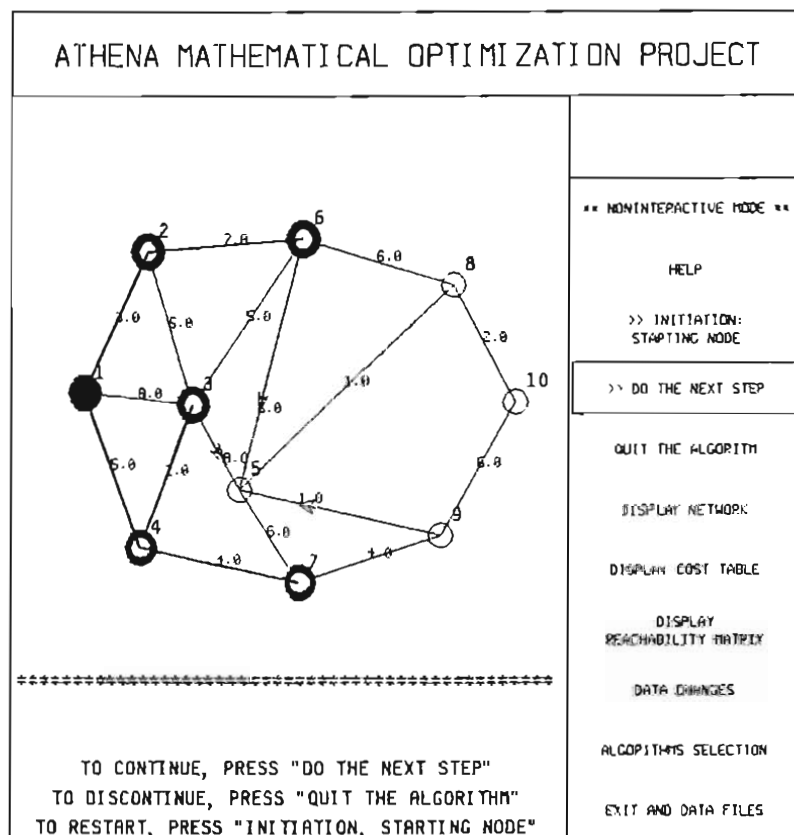


Figure 5.20: Noninteractive Dijkstra: Partial tree of shortest paths (from node 1) and closed nodes

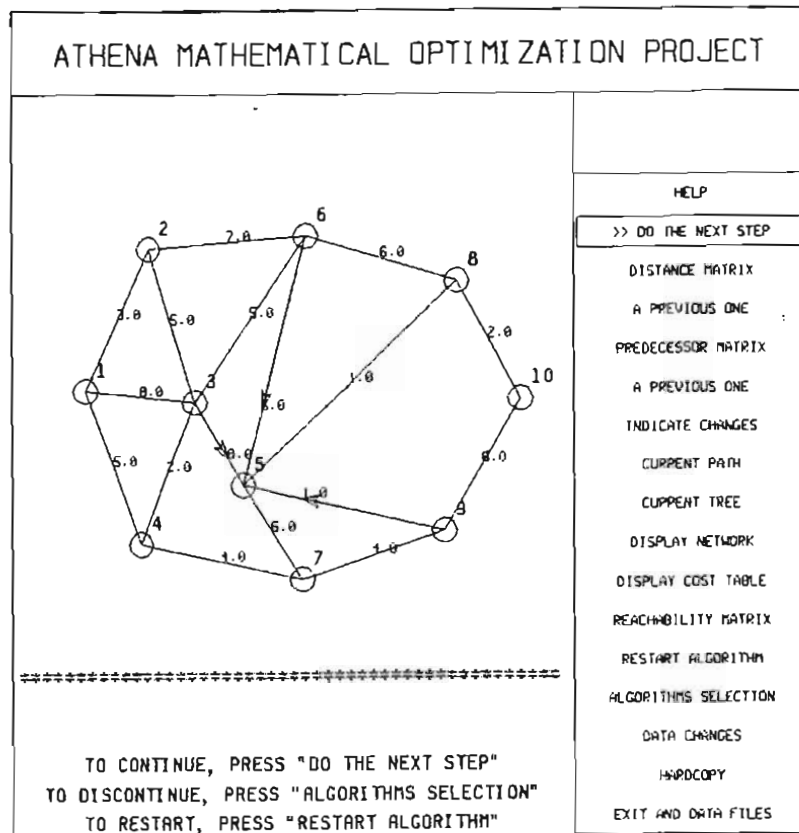


Figure 5.21: Floyd's algorithm: Menu selection

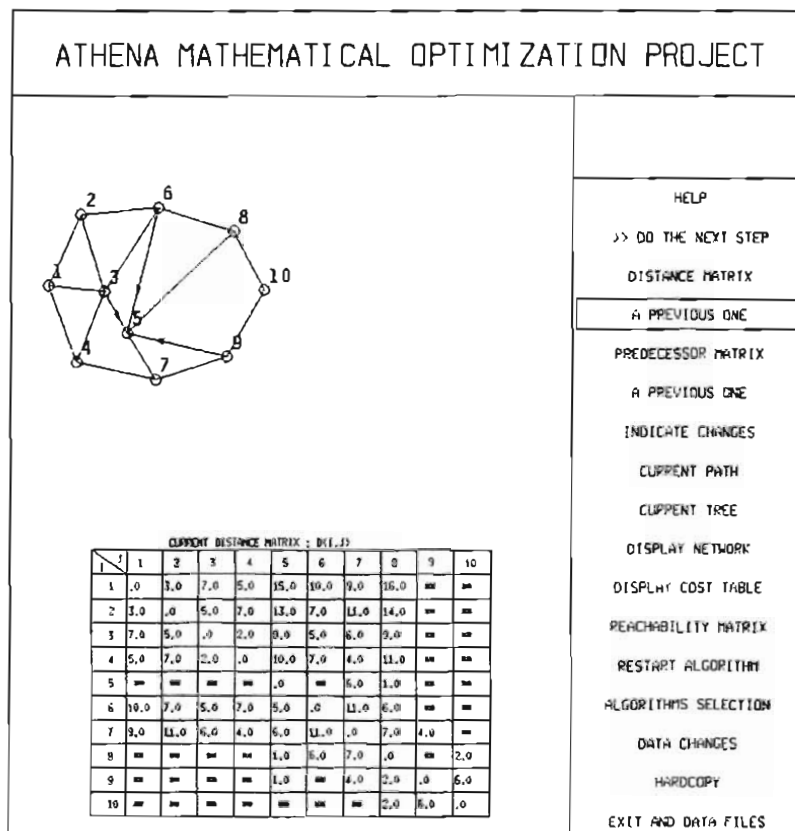


Figure 5.22: Floyd: Current distance matrix

iteration, and INDICATE CHANGES shows you those matrix entries that have been modified from the previous iteration to the current one (see Figures 5.22, 5.23, and 5.24 for samples of these matrices).

After Floyd is terminated (see Figure 5.25), you get a menu that allows you to obtain additional information. SELECTED TREE displays the tree of shortest paths emanating from a specified node (node 5 in Figure 5.26). SELECTED PATH displays the shortest path between two specified nodes (nodes 2 and 10 in Figure 5.27).

Note: Floyd's algorithm can tolerate negative edge costs, but assumes there are no negative cost cycles. However, these cycles are detected by the program, if they exist, by a negative entry in one of the "distance" matrices.

Reference

Larson and Odoni (1980), Chapter 6.

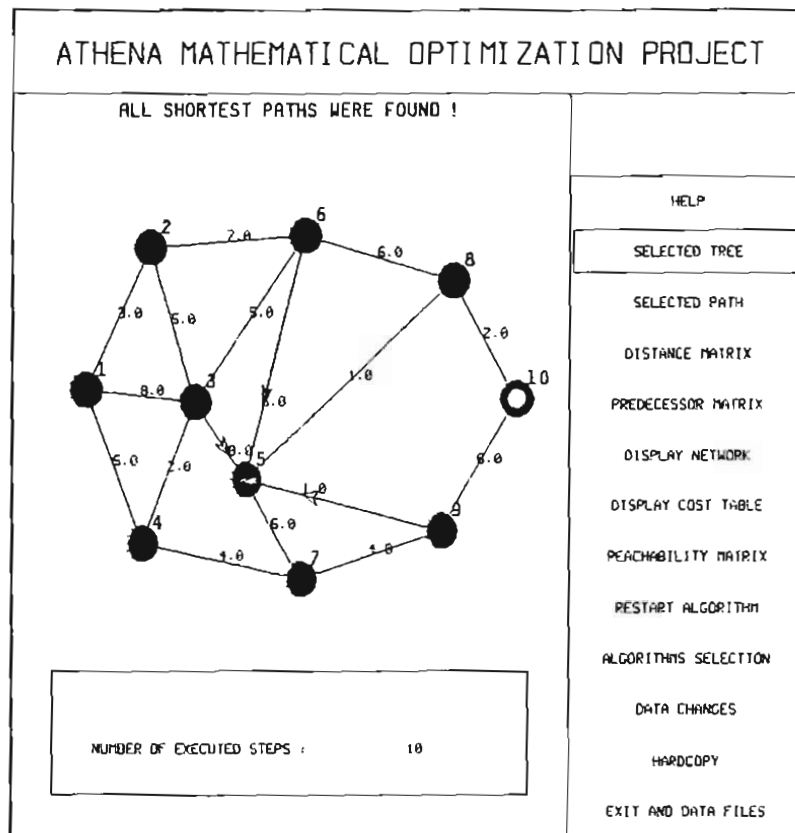


Figure 5.25: Floyd: End of algorithm

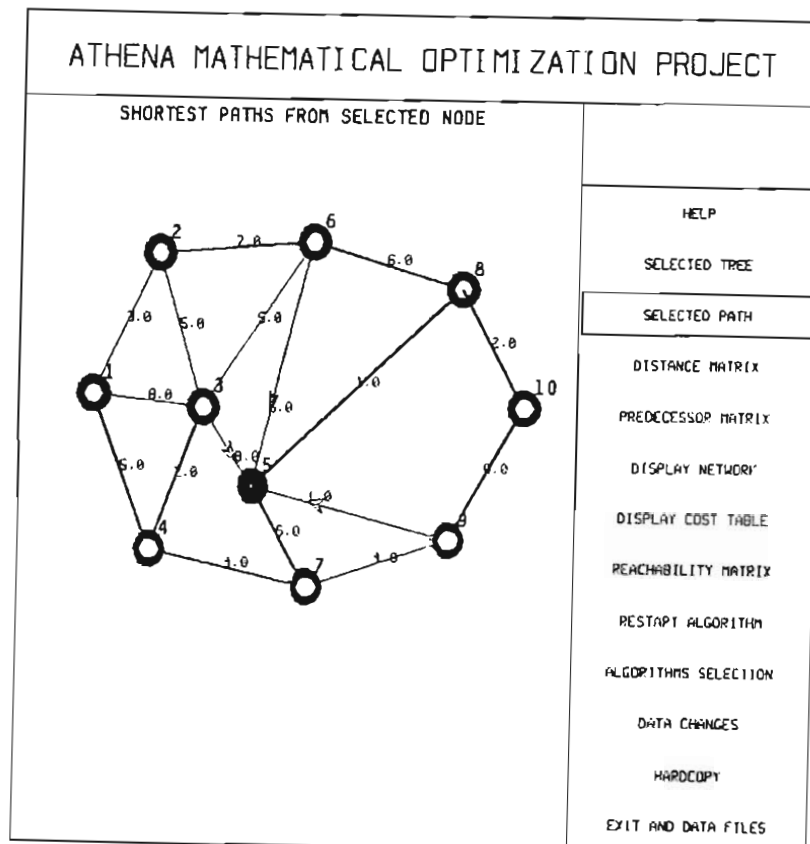


Figure 5.26: Floyd: Tree of shortest paths from selected node (here node 5)

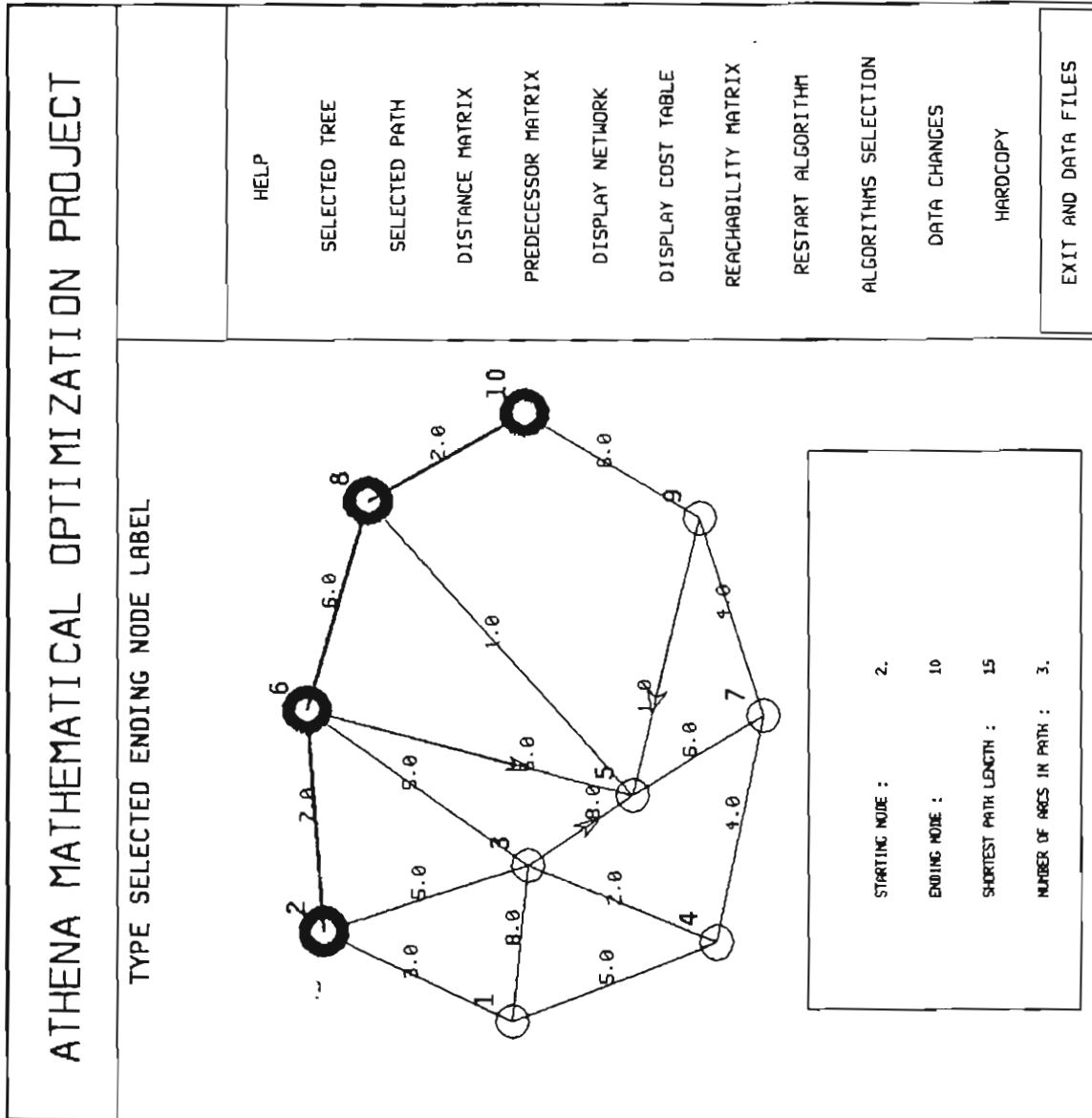


Figure 5.27: Floyd: Shortest paths between specified nodes (here nodes 2 & 10)

CHAPTER 6

THE MINIMUM COST FLOW PROBLEM

Developer: Jihong Ou

Introduction:

The purpose of package mcost is to allow the user to interactively solve the minimum cost flow problem on a specified network. As of 9/88, the major algorithmic option is Simplex (Phases I-II). All steps of the procedure are shown on the screen.

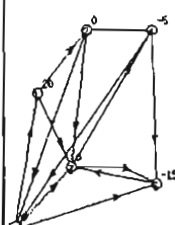
Summary of User Options (as of 9/88)

- 1) Enter and edit specified network (or read from a file).
- 2) Solve problem using Simplex method.
- 3) Do Phases I and II of Simplex.
- 4) Observe flows and multipliers and reduced costs.
- 5) Store network in a file.

Instructions

This section is to be completed soon. Package is operational, but still under development. See sample of screens for a flavor.

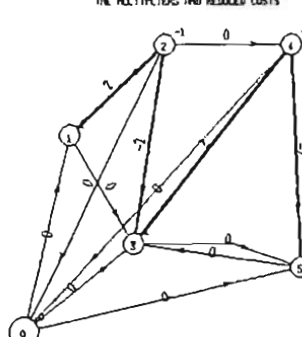
ATHENA MATHEMATICAL OPTIMIZATION PROJECT



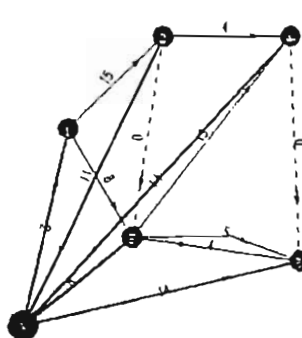
We are in Phase I.

| ARC | UPPER BOUND | LOWER BOUND | COST |
|-----|-------------|-------------|------|
| 0-1 | 100 | 0 | 1 |
| 0-2 | 100 | 0 | 1 |
| 0-3 | 100 | 0 | 1 |
| 1-2 | 15 | 0 | 0 |
| 1-3 | 0 | 0 | 1 |
| 2-3 | 100 | 0 | 1 |
| 2-4 | 100 | 0 | 1 |
| 2-5 | 100 | 0 | 1 |
| 3-4 | 15 | 0 | 0 |
| 3-5 | 5 | 0 | 0 |
| 4-5 | 100 | 0 | 0 |
| 5-1 | 0 | 0 | 0 |

THE MULTIPLIERS AND REDUCED COSTS



THE FLOW



PRESS IN THE WORK AREA TO FIND AN ENTERING ARC

*****MENU SELECTION*****

MOVE A NODE

PUT ARCS INTO THE TREE

DROP ARCS FROM THE TREE

ASSIGN ARCS FLOWS

PICK AN ENTERING ARC

DO PIVOTS ON THE TREE

THE FLOW GRAPH ONLY

THE MUL&R COST ONLY

ALL GRAPHS

RETURN

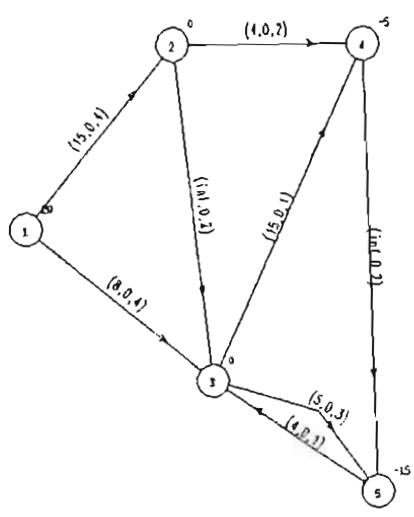
HELP

HARDCOPY

TO INTRODUCTION

TERMINATE PROGRAM

ATHENA MATHEMATICAL OPTIMIZATION PROJECT



FILE READING COMPLETED

*****MENU SELECTION*****

MODE: SMALL MED BIG

SIZE: DRAW **MOVE** DEL

ARC: DRAW DEL

INPUT THE NODE DATA

INPUT THE ARC DATA

DELETE THE NETWORK

PUT THE NETWORK IN FILE

DONE

HELP

HARDCOPY

TO INTRODUCTION

TERMINATE PROGRAM

CHAPTER 7
THE MAXIMUM FLOW PROBLEM

Developer: Peter Vranas

Introduction

The purpose of package maxfl is to allow the user interactively solve the maximum flow problem on a given network. Algorithms implemented include the Ford and Fulkerson, preflow-push, and shortest augmenting path methods. A "manual" algorithm, allowing the user to define flows and examine cuts on the network, is also included. All steps of these procedures are displayed on the screen.

Summary of User Options

- 1) Define and edit a network on the screen, or read from a file.
- 2) Display network, flows, cuts, and residual network.
- 3) Augment (or reduce) flows.
- 4) Solve using Ford and Fulkerson, preflow-push, shortest augmenting paths, or manually.
- 5) Observe all steps of these algorithms on the screen.
- 6) Store the network in a file.

Instruction

This package uses a network editor quite similar, but not identical to the ones described for packages spm (Chapter 5) and mcost (Chapter 6). After

all nodes have been defined, the user should identify the source and sink nodes. To define the source node, recommend to choose the leftmost node on the screen (see Figure 7.1). Select DEFINE SOURCE from menu and then click mouse cursor into source node. The node then becomes highlighted as in Figure 7.2. In Figure 7.2 an arc (here from node 5 to node 15) is defined. Enter the capacity of this arc (INF for infinite). Use the DEFINE SINK command and define the right-most node as the sink node.

Figure 7.3 shows a network (read from a file) and the algorithms menu of the package. There are four solution options, one MANUAL, and three algorithmic. The latter are FORD-FULKERSON, SHORTEST AUGMENTING PATH, and PREFLOW-PUSH. Each of the algorithmic options can either be INTERACTIVE or NONINTERACTIVE. The INTERACTIVE mode (as in previous packages) is a maximum-information, maximum-user-participation mode, whereas the NONINTERACTIVE mode is the opposite. We next describe these options in some detail.

The purpose of the MANUAL option is to allow the user to "play" with flows in the network, that is, push, reverse, and augment flows along prescribed paths, observe cuts, display the residual network, and, in general, get his/her feet wet with the problem. Flow feasibility is always preserved.

Figure 7.4 shows a typical flow in the network. Solid thin lines represent nonzero flows below capacity. Dashed lines denote zero flows. Arcs at capacity (not shown in Figure 7.4) are drawn thicker. On each arc one can see its capacity (in parentheses), and the value of the flow (if nonzero). Select SHOW KEY from manual menu to familiarize yourself with this notation.

In Figure 7.4 the user has selected DEFINE FLOW IN A PATH. This means selecting a path from the source to the sink, and modifying the flow on that path. For this example, suppose we pick path 2-4-5-10. The result is shown in

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|--|----------------------|
| CHOOSE (LEFTMOST) NODE AND PRESS INSIDE | DATA ENTRY MENU |
| | DRAW NODES |
| | DELETE NODES |
| | DRAW LINKS |
| | DELETE LINKS |
| | DRAW ARCS |
| | DELETE ARCS |
| | MOVE NODES |
| | DEFINE SOURCE |
| | DEFINE SINK |
| | DISPLAY NETWORK |
| | DELETE NETWORK |
| | CLEAR WORK AREA |
| | CLEAR SCREEN |
| | *** HARDCOPY |
| *** GO TO MAIN MENU | |

Figure 7.1: Source node definition.

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|--|-----------------|
| | DATA ENTRY MENU |
| | DRAW NODES |
| | DELETE NODES |
| | DRAW LINKS |
| | DELETE LINKS |
| | DRAW ARCS |
| | DELETE ARCS |
| | MOVE NODES |
| | DEFINE SOURCE |
| | DEFINE SINK |
| | DISPLAY NETWORK |
| | DELETE NETWORK |
| | CLEAR WORK AREA |
| | CLEAR SCREEN |
| | *** HARDCOPY |
| *** GO TO MAIN MENU | |

TYPE CAPACITY AT THIS ARC ("INF" IF INFINITY)
50

Figure 7.2: Arc capacity definition

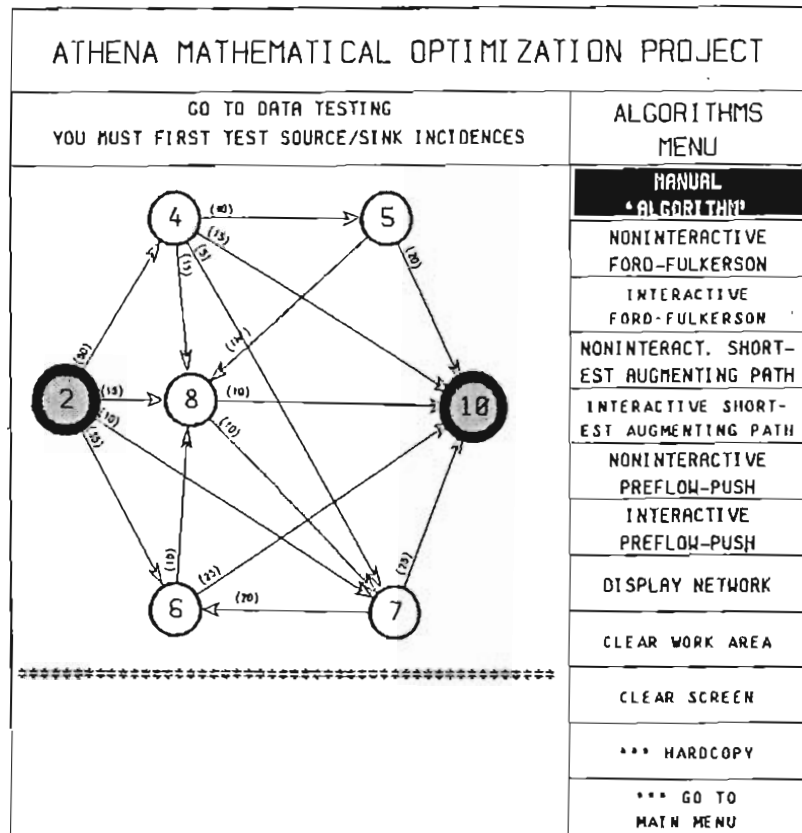


Figure 7.3: Algorithm menu

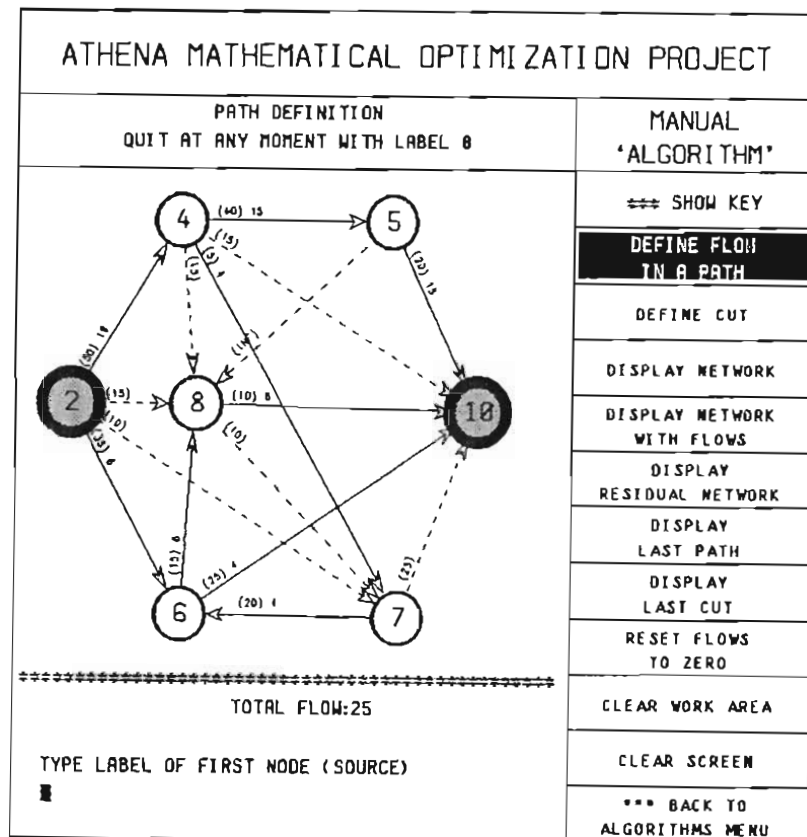


Figure 7.4: Augmenting path definition.

Figure 7.5.

In Figure 7.5, the selected path is highlighted in very thick lines, and a flow gauge is presented at the bottom of the screen, showing range of allowable flow augmentation along that path. In this example, flow on path 2-4-5-10 can decrease by as much as 15 units and increase by as much as 5 units. This range is dictated by the amount of current flow and by the capacities of arcs on that path.

If we move the mouse cursor within the bar-shaped gauge, we can dynamically (i.e. in real-time) update the flow on the path. The actual value of flow augmentation is represented by the black bar on the gauge (in Figure 7.5 it is equal to +2, corresponding to a new total flow of 27 units). Flows along all arcs of the path are also dynamically updated.

Once we decide how much new flow to send, we click the mouse, and that flow is "frozen". The result is shown in Figure 7.6. We can then proceed with a new augmenting path if we wish.

In Figure 7.7 we augment the flow on path 2-6-8-10 by -3 units. Using this mechanism, we can essentially do anything that is imaginable on the flow of the network. One can even attempt to find an optimal flow. Exercise: Can you improve upon the flow shown in Figure 7.8? (Hint: the augmenting path includes a "reverse" arc).

DISPLAY RESIDUAL NETWORK does just that. Figure 7.10 shows the residual network associated with the flow depicted in Figure 7.9.

The manual algorithm also allows for cut definition. To define a cut, select DEFINE CUT by specifying the set of nodes that belong to the same cut set as the source. Figure 7.11 shows a cut in which nodes 2 and 5 belong to one of the cut sets (and all other nodes to the other). Highlighted are the arcs that

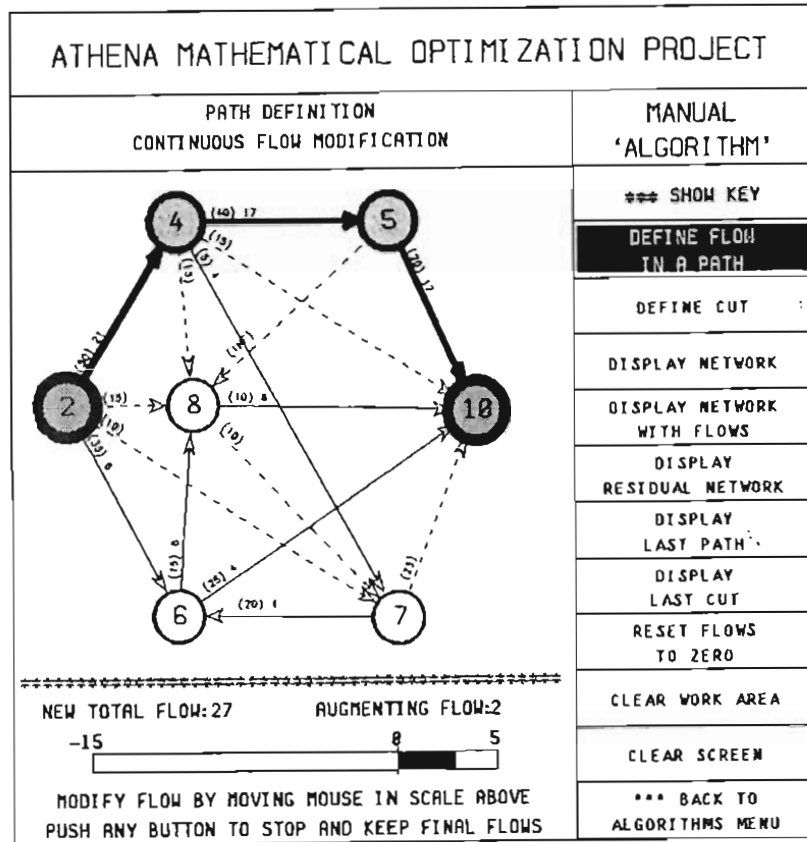


Figure 7.5: Augmenting the flow along path

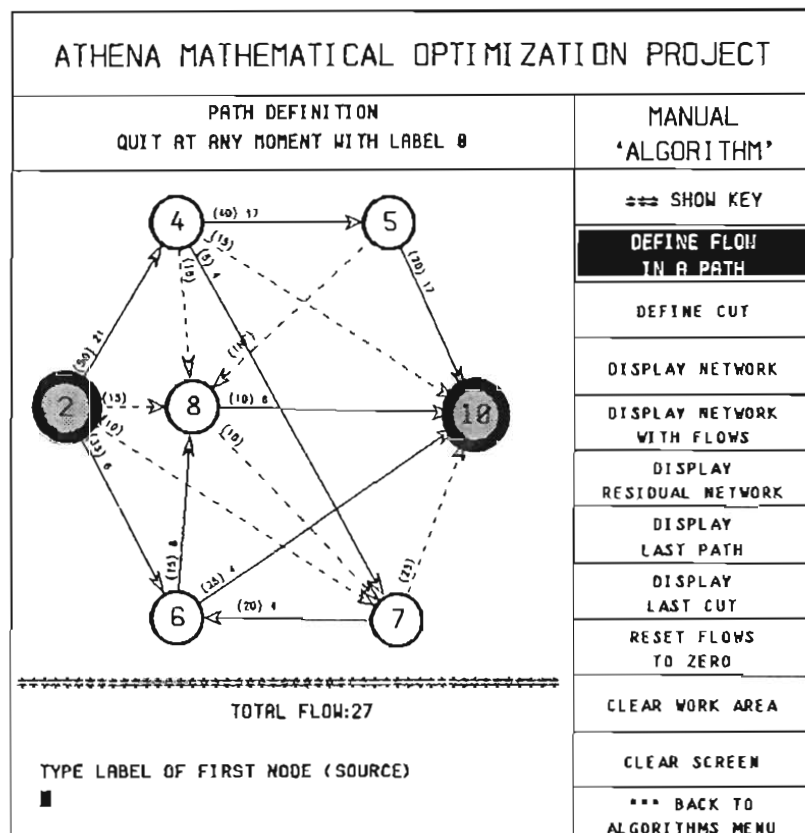


Figure 7.6: New flow

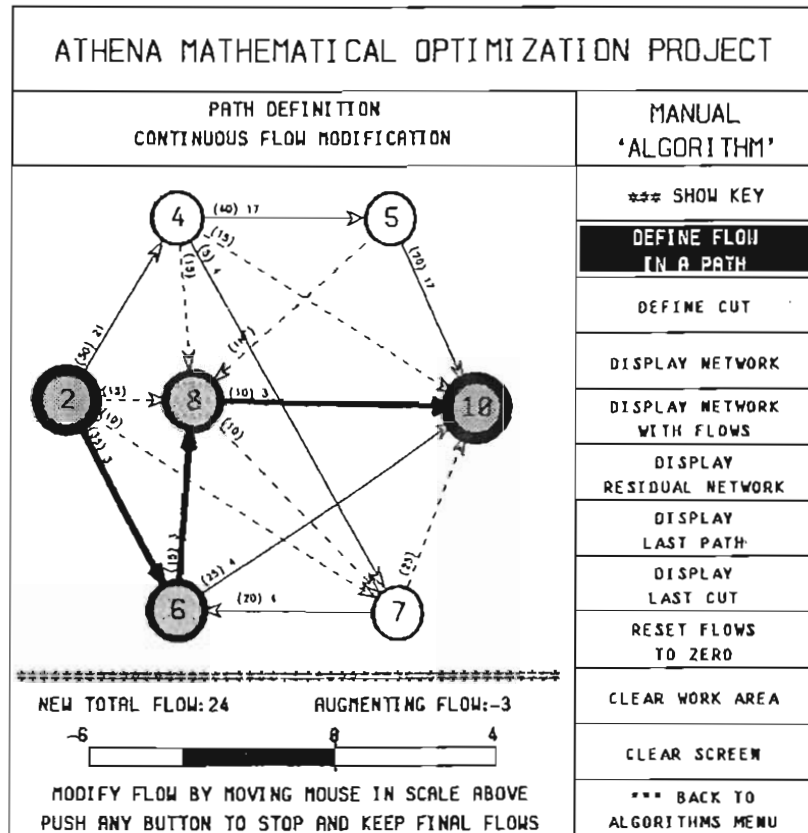


Figure 7.7: Another augmenting path

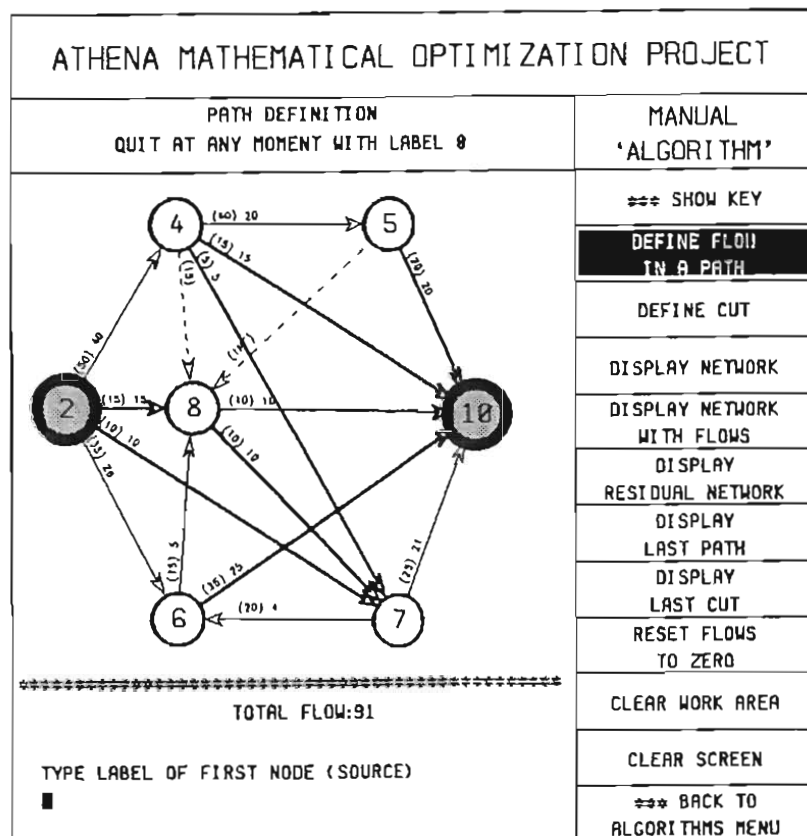


Figure 7.8: Exercise: Can you find a path that improves this flow?

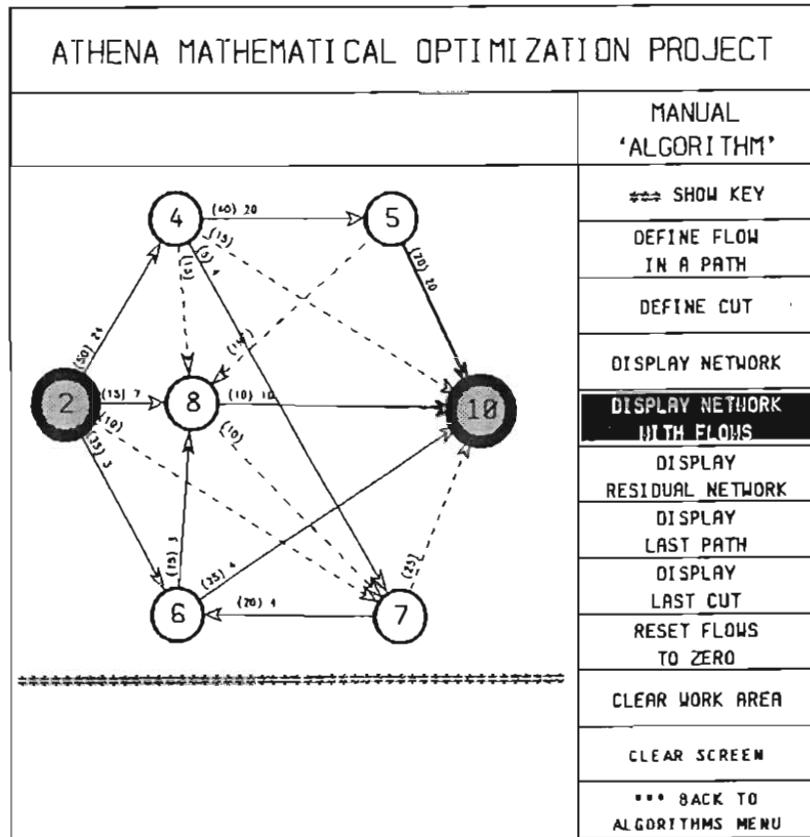


Figure 7.9: A flow, and ---

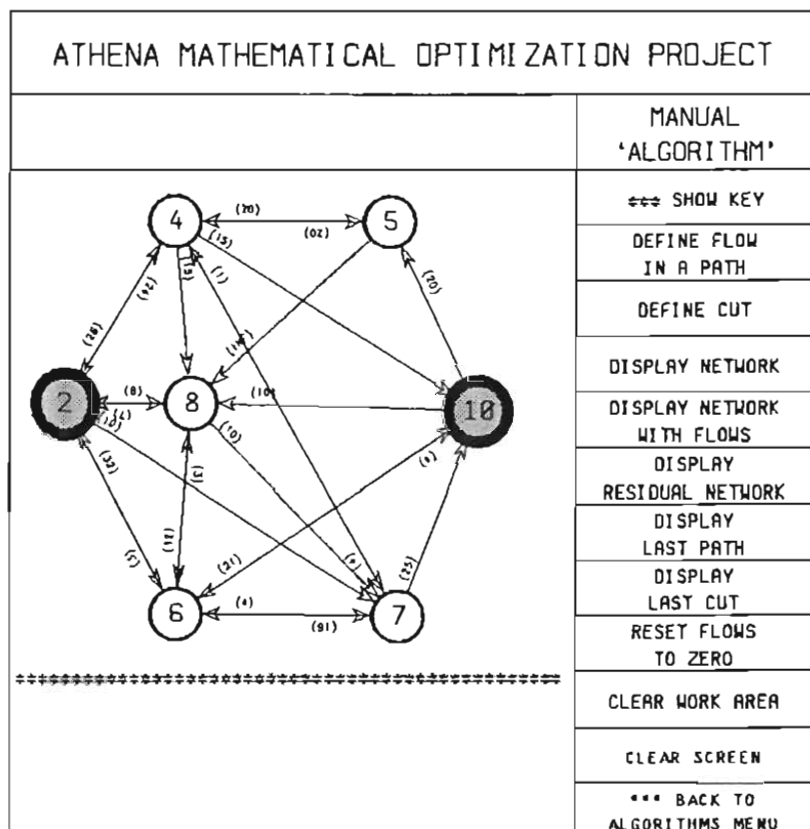


Figure 7.10: --- Its associated residual network

go from the former cut set to the other. The user is asked to calculate the capacity of this cut. The user is also asked to say whether this cut is an optimal (minimum) one. A definite answer to this question can be provided only if the cut capacity is equal to the total flow already established.

Invoking INTERACTIVE FORD-FULKERSON can be done even if initial flows are nonzero. Figure 7.12 shows the first display after INITIALIZE has been selected. Existing flow values can be kept, or reset to zero, at user discretion (in this case they were reset to zero).

The next step is SCAN. Scanning involves sequentially labeling nodes starting from the source, until the sink node is labeled. Selecting which node to label is done by the user. Scanned nodes are shaded. Figures 7.13 and 7.14 show the scanning of nodes 2 and 8. At this point, the source is labeled, and we proceed with AUGMENT FLOW. We proceed with SCAN and AUGMENT FLOW until the algorithm terminates at the optimal solution. At the end, the maximum flow and corresponding minimum cut are shown.

NONINTERACTIVE FORD-FULKERSON works similarly, but all the user has to do is select START ALGORITHM, and then DO THE NEXT STEP, until an optimal solution is found. Nodes are scanned and labeled automatically. It is interesting to note that an experienced user may obtain an optimal solution faster with the INTERACTIVE option than with the NONINTERACTIVE one!

The INTERACTIVE SHORTEST AUGMENTING PATH option has INITIALIZE, FIND MINIMUM POSITIVE THROUGHPUT, and PUSH/PULL as its main menu items. In Figure 7.15 (after flows were initialized to zero) the user is asked to provide the node with the least positive throughput (node 8 in this example). This node is shaded, and then the user proceeds to PUSH/PULL (see Figure 7.16). The result is shown in Figure 7.17. Going through this sequence of steps eventually

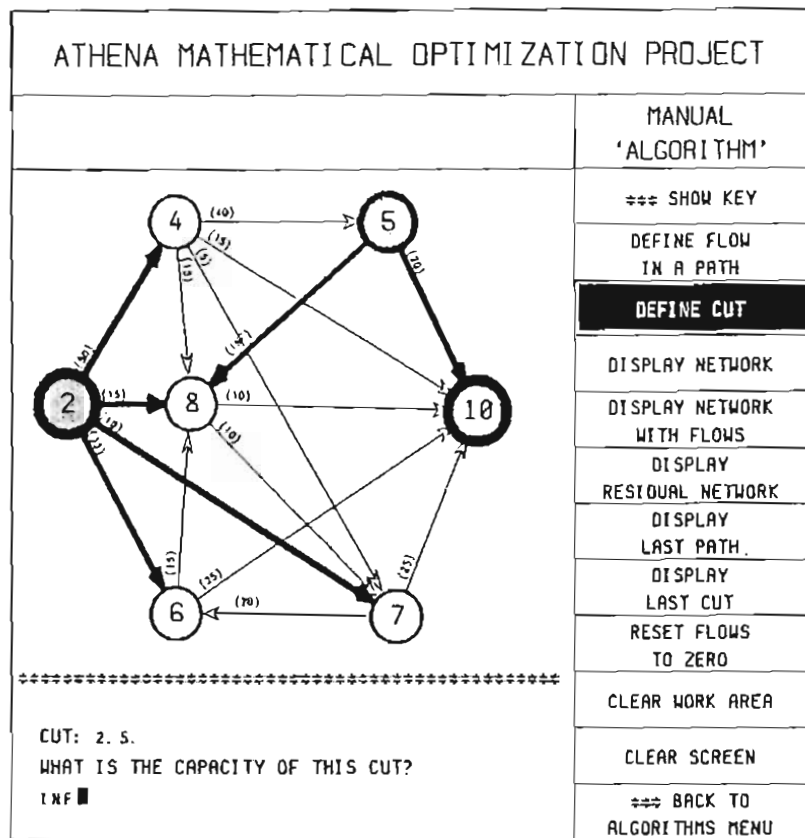


Figure 7.11: Nodes 2 and 5 define a cut

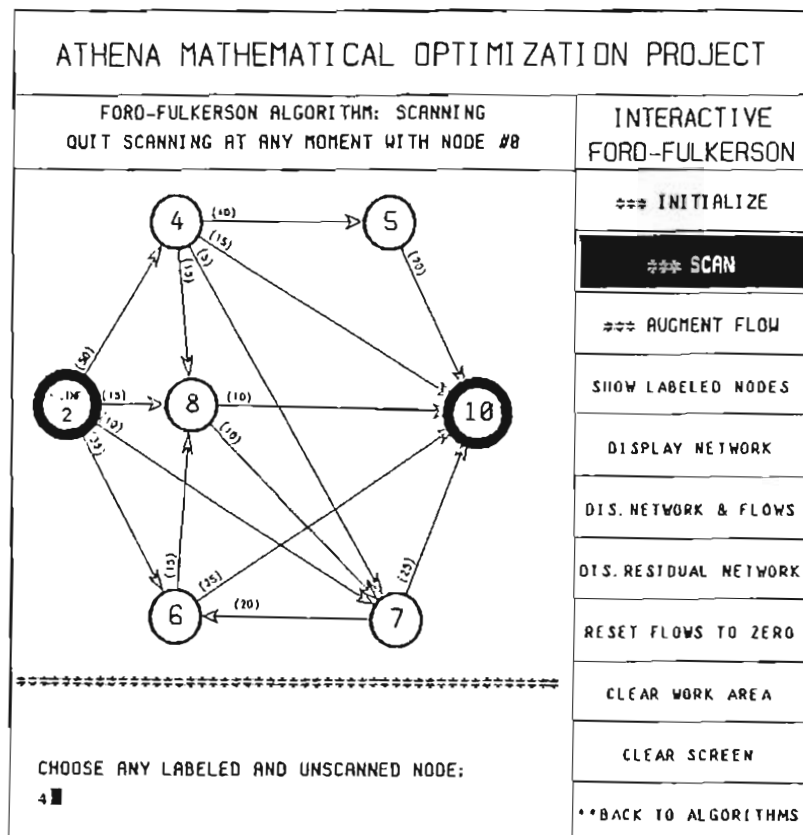


Figure 7.12: Ford-Fulkerson: Scanning (I)

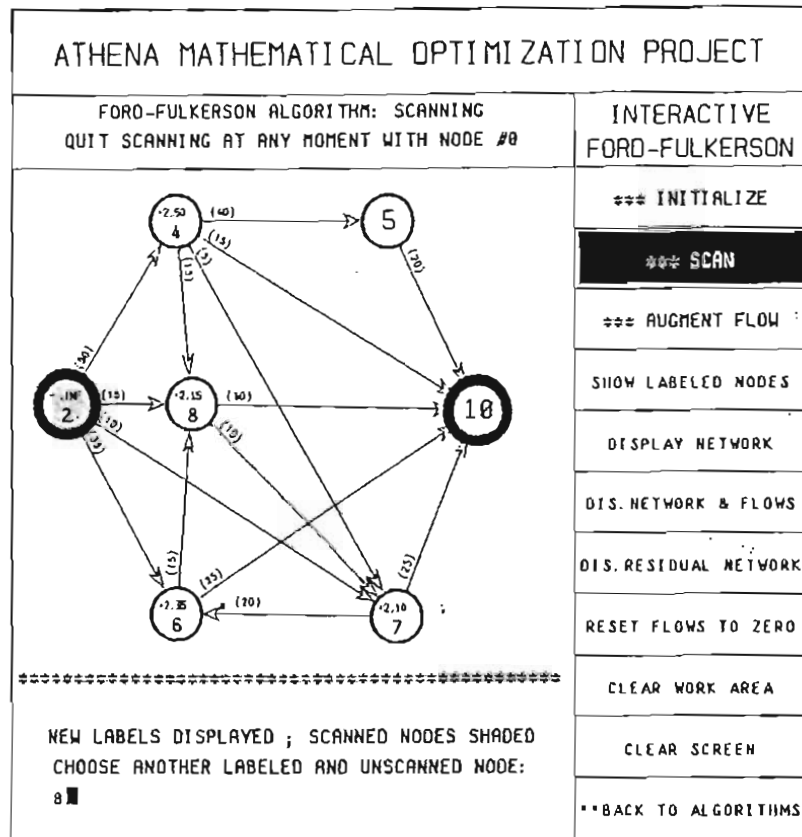


Figure 7.13: Ford-Fulkerson: Scanning (II)

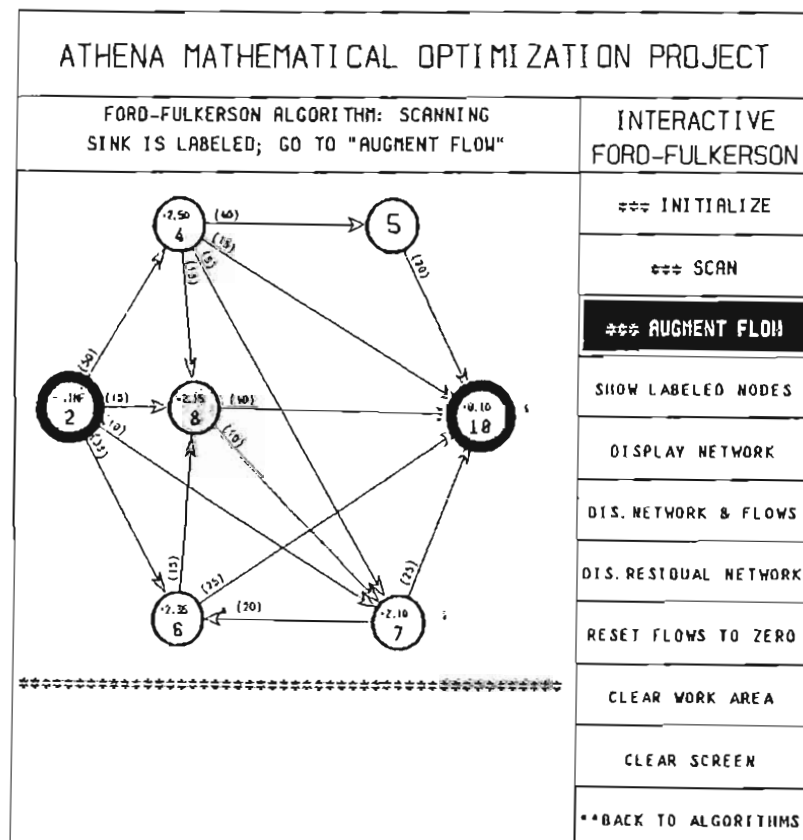


Figure 7.14: Ford-Fulkerson: flow augmentation

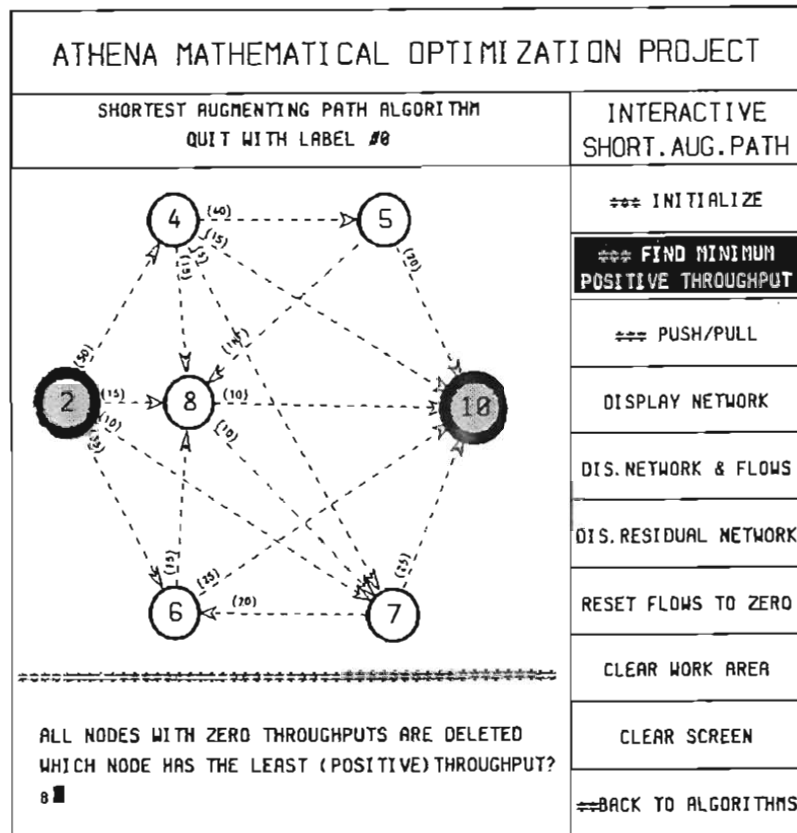


Figure 7.15: Shortest augmenting path: Finding minimum positive throughput

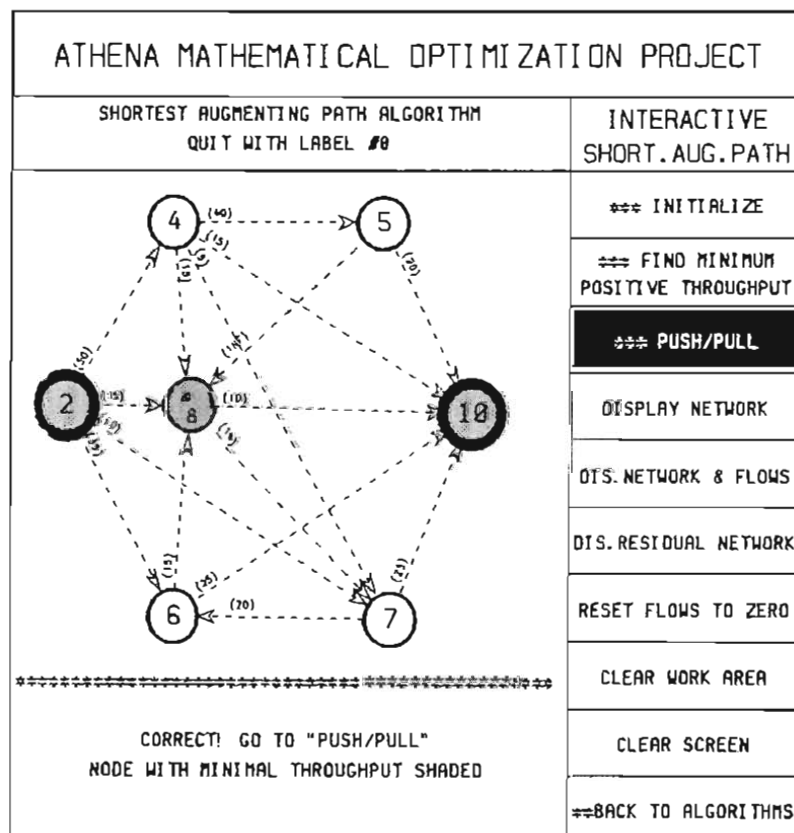


Figure 7.16: Shortest augmenting path: Push/Pull (I)

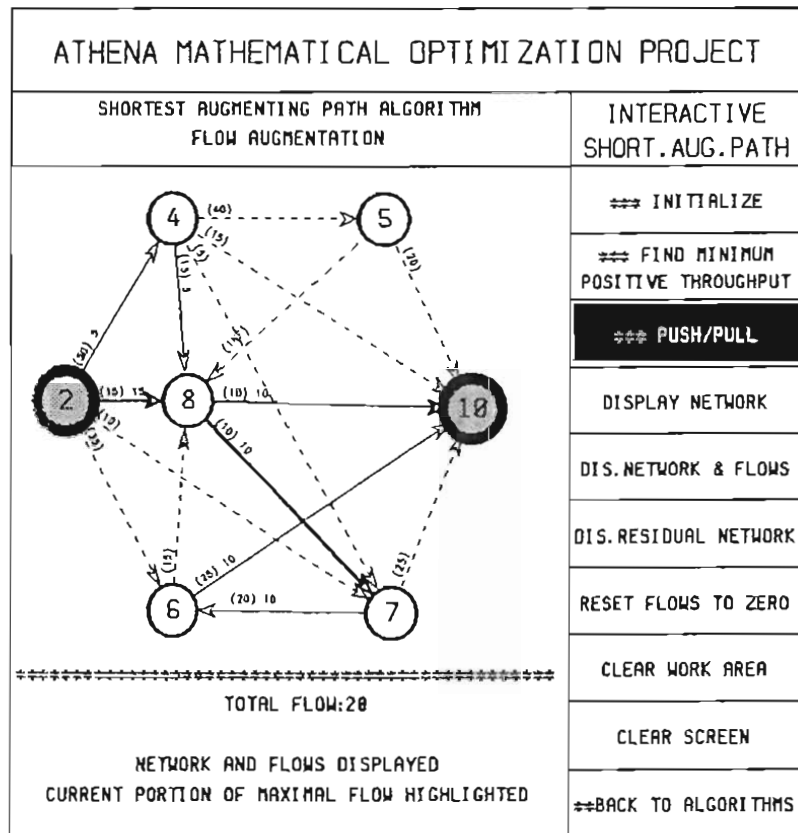


Figure 7.17: Shortest augmenting path: Push/Pull (II)

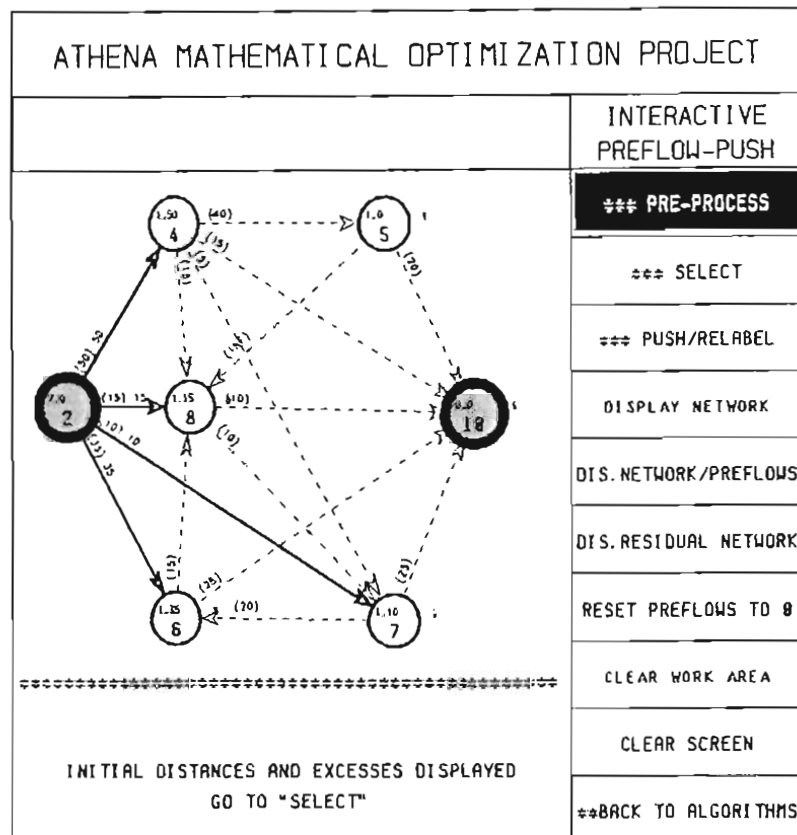


Figure 7.18: Preflow-Push: Preprocess

produces an optimal solution.

The INTERACTIVE PREFLOW-PUSH algorithm has PREPROCESS, SELECT, and PUSH/RELABEL as its main menu items. PREPROCESS displays initial distances and excesses (see Figure 7.18). Then you are asked for a node with positive excess (node 4 in Figure 7.19). The result is shown in Figure 7.20, and the procedure is applied until an optimal solution is found.

The NONINTERACTIVE versions of the last two algorithms have DO THE NEXT STEP as their main menu item.

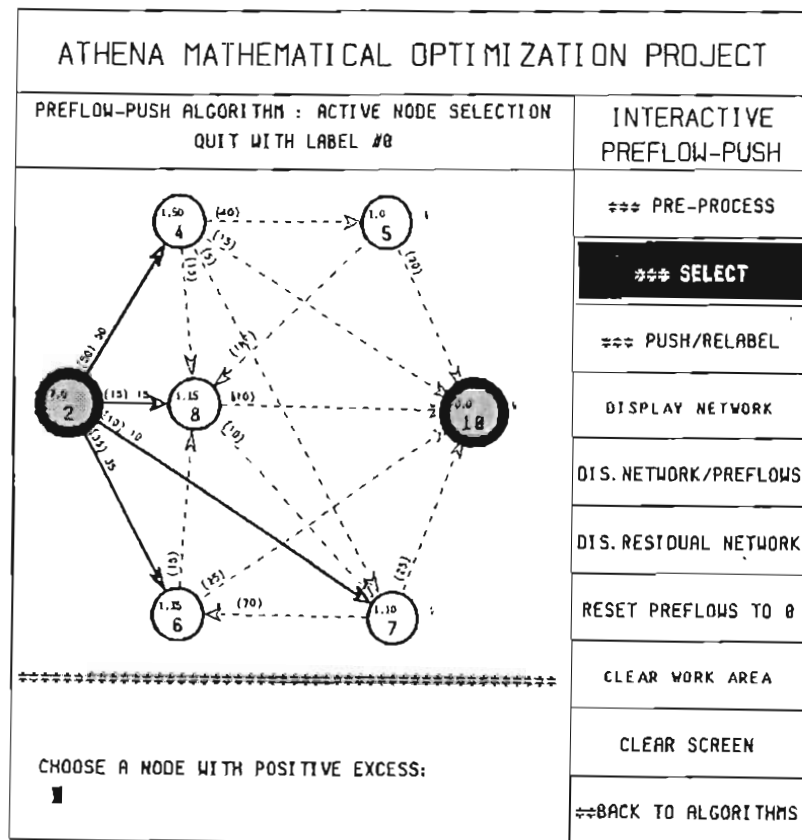


Figure 7.19: Preflow-Push: Select node with positive excess

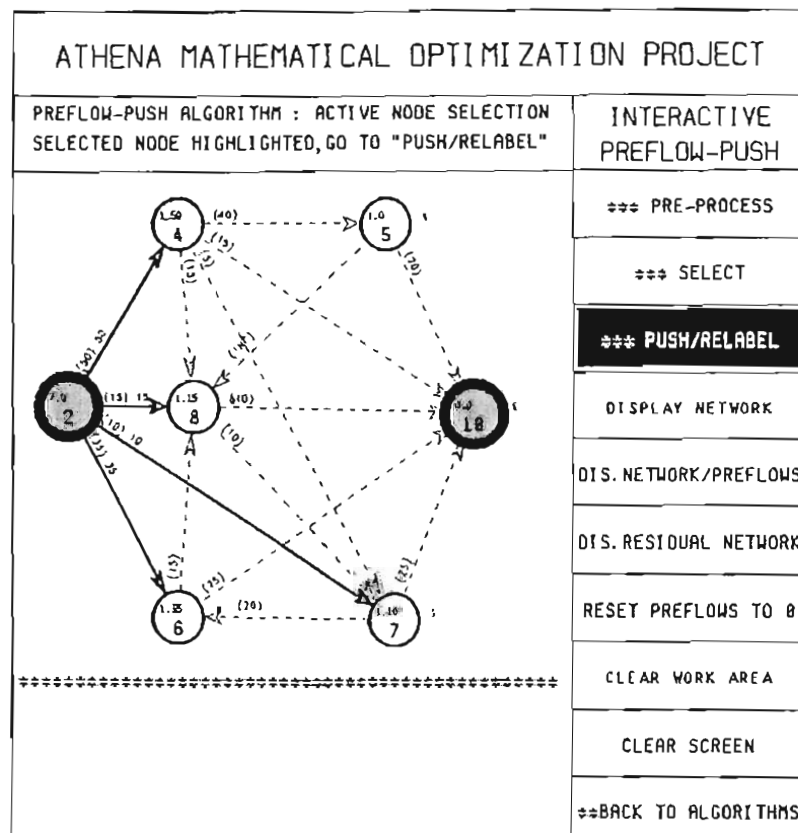


Figure 7.20: Preflow-Push: Push-relabel

CHAPTER 8
THE TRAVELING SALESMAN PROBLEM

Developer: Lambros Anagnostopoulos

Introduction

The purpose of package travel is to interactively solve a traveling salesman problem defined on a specified network. A network editor is used to define the network. Several heuristic (approximate) procedures are available to solve the problem. All steps of these procedures are shown on the screen.

Summary of User Options

- 1) Enter and edit a general network (or read from a file).
- 2) Enter a Euclidean network.
- 3) Form an initial T.S. tour either manually, or by the Nearest Neighbor, Minimum Spanning Tree, or Christofides' heuristics.
- 4) Improve on the initial tour by performing 2-interchanges or 3-interchanges.
- 5) Store the network in a file.

Instruction

The network editor for the TSP is based on the same principles with (but is not identical to) the editors for the network problems examined in chapters 5, 6, and 7. Thus, and as before, we have again DRAW NODES, DELETE NODES, DRAW LINKS, DELETE LINKS, DRAW ARCS, DELETE ARCS, etc.

The major distinct option for the TSP is EUCLIDEAN NETWORK. This option

allows you to define a network on the Euclidean plane.

Once you select this option, you are asked for the number of nodes. You are then presented with a square work area, where all nodes should be located. When prompted, enter the dimension of the square (default is 1 unit - see Figure 8.1).

Nodes are subsequently defined by clicking the mouse cursor into the square. All nodes are automatically numbered (sequentially). Once all nodes have been defined, you get a message "this is the euclidean network" (Fig. 8.2) and then you are ready to proceed with the solution. In the Euclidean case, link costs are assumed equal to the Euclidean distances between nodes, and the links themselves are not displayed in this initial screen.

Other network input and edit options are:

COMPLETE NETWORK: Use this option after you have defined all nodes, and if the network is complete. You will be asked to enter all link (arc) costs interactively.

INPUT NETWORK DATA: This option is not currently recommended. It is intended for (x, y) node coordinate input.

MAKE NETWORK SMALL and DISPLAY COST DATA: Use both commands to show the link cost (distance) matrix. Matrices for more than 10 nodes are displayed with difficulty due to screen resolution limitations.

CHANGE COST DATA: Select this to modify link (arc) costs (proceed as instructed).

DELETE NETWORK: To delete all data and start from scratch.

ERASE PICTURE: To erase the screen work area without affecting the data.

To solve the TSP problem, select SOLVE TSP PROBLEM and then first select one of the four options under the -- INITIAL TOUR -- menu.

| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|---|--|
| <p style="text-align: center;">Press in the work area</p> <div style="border: 1px solid black; height: 200px; margin: 10px auto; width: 80%;"></div> <p>Give number of the network nodes</p> <p>20</p> <p>Give the scale of square for default (1.0 X 1.0) press " 1 "</p> <p>1</p> | <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>HELP:</p> <p style="text-align: center;">on</p> <p style="text-align: center;">off</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p>Draw nodes</p> <p>Delete nodes</p> <p>Draw links</p> <p>Delete links</p> <p>Draw arcs</p> <p>Delete arcs</p> <p>Make network small</p> <p>Resize network</p> <p>Input network data</p> <p>Complete network</p> <p>Display cost data</p> <p>Change cost data</p> <p>Euclidean Network</p> <p>Solve T.S.P. problem</p> <p>Delete network</p> <p>Erase picture</p> <p>Store picture</p> <p>Recall picture</p> <p>Hardcopy</p> <p>Exit</p> </div> |

Figure 8.1: Euclidean network: work area before node input

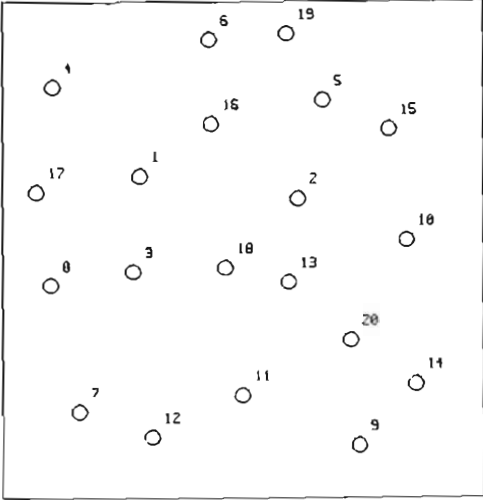
| ATHENA MATHEMATICAL OPTIMIZATION PROJECT | |
|--|--|
| <p style="text-align: center;">Press at center of the node</p> <div style="border: 1px solid black; padding: 10px; margin: 10px auto; width: 80%;">  </div> <p>Give number of the network nodes</p> <p>20</p> <p>This is the euclidean network</p> | <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p>HELP:</p> <p style="text-align: center;">on</p> <p style="text-align: center;">off</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p>Draw nodes</p> <p>Delete nodes</p> <p>Draw links</p> <p>Delete links</p> <p>Draw arcs</p> <p>Delete arcs</p> <p>Make network small</p> <p>Resize network</p> <p>Input network data</p> <p>Complete network</p> <p>Display cost data</p> <p>Change cost data</p> <p>Euclidean Network</p> <p>Solve T.S.P. problem</p> <p>Delete network</p> <p>Erase picture</p> <p>Store picture</p> <p>Recall picture</p> <p>Hardcopy</p> <p>Exit</p> </div> |

Figure 8.2: Euclidean network: work area after node input

If you select ENTER MANUALLY, you will be asked to provide a sequence of tour nodes. Once you do that, the initial tour is displayed on the screen.

If you select NEAREST NEIGHBOR, you will be asked to give the first node. The program then forms and displays the initial tour (see Figure 8.3 - Note that the cost (length) L of the tour is also displayed, and for Euclidean cases, also the value of $L/\sqrt{\text{Area} \cdot n}$).

If you select MINIMUM SPANNING TREE, you will be first presented with a display of the network's minimum spanning tree (Figure 8.4), and then you will be asked if you want to create a tour out of this tree. Typing "yes" forms a tour that goes around the tree (Figure 8.5).

The CHRISTOFIDES option is valid for symmetric networks for which the triangle inequality holds (these include, but are not limited to, Euclidean networks). Nevertheless, the user has the option of using this heuristic for other networks as well, and see what happens.

Selecting CHRISTOFIDES involves three steps: The first step is the formation of the minimum spanning tree of the network, and is automatically performed by the program (see Figure 8.6). The second step involves the matching of odd-degree nodes of the MST, and should be manually performed by the user. The third step uses shortcuts to make a tour out of the resulting network. Alas, this step should be manually performed by the user as well.

For the first step, odd-degree nodes of the minimum spanning tree are shown in black. The user is then asked whether each specific pair of odd-degree nodes should be included in the matching. In Figure 8.6, the user wants nodes 2 and 3 to be matched. The result appears in Figure 8.7, where a link is added between the matched nodes, and their color shifts to white (they are no longer odd-degree). The queries go on until all odd-degree nodes are matched.

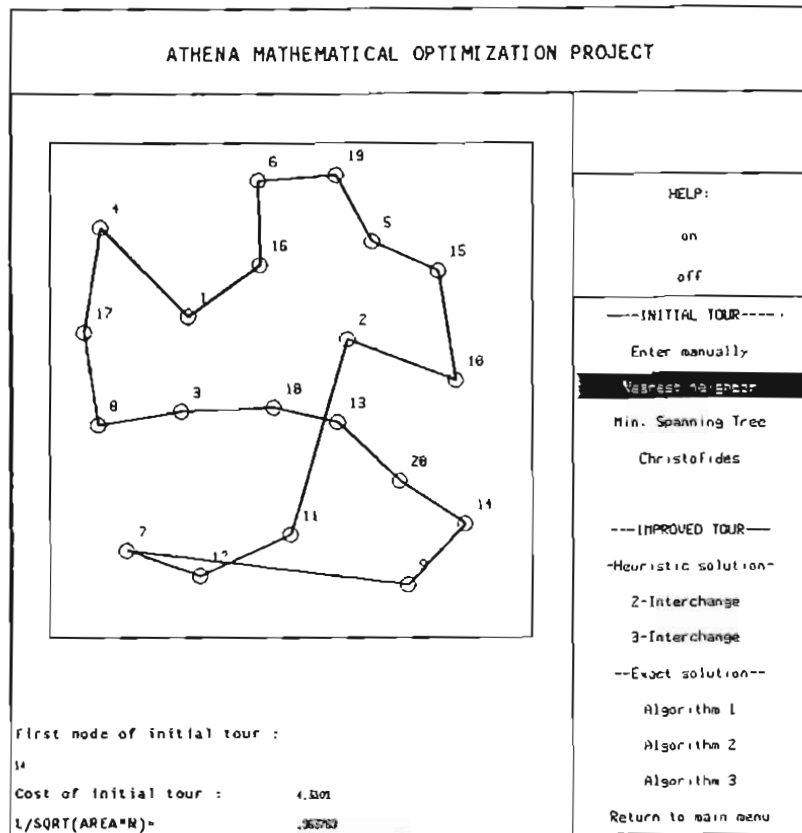


Figure 8.3: Nearest neighbor: initial tour starting from node 14

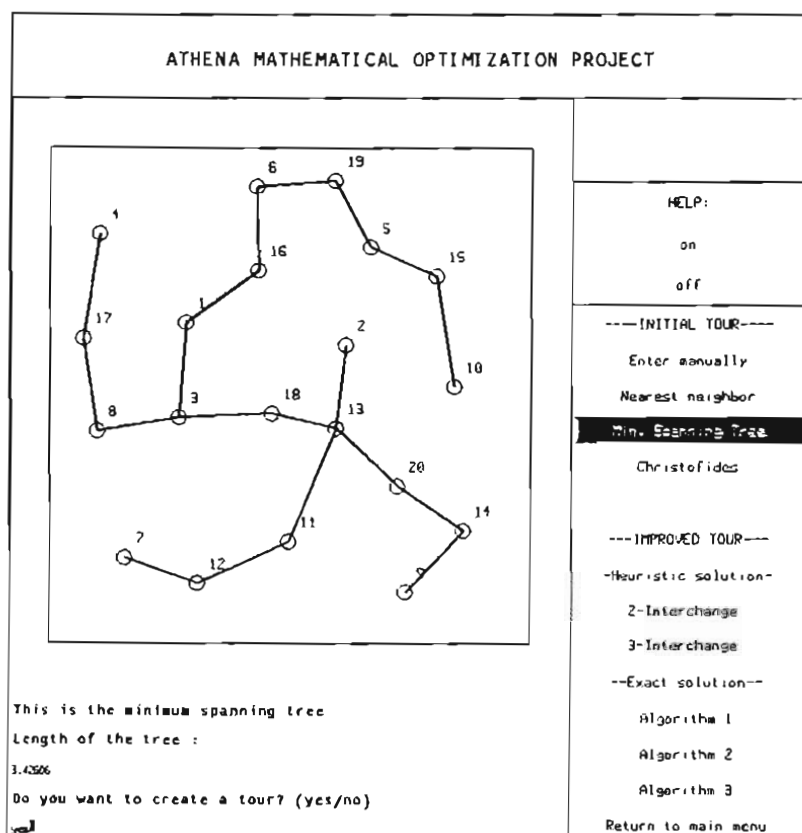


Figure 8.4: Minimum spanning tree of Euclidean network

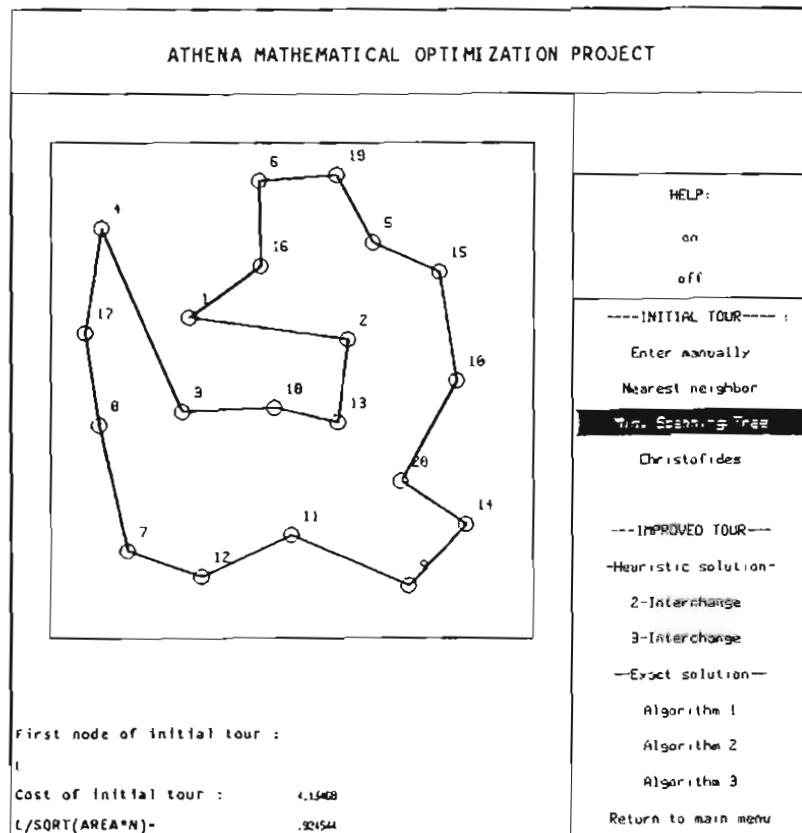


Figure 8.5: Tour produced from MST

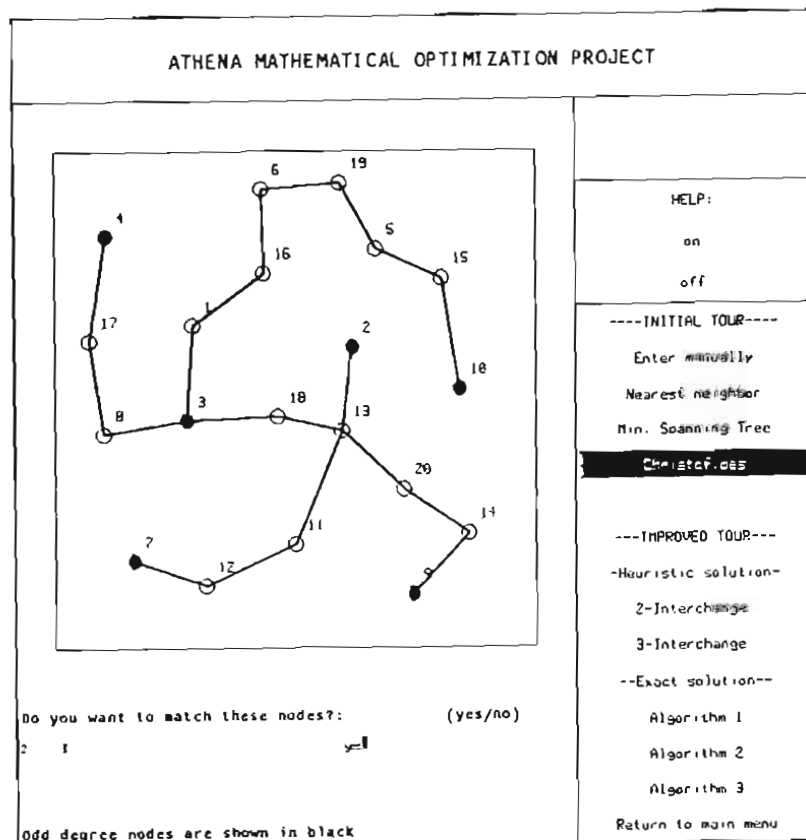


Figure 8.6: Christofides algorithm: Odd degrees of the MST are in black. User matches nodes 2 and 3.

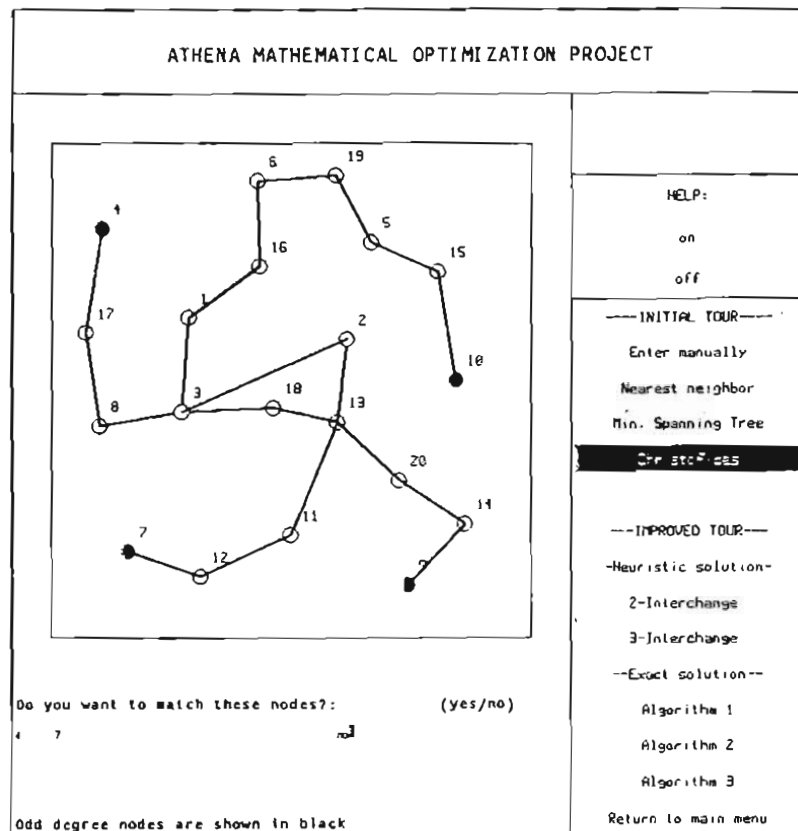


Figure 8.7: Christofides algorithm: Link (2,3) is added to the matching.

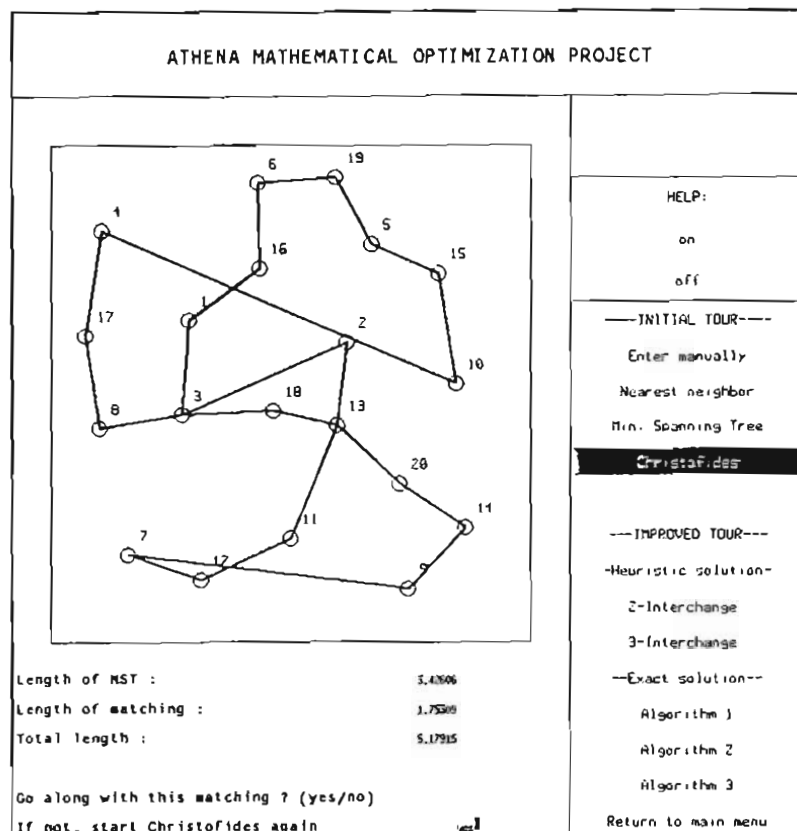


Figure 8.8: Christofides algorithm: Final result of MST & Matching

There are obviously many ways odd-degree nodes can be matched. Recommend to decide upon overall matching before you enter it. Obviously, your matching is not necessarily an optimal one. If you are not satisfied with a particular matching, start CHRISTOFIDES from scratch. If you are satisfied, reply "yes" to the question of Figure 8.8, to proceed to the third step which is using shortcuts to make a tour out of the resulting network.

The shortcuts step is straightforward. You are sequentially presented with a series of potential shortcuts (see Figure 8.9), and asked whether or not to use each shortcut. In Figure 8.9, the shortcut would substitute link (1,2) for links (1,3) and (3,2). Notice that the cost improvement is also displayed. Hint: Do not use a shortcut if the network resulting from it becomes disconnected!

After all shortcuts are obtained, the resulting initial tour is displayed (see Figure 8.10).

Improving the initial tour can be done by selecting 2-INTERCHANGE or 3-INTERCHANGE from the -- IMPROVED TOUR -- menu. Either option allows for the user to go "directly to the optimum" or go "step by step". "Directly to the optimum" only shows the final (2-optimal or 3-optimal) tour, whereas "step by step" shows all interchanges. Figure 8.11 shows a typical 2-interchange, and Figure 8.12 shows a typical 3-interchange. Figure 8.13 shows a 3-optimal tour. Note: All 3-interchanges are strict, that is, exactly 3 links are exchanged each time. This means that even after a 3-optimum is found, one might still be able to further improve upon it by performing some 2-interchanges! Obviously, the reverse is also true.

At this time, the -- EXACT SOLUTION -- menu is not operative.

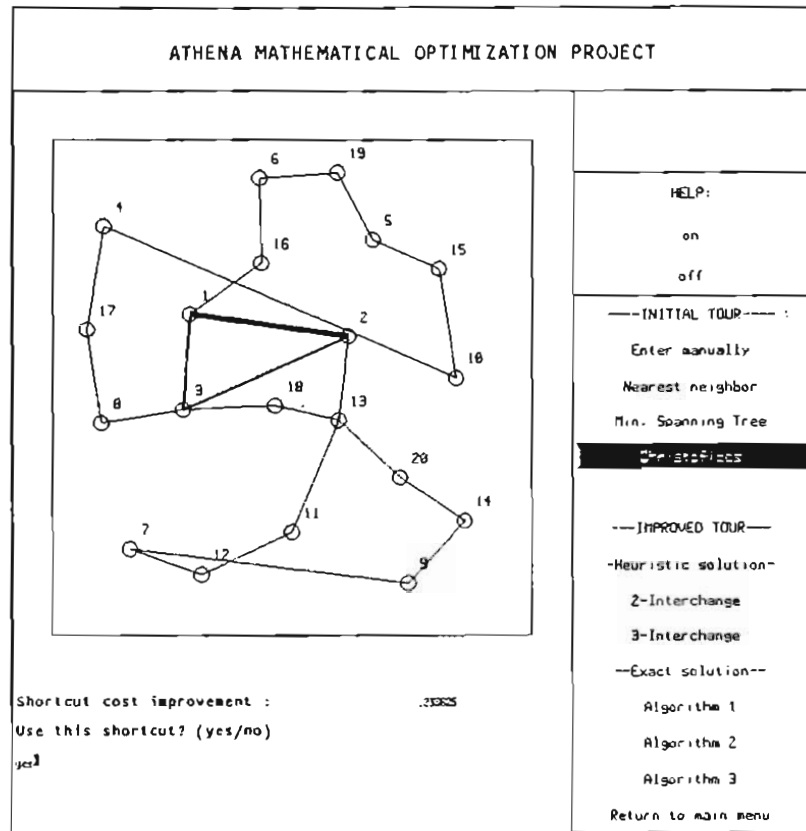


Figure 8.9: Christofides algorithm: Improving by using shortcuts.

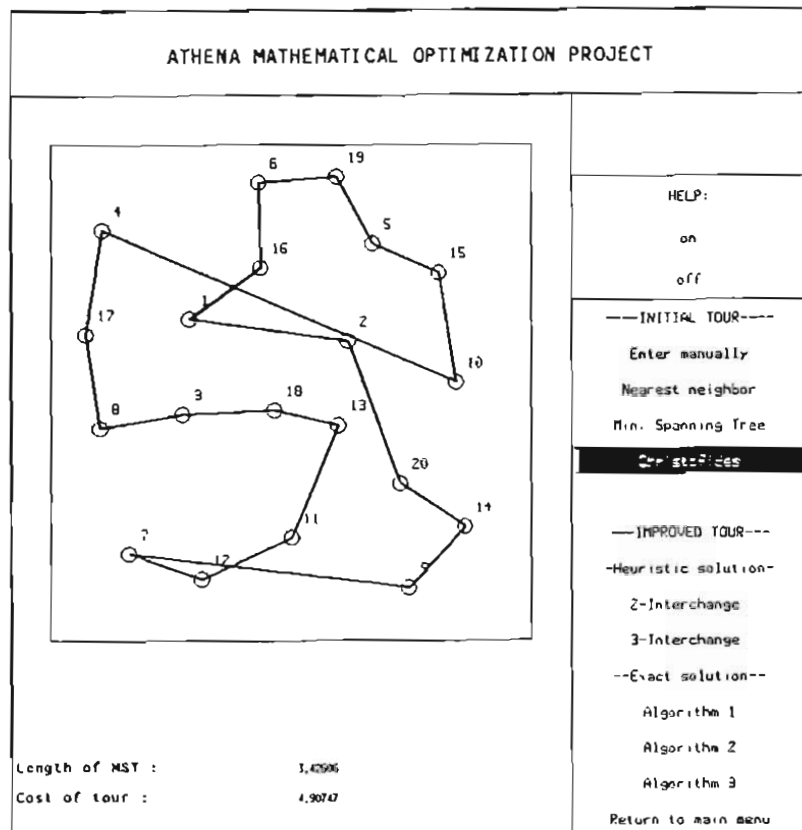


Figure 8.10: Christofides algorithm: Final tour

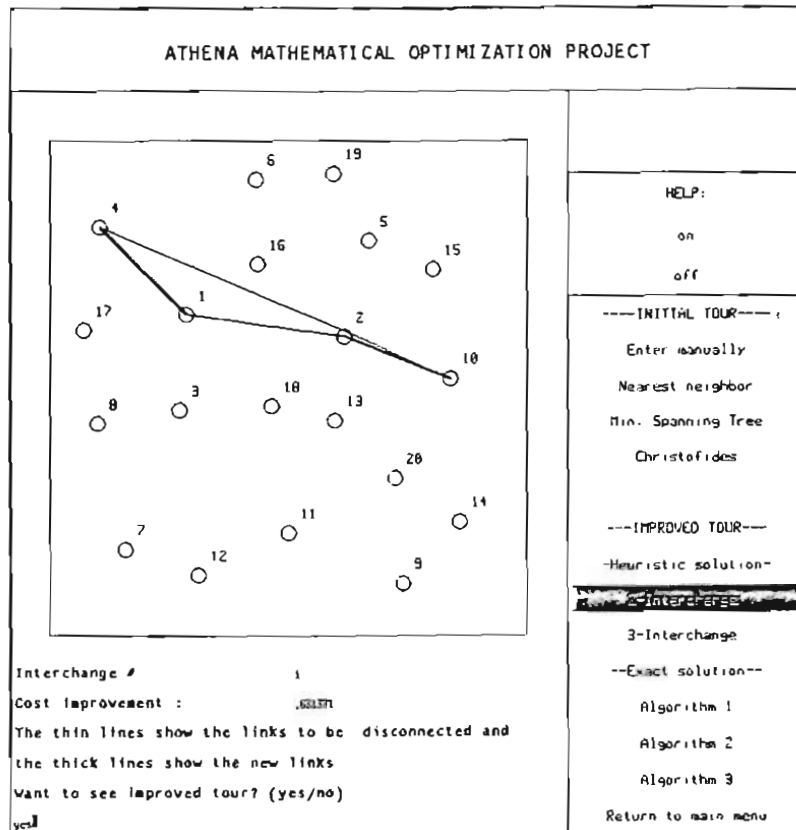


Figure 8.11: A 2-interchange

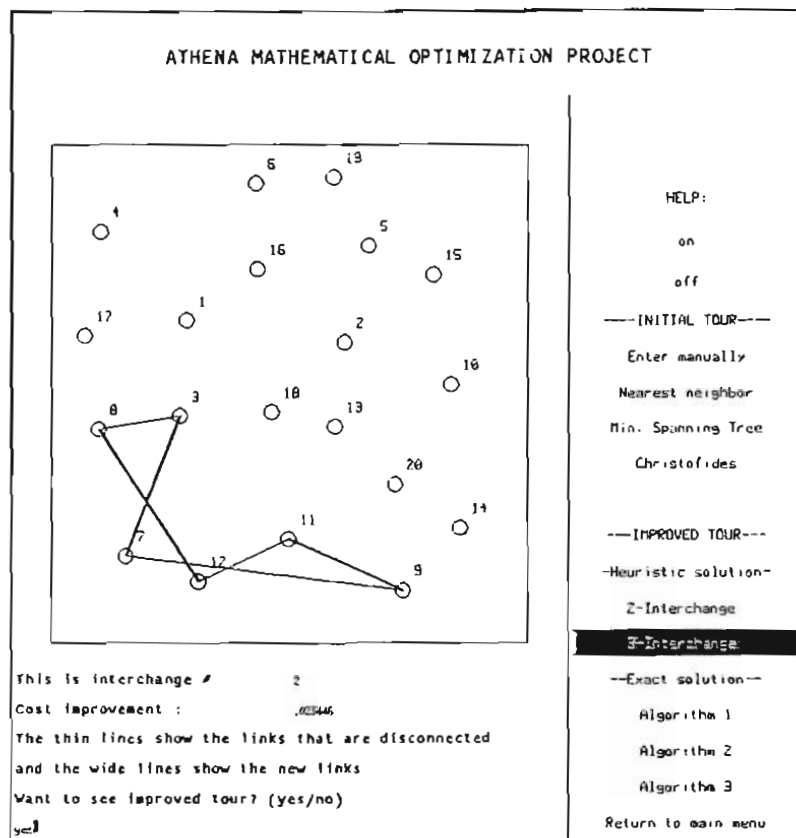


Figure 8.12: A 3-interchange

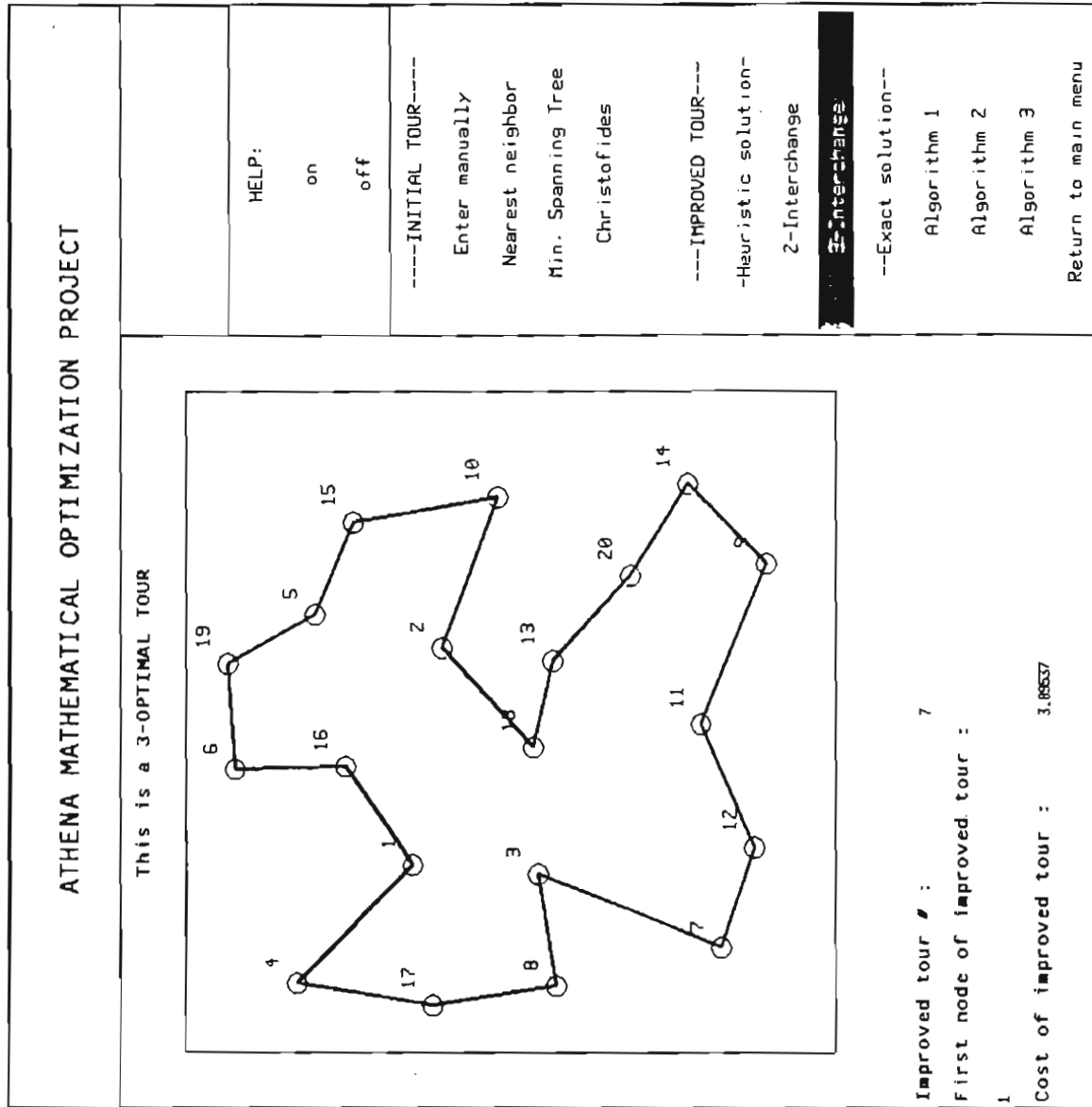


Figure 8.13: A 3-optimal tour

Directions for further work

This is essentially an open ended application, for there are so many TSP algorithms, both exact and approximate. For the CHRISTOFIDES heuristic, a matching module can be added and would be helpful.

Reference

Lawler, Lenstra, Rinnooy Kan and Shmoys (1985). Larson and Odoni (1980), Ch.6.

CHAPTER 9

THE VEHICLE ROUTING PROBLEM

Developer: Christopher Hrut

Introduction

Package vrp is an implementation of the Clarke and Wright "savings" algorithm for the vehicle routing problem. It allows the user to define a network, including a central depot, and route a capacitated fleet of vehicles through specified demand points. Routes and other algorithm information are displayed.

Summary of User Options

- 1) Input and edit a network (or read from a file)
- 2) Specify vehicle capacity and demands at each node.
- 3) Go through the Clarke and Wright procedure step by step.
- 4) Show routes, savings, and other information.
- 5) Store the network in a file.

Instructions

As in all network applications, this one too incorporates a "network editor". This particular network editor is virtually identical to the one developed for package spm (see Chapter 5). Given that Clarke and Wright requires a complete network, an interactive procedure has been added to the network editor in case the network specified is not complete (the procedure asks the user to specify the costs of the missing links, one by one).

Clarke and Wright in fact requires a triangle-inequality network, but this package can run the algorithm even if the triangle inequality does not hold.

After the network has been entered and tested for connectivity, the user is presented with the menu shown in Figure 9.1. The major difference between INTERACTIVE VRP ALGORITHM and NONINTERACTIVE VRP ALGORITHM is the amount of intermediate information that is displayed (large for the former option, small for the latter).

To start the INTERACTIVE mode, first specify depot location (node 7 in Figure 9.1). The result is shown in Figure 9.2, in which (a) the depot node is highlighted and (b) a new menu is displayed, in which the next step is to ENTER DEMANDS AND VEHICLE CAPACITY. Vehicle capacity is entered first (10 units in Figure 9.2).

After vehicle capacity is entered, the program asks for demand at each node, by highlighting that node (node 2 in Figure 9.3 -demand is equal to 3 units).

After all demands have been entered, select DO THE FIRST STEP from the menu shown in Figure 9.4. The result is shown in Figure 9.5, which shows a partial set of routes (in this case only one route - RT1, serving nodes 2 and 3 - is shown, all other nodes not yet considered). Notice the menu in Figure 9.5, allowing the user to observe several pieces of information at each step of the procedure, and the route table at the bottom, showing some statistics for the routes. The small box at the upper right of the screen shows TOTAL LENGTH, which is the total distance traveled to serve all points. Since many nodes are initially served only by round-trips, this total distance will go down at subsequent iterations. Selecting DO THE NEXT STEP sequentially will run you

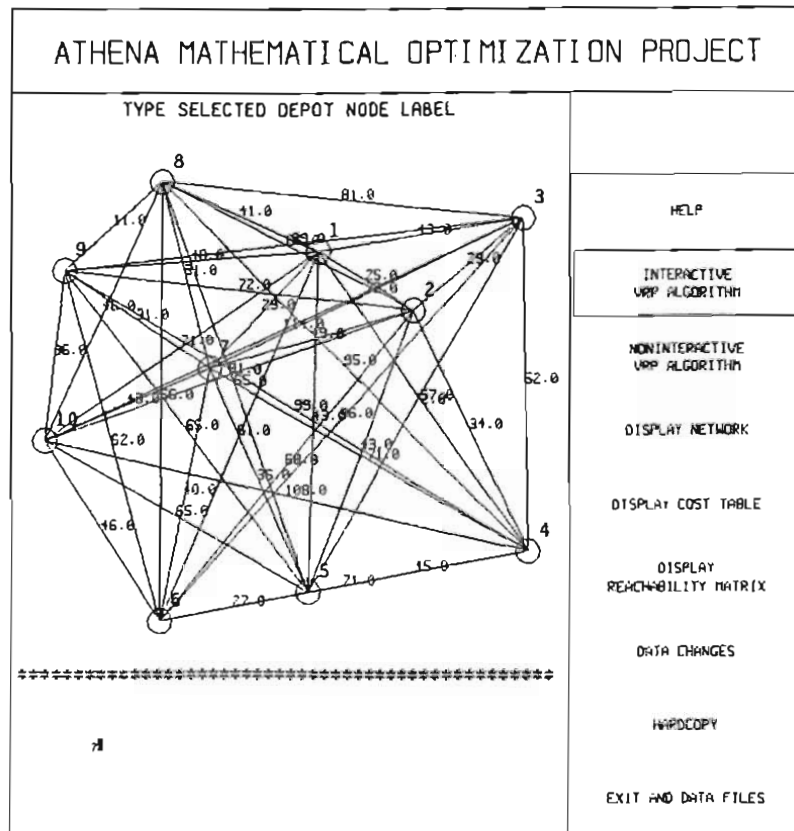


Figure 9.1: Specification of depot location

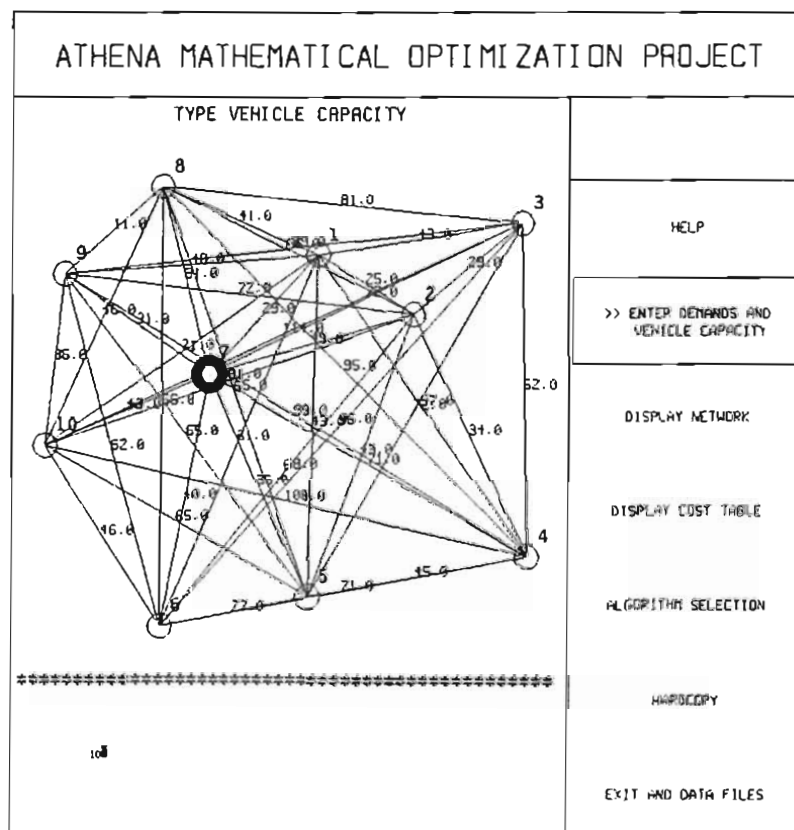


Figure 9.2: Enter vehicle capacity

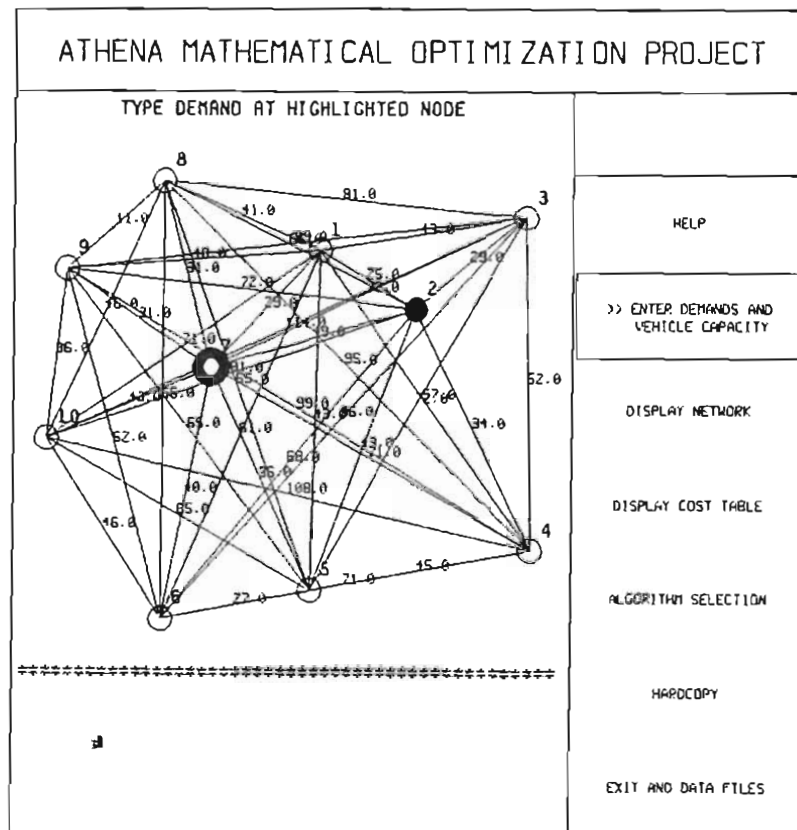


Figure 9.3: Enter demand at specified node (here node 2)

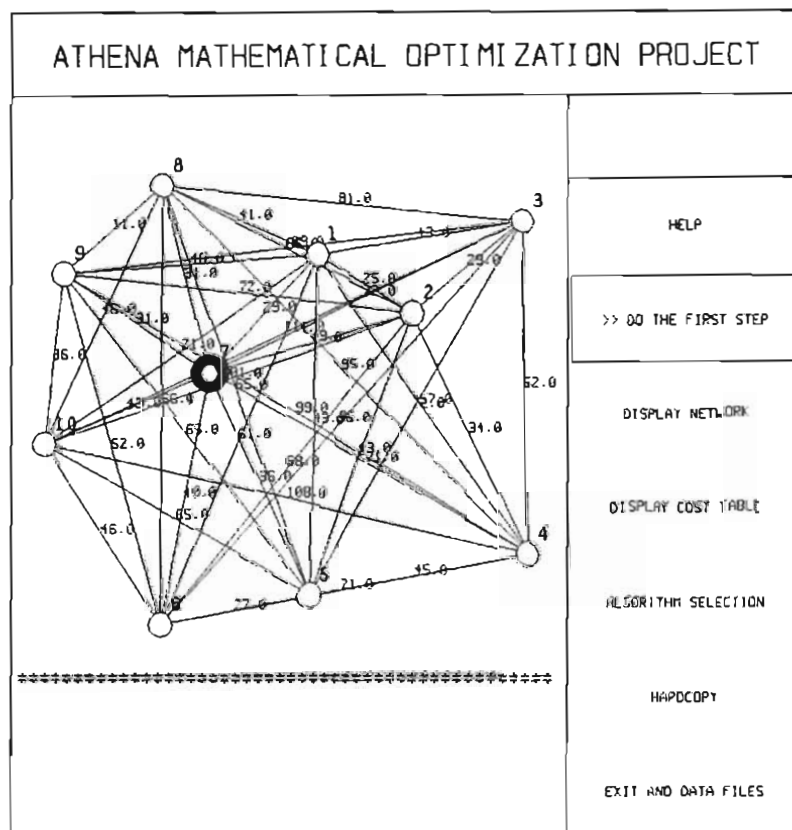


Figure 9.4: Proceed with first step

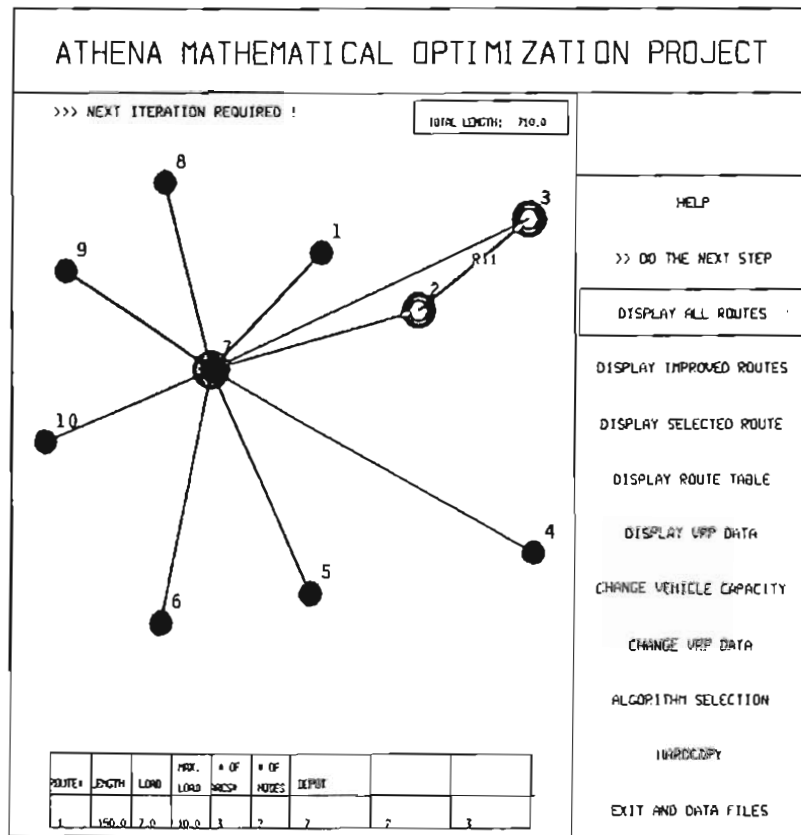


Figure 9.5: First iteration: Nodes 2 and 3 from a (partial) route, RT1

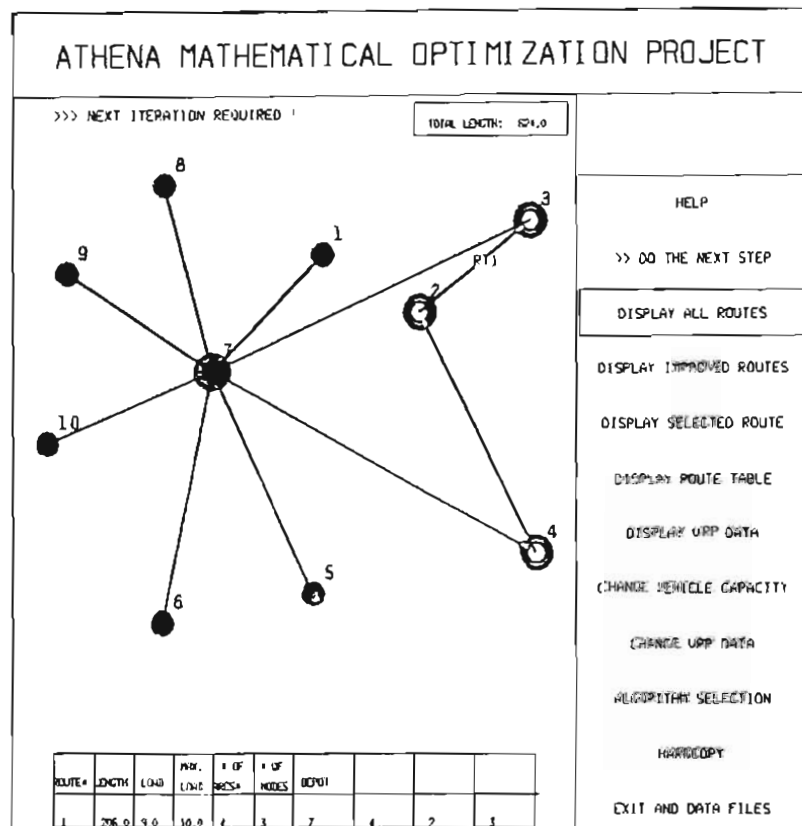


Figure 9.6: Second iteration: Node 4 is added to RT1

through the Clarke-and-Wright steps. In Figure 9.6, node 4 was added to RT1, and the route table was updated.

Figure 9.7 is what happens if DISPLAY VRP DATA is selected. Distances and "savings" are displayed at the bottom, node demands and vehicle capacity at the table above. Notice from the savings table that the highest two savings entries are between nodes (2, 3) and (3, 4), which in fact explains why nodes 2, 3, and 4 have been first selected for RT1.

In Figure 9.8, a new route, RT2, is initiated, and in Figure 9.9, a third route, RT3, appears. Figure 9.10 shows the solution at the final iteration of the procedure (one can see also a fourth route, RT4).

The NONINTERACTIVE option avoids all intermediate displays, and just shows the result at the final iteration of the procedure.

Directions for further work

Displaying a rank-ordered list of savings would further enhance this package. Also, some more verbose explanation of why each node is added (or not added) to a particular route might be worthwhile.

Reference

Larson and Odoni (1980), Chapter 6.

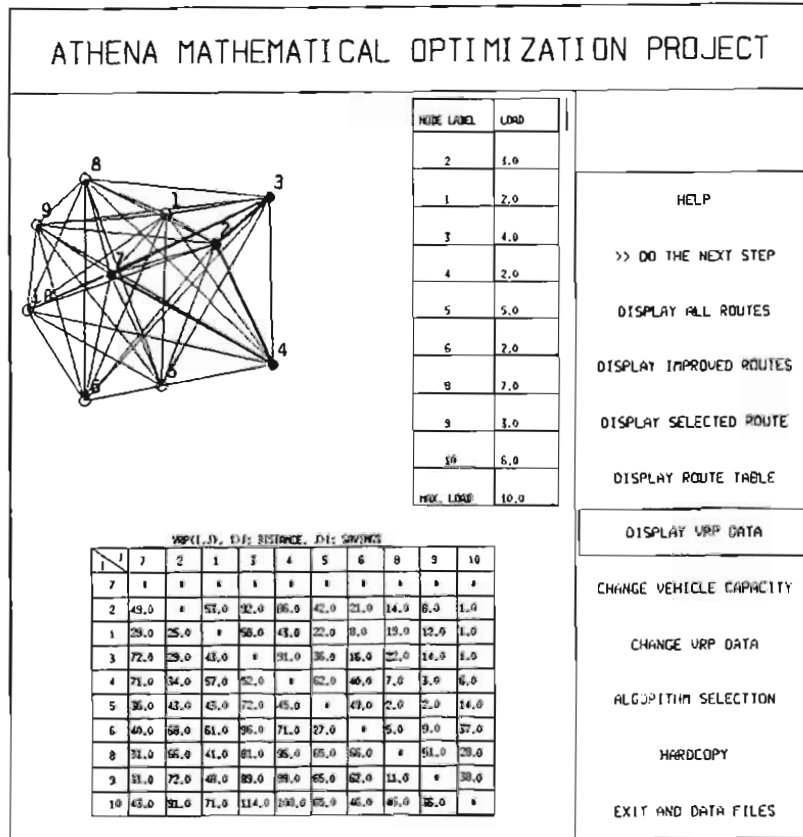


Figure 9.7: Costs, savings, and other data

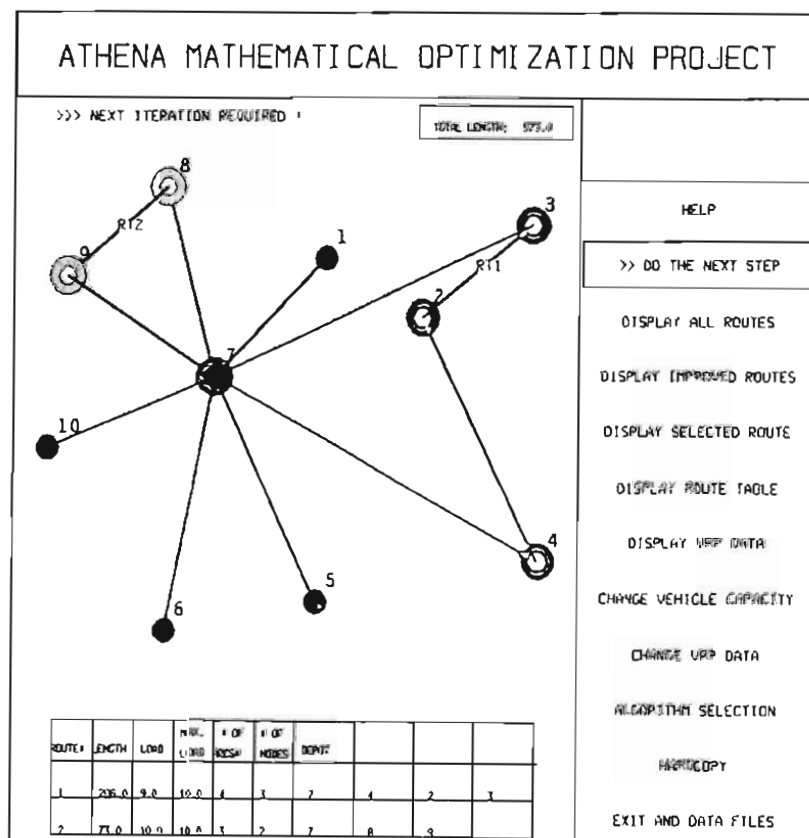


Figure 9.8: Third iteration: Nodes 8 and 9 form a route, RT2

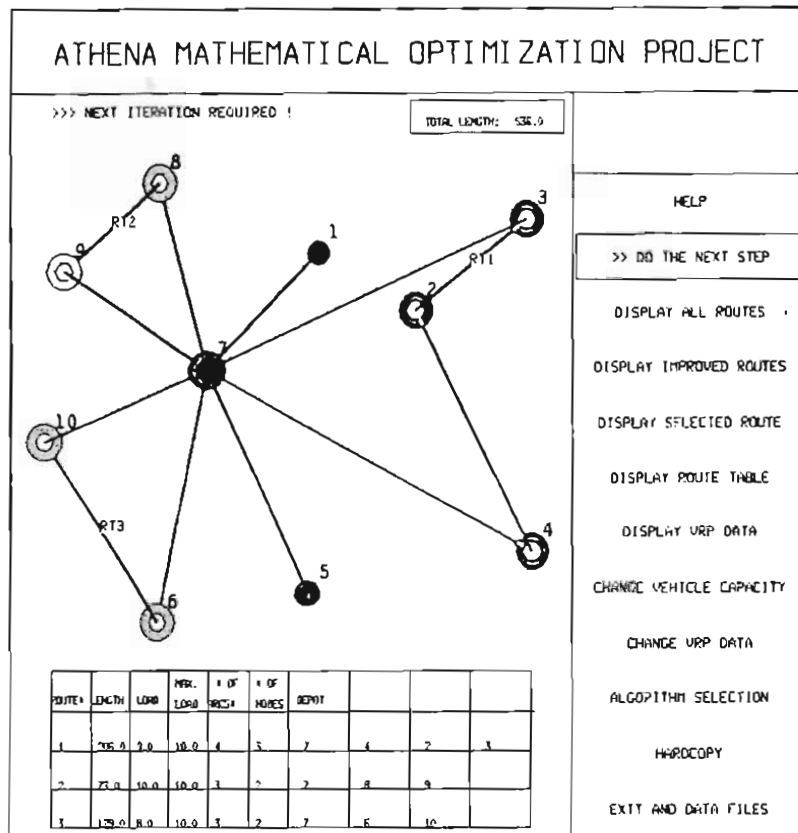


Figure 9.9: Fourth iteration: Nodes 6 and 10 form a route, RT3

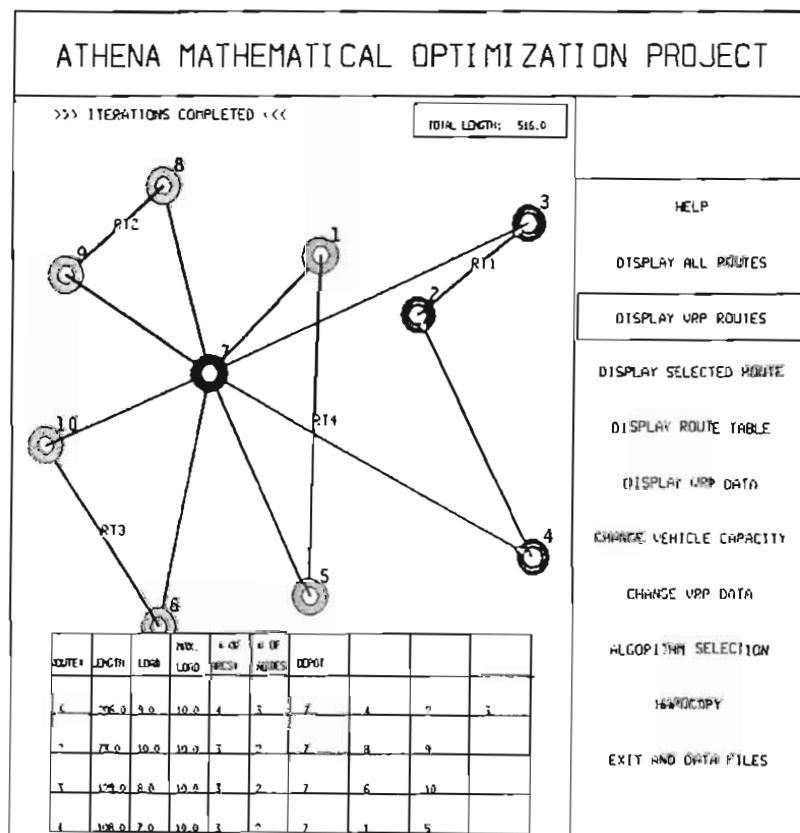


Figure 9.10: Final iteration

CHAPTER 10

CONCLUSIONS AND RECOMMENDATIONS

This report has described the set of software developed under the Athena project "Graphics Software for Mathematical Optimization". As of 9/88, nine graphics packages are operational. Our target is that by the end of this project phase (1/89), we will have at least one more package on line, most probably illustrating Karmarkar's algorithm in linear programming.

Although the project team has put a considerable amount of effort to making sure all of these packages are user-friendly and bug-free, it would be unrealistic to claim that there is no room for further refinement and improvement. Being continually faced with the choice between (a) continuing making marginal refinements in the already developed packages and (b) developing new packages, we invariably opted for (b). Still, we would very much appreciate suggestions for possible improvements.

Should a project of this nature continue in the future, and in addition to the suggestions for further work already outlined in the previous chapters, the following is a non-exhaustive list of possible directions for future work in this area:

(1) Developing additional packages: This is really an open-ended task.

Dynamic programming and constrained nonlinear programming are two areas that have not been explored so far. Many other areas exist, particularly in network optimization and nonlinear programming.

(2) Adding a color dimension: This would further enhance the visual impact of the software. Given that the Athena system does not currently support color

(although there is a small number of GPX (color) workstations), we did not seriously engage in such a task.

(3) Transferring the software to other systems. The Macintosh-II system is the prime candidate here, particularly now that Apple has developed its own version of Unix and X-windows. At this point, it is not clear how much additional effort would be involved in such a task, but given the user-friendliness of the Macintosh system, such a direction seems worth exploring.

(4) Developing computer-driven tutorials: This has been done in other Athena projects, and it would be worthwhile to attempt it for optimization as well. Some discussion and analysis as to how to maximize the pedagogical value of these tutorials should be carried out.

This page
to appear
on the next.

REFERENCES

Bradley, S., A. Hax, and T. Magnanti, "Applied Mathematical Programming", Addison-Wesley, 1977.

Garfinkel, R., and G. Nemhauser. "Integer Programming", John Wiley and Sons, 1972.

Larson, R., and A. Odoni, "Urban Operations Research", Prentice-Hall, 1980.

Lawler, E., J.K. Lenstra, A. Rinnooy Kan, and D. Shmoys, "The Traveling Salesman Problem", John Wiley and Sons, 1985.

Luenberger, D., "Introduction to Linear and Nonlinear Programming", Addison-Wesley, 1973.

(to be completed in the final version).