Fall 2006

Design and Development of a Software Module for Minimizing Transportation Cost

Gopalakrishna Udupi
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

SOFTWARE MODULE FOR MINIMIZING TRANSPORTATION COSTS

DESIGN AND DEVELOPMENT OF A SOFTWARE MODULE FOR MINIMIZING TRANSPORTATION COSTS

Gopalakrishna Udupi

Regis University

School for Professional Studies

Master of Science in Computer Information Technology

Abstract

The goal of this project is to design and develop a software module to solve a transportation problem, relating to minimizing costs to transport finished goods from multiple origins to multiple destinations. The transportation problem will be modeled as a linear programming model, using AMPL linear programming (LP) software. A graphical user interface (GUI) will be developed to enable the user to enter the data and parameters for the transportation problem. The GUI will be developed using C# programming language within the Microsoft® .NET framework. The GUI will also enable the user to launch the AMPL module to solve the transportation problem to calculate optimum transportation costs. A relational database will be designed and developed to store the parameters and data for the AMPL LP module. Both the AMPL LP model and the GUI will be interfaced with the relational database.

Table of Contents

List of Tables and Figures

Chapter One: Introduction

This software project was intended to design and develop a windows application to optimize transportation costs. The primary objective to be met by this application was to minimize costs to transport multiple product lines from several origins to several destinations. The secondary objective was to learn how to interface disparate software modules, such as a linear programming module, a Windows application and a relational database, in order to solve a business (transportation) problem.

Project Goals

The primary goal of this project was to learn how Linear Programming (LP) techniques could be applied, using a combination of software tools to solve a hypothetical business problem. The hypothetical business problem was related to minimizing the total cost to transport finished goods from multiple manufacturing plants (origins) to multiple warehouses (destinations). When solving the business problem using the software tools and LP techniques, knowledge of a programming language, database package and a modeling software package would be gained. This knowledge could be used to solve real-world business problems of the same nature. This solution would be more flexible and customizable than off-the-shelf packages available for optimizing transportation costs, such as the Logility Voyager® solutions suite.

As this project plan was executed, a hypothetical business problem was selected in the area of transportation of goods from manufacturing plants to distribution centers across the country. An LP model, with an objective of minimizing transportation costs,

was developed for the problem. A mathematical modeling tool was used for this purpose. This model was interfaced with a relational database, which contained the data related to the problem. Finally, a graphical user interface (GUI) was developed to enable the user to enter the data and parameters related to the problem and solve it.

A final project outcome analysis report was written after all the components for the project were designed and developed. The background of the business problem was described. The mathematical model and all the software components designed and developed for solving the problem were also described in detail. This Professional Project Paper is the collation of these above-mentioned documents, along with all additional project details and project outcomes.

Barriers and/or issues

The barriers against successful implementation of this project were the following:

a) The business problem could be too complex to model using LP techniques
b) The tools used to develop the GUI and data repository (C# and Access) are new and continually changing.

The project discusses how these issues were addressed.

Scope of project

The long-term objective of the Minimize Transportation Cost project was to solve a critical business problem facing the shipping and transportation department of a manufacturing company. The critical business problem typically involves minimizing the costs to transport goods from manufacturing plants to warehouses of a real-world

business. This usually involves multiple origins and destinations, with layered transportation costs. Transportation in the real world might not lend itself to being modeled as a linear problem and might need non-linear modeling techniques. However, the scope of this proposed project was limited to solving a hypothetical transportation problem. If the business problem could be solved using mathematical modeling techniques, relational database and a GUI, then the project could be expanded to solve a real world business problem. However, if the outcome of this project was negative, or only partially successful, then the resulting "lessons learned" knowledge could still be a great asset to future research and/or engineering projects that attempt to solve this same problem.

Outline of the project

The project was conceptually organized into two parts. The first part dealt with the transportation problem, and the methods and tools to solve it. It also considered in detail the chosen problem and its software model. The first part is covered by the chapters as follows

1. Introduction
2. Literature review
    - i) Research overview
    - ii) Linear programming (LP) techniques for solving transportation problems
    - iii) Software tools for modeling/solving linear problems

iv) Reasons for choosing AMPL modeling package.

3. Project Methodology

i) Methodology Overview

ii) Development of the AMPL Model and RDB interface

iii) Design and Development of the GUI for the transportation module

1. Multi-commodity transportation problem

i) Description of the problem

ii) AMPL model for the transportation problem

iii) Data for the Transportation problem

The second part of the project dealt with the graphical user interface for the AMPL LP model, as covered in the following chapters:

2. User interface for the transportation problem

i) Principles of good GUI design

ii) Design and development of the GUI

3. Testing and validation of the GUI

The final part of the project considered the lessons learnt from this project and ways to further enhance it:

4. Lessons learnt and conclusion

Chapter Two: Literature review

Research Overview

The theoretical research performed for this project was in the area of Linear Programming (LP) techniques to solve business problems in manufacturing and distribution. Another type of research performed for this project was experimental research. This involved selecting a software package to create the LP model and solve the problem. The next step was to research and learn how to develop the model itself. Experimental research also involved choosing a software package/tool to develop the graphical user interface (GUI) for the LP model. Finally, the selected GUI tool was researched and learned, in order to develop the GUI for the LP model.

Review of existing solutions

In today's competitive world, companies have manufacturing plants and warehouses in different, geographically distant locations, from which to deliver products to the customer in the timeliest fashion. Thus transportation costs incurred to deliver finished products from manufacturing plants to warehouses become a big component, affecting the profitability of these companies. Such companies can increase profitability by minimizing transportation costs. Transportation costs are minimized by optimizing the amount of goods transported along the various routes between origins and destinations. There are many off-the-shelf solutions available to companies to optimize transportation costs. Examples of such solutions are Logility Voyager® solutions suite and Oracle's JD Edwards EnterpriseOne Strategic Network Optimization® module.

These modules are typically tightly integrated with the company's Enterprise Resource Planning (ERP) backbone. Any changes to the data being passed between the

two systems or setup information within the two systems require programming changes and extensive testing. For example, a company may have plans to acquire a manufacturing plant and a set of related distribution centers. Bringing these into the ERP system and the off-the-shelf module will be a long-term project. The software module developed as part of this project aims to be a prototype of a quick and highly customizable solution that can be used for optimizing transportation costs for a department or division not served by the ERP and off-the-shelf optimization module.

Linear Programming

After World War II, universities and industry conducted considerable research into solving large-scale problems related to flow of commodities between industries, planning large-scale military operations and crop rotation. As efforts were made to provide a mathematical framework for these problems, linear programming emerged as a strong tool for solving them.

If the system exhibits a structure that can be represented by a mathematical equivalent, called a mathematical model, and if the objective can also be quantified, then some computational method may be evolved for choosing the best schedule of actions amongst alternatives. Such use of mathematical models is termed mathematical programming (Dantzig, 1963). Mathematical programming has been used for solving problems involving maximizing profits and minimizing costs, subject to constraints on resources, capacities, supplies and demand. Linear programming is among the most powerful of mathematical programming techniques. Linear programming problems can

be solved in many ways. The Simplex algorithm is the most efficient technique for solving linear problems, and is also the most conducive to being developed as a software model.

Simplex method

One of the common techniques of solving Linear programming problems is by using the Simplex algorithm. The Simplex algorithm solves a LP problem algebraically (Dantzig, 1963). The algorithm has two basic parts. First, it finds out whether a given basic feasible solution (BFS) is an optimal solution. If not, it obtains an adjacent BFS with a larger or smaller value (depending on whether the problem is maximization or minimization) for the objective function. The Transportation Simplex Method is a special version of the Simplex Method used to solve transportation problems.

Linear programming in AMPL

This project concentrated on linear programming, which is the best-known and easiest method for modeling and solving. AMPL is a language for specifying such optimization problems. AMPL provides an algebraic notation that is very close to the way a problem is described mathematically. Simple linear programs can be replicated and combined to dealing with complex problems.

Separation of model and data is the key to describing more complex linear programs in a concise and understandable fashion. This project achieves this by interfacing the AMPL LP model with a relational database.

Other mathematical modeling tools

There are other mathematical tools, such as GAMS. Spreadsheet solutions, such as Excel, ILOG, are available but they are lacking in certain features that make the modeling tool easy to use and understand. These features are: availability of control flow statements, procedures and intuitive modeling commands. Some of the other features that make AMPL superior to other models are availability of OSBC/OLE linkage, Windows-based IDE and diagnostic tools.

Transportation problem

The transportation problem is a classic operations research problem where the objective is to determine the schedule for transporting goods from origins to destinations in a way that minimizes the shipping cost, while satisfying supply and demand constraints (Fourer, Gay and Kernighan, 2003). The transportation cost problem is a common type of minimum cost flow model. It is a specialized form of a network flow model, where nodes representing origins and destinations are connected by arcs that carry flows of some kind. There are a few existing solutions for the problem this project examined.

Methods for solving the transportation problems

Transportation problems (TSP) can be modeled as Linear Programming (LP) problems. To set up the transportation problem as a LP problem, the following elements need to be considered:

Variables: The variables in the LP model of the TSP will hold the values for the number of units shipped from one source to a destination.

X_{ijo} = Number of units of product p, shipped from source i to destination j

Sets: products, origins and destinations will be stored in sets to enable separation of the LP model and its related data.

Parameters: supply at origins, demands at destinations, and transportation limits imposed on an origin-destination pair, are stored in parameters.

Objective Function: The objective function is a minimization problem that seeks to minimize the total transportation cost.

Let C_{ijp} denote the cost of shipping one unit of product p from source i to destination j.

$$\text{Minimize } Z = \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{p=1}^{r} C_{ijp} X_{ijp}$$

Constraints are the set of equations and/or inequalities that restrict the solution space of the problem. If a problem is not constrained by equations, the solution space will not be well defined (Fourer, Gay and Kernighan, 2003)

Chapter Three: Project Methodology

Methodology Overview

This project involved designing and developing a software module for solving a transportation problem. Therefore, different methodologies were used to develop the AMPL module and the GUI for the AMPL module. Developing the AMPL module

involved choosing a textbook transportation problem involving multiple products, origins and destinations and modeling it using AMPL commands. After the development of the model, the relational database for storing the data for the AMPL LP model was designed and developed. Once the relational database was designed and developed, it was interfaced with the AMPL module.

The next major step was to design and develop the GUI. The rapid prototyping method was used for designing and developing the GUI. The details of the design and development processes for the GUI are described in the following sections.

Development of the AMPL model and RDB interface

The selected transportation problem was modeled as a linear problem using AMPL commands. The objective function was developed based on the objective of minimizing the total transportation cost. The constraints were developed in order to ensure that the total amount produced was equal to the total amount shipped.

The relational database was designed and developed based on the data and parameters for the AMPL LP model. Then it was interfaced with the AMPL LP model. The AMPL model, its data and the interface with the RDB are described in detail in Chapter 4.

Design and development of the GUI for the transportation module

The graphical user interface (GUI) for the transportation problem modeled in AMPL was developed using Visual C# and .Net technology. The .Net framework was chosen because the of the advantages outlined in the following section

Advantages of the .Net Framework

The .Net framework has the following advantages over the other software development tools and frameworks:

a) Interoperability and management: One of the most significant advantages of the .NET framework is its level of interoperability with other languages, applications, and systems.

   The solution designed and developed in this project is a Windows application. However, it could be enhanced to run on the web and to interface with other web applications. At the heart of .NET is the ability to help businesses integrate and manage their web-based solutions through web services. .NET enables modern software applications to communicate through standard Internet protocols, such as XML and SOAP, creating a channel through which internal and remote systems can easily interact. Applications hosted in-house—in addition to external systems—can be "stitched together," allowing businesses to meet their unique business needs quickly, through specialized yet economical solutions (MSDN documentation, 2006).

b) Greater support for security: The .Net framework allows the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. These distributed applications, developed using the .Net framework, work seamlessly with the network security framework.

c) Easy deployment and maintenance: The .NET framework makes it easy to deploy applications. In the most common form, to install an application, one needs to copy the application, along with the components it requires, into a directory on the target computer. The .NET framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET framework ensures that all the components that the application depends upon are available on the computer before the application begins to execute.

Among the various languages available in the .Net framework, C# was chosen for this project, for the reasons mentioned in the following section.

Advantages of the C# programming language

a) Rapid development tool: Visual C# is a very robust object-oriented language and is designed to be a fast and easy way to create .NET applications, including web services and ASP.NET web applications. Applications written in Visual C# are built

on the services of the common language runtime and take full advantage of the .NET framework.

b) Commonality with C, C ++ and Java: Since C# is very similar to C and C++, it has all the powerful features of these languages. A developer familiar with these languages can easily become productive in C#.

c) Superior to C and C++ in some aspects: Development in C# is simpler than it would be in C and C++ languages, because some of the more complex aspects of these languages, such as namespaces, classes, enumerations, overloading, and structured exception handling, have been simplified. C# also eliminates C and C++ features, such as macros, multiple inheritance, and virtual base classes.

d) Superior to VB.net: C# is a more robust object-oriented language than VB.net. Thus it brings the benefits of object-oriented development to the developer in a better way than VB.net.

Requirements for the graphical user interface

The GUI for this project should meet the following requirements.

a) The user should be able to navigate, from a main screen to the various screens to enter/modify the parameters and data related to the transportation problem.

b) The user should be able to enter, delete and save values that will be used as origins in the transportation problem.

c) The user should be able to enter, delete and save values that will be used as destinations in the transportation problem.

d) The user should be able to enter, delete and save values that will be used as products in the transportation problem.

e) The user should be able to enter and delete values for the supply from each origin per product.

f) The user should be able to enter and delete demand at each destination per product.

g) The user should be able to enter transportation costs for each origin-destination pair per product.

h) The user should be able to solve the transportation problem by executing the AMPL LP modules developed as part of this project.

i) The user should be able to switch between screens by without having to open and close them each time.

Formats for presenting results/deliverables

MS Word document

The final Masters thesis document will be delivered in MSWord format.

Diagrams

The navigation diagram for the GUI will be drawn using the drawing editor within MSWord.

Methodology outcomes

The methodology adopted worked well in delivering the results for the project. The AMPL module was developed by researching the problem, and modeling it, using

AMPL commands. The data and parameters for the AMPL module were modeled in a database using principles of entity integrity and relational integrity.

The graphical user interface was designed by applying principles of a good GUI design. It was then developed by using the features of the .NET framework. Microsoft Visual Studio® was the development environment used to develop the GUI. Active Data Objects (ADO.NET®) component of the .NET Framework was used to interface the GUI with the relational database. After the development of the GUI, it was also interfaced with the AMPL module, to enable the user to launch it and solve the transportation problem. Finally, testing was performed to validate the total solution, consisting of the AMPL module, GUI and the relational database.

Chapter Four: Multi-commodity transportation problem

The hypothetical transportation problem chosen for this project involved transporting multiple types of products, from several origins to several destinations at minimum overall cost.

AMPL model for the transportation problem

AMPL linear programming language provides many statements and constructs to enable modeling of LP problems. AMPL enables the programmer to define the LP problem in a manner that mirrors the algebraic form. For modeling larger LP problems, it is necessary to separate the model and data into 2 separate files to enable modularity.

Separation also enables modification of the problem data without changing the model. The LP problem is defined in an AMPL model file, which is described below. The data for the problem can either be stored in a text file or a relational database.

Three fundamental sets of objects underlie the transportation problem: sources (manufacturing plants), destinations (distribution centers) and product lines. These are declared in the beginning of the AMPL model.

```
set ORIG;
set DEST;
set PROD;
```

There is a supply for each product at each source and there is a demand for each product at each destination. Such non-negative quantities are defined as param statements, indexed over a set or a combination of sets.

```
param supply {ORIG,PROD} >= 0;  # amounts available at origins
param demand {DEST,PROD} >= 0;  # amounts required at destinations
```

The transportation model has been set up in such a way that the sum of supply for a finished good item equals the sum of demand for that item at all destinations. There is a check statement in AMPL that ensures that this test is performed on the data after it has been read. AMPL will issue an error if the “check” condition is violated. The check statement is given below.

```
check {p in PROD}: sum {i in ORIG} supply[i,p] = sum {j in DEST} demand[j,p];
```

There is a cost associated with transporting each product from each origin to all the destinations.

```
param cost {ORIG,DEST,PROD} >= 0;  # shipment costs per unit
var Trans {ORIG,DEST,PROD} >= 0;   # units to be shipped
```

The amount of product p transported from origin i to destination j, is defined as Trans[i,j,p]. The corresponding cost of transporting the goods per unit is cost[i,j,p]. The total cost for the combination is cost[i,j,p] * Trans[i,j,p].

Objective Function

The objective function is to minimize the total transportation cost of transporting all the product types from all the origins to all destinations. This is expressed in AMPL as follows:

```
minimize Total_Cost:  sum {i in ORIG, j in DEST, p in PROD} cost[i,j,p] * Trans[i,j,p];
```

Constraints

The first constraint states that the sum of all shipments of all products from an origin is equal to the supply available. Since the amount transported of a product p from an origin i to a destination j is Trans[i,j,p], the amount shipped to all destinations is : sum {j in DEST} Trans[i,j,p].

A parameter called supply has been defined, indexed over origins and products. Therefore the supply constraint is:

```
subject to Supply {i in ORIG, p in PROD}:  sum {j in
DEST} Trans[i,j,p] = supply[i,p];
```

The demand constraint is defined similarly, using the demand parameter and corresponding index variables:

```
subject to Demand {j in DEST, p in PROD}: sum {i in
ORIG} Trans[i,j,p] = demand[j,p];
```

The last constraint limits the sum of shipments of all products from an origin i to a destination j to limit[i,j]. Parameter limit has been indexed over origins and destinations.

All the above definitions, objective function and constraints are written into an AMPL model file with a .mod extension. Model file is read into AMPL using a `model` command

Data for the transportation problem

Data for an LP model can either be stored in a text file with a .dat extension, or in a relational database. As already stated, parameters and variables, defined in an AMPL model file, are indexed over sets. This indexed data is similar to the structure of relational tables in a database linked by foreign keys. The AMPL `table` declaration can be used to define connections between sets, parameters, variables and relational database tables maintained by other software. Storing the LP data in a relational database has the following advantages:

a) Data can be entered, stored and sorted easily when stored in a relational database. Most modern relational database applications provide GUI tools to develop applications over the tables
b) Better data integrity is maintained when a relational database is used, as rules can be developed to prevent accidental deletion of records
c) Parameters/variables in AMPL that are indexed over multiple sets correspond to tables with a composite key structure, and hence provide an easy understanding of how the data is structured.

For the above reasons, the data for the transportation model solved as part of this project was stored in a relational database. Microsoft Access was chosen as its structure is easy to understand and technical documentation is available. AMPL provides in-built table handlers to interface model files with relational databases.

As mentioned earlier, AMPL `table` declaration is used to establish a connection between the elements within a model and the tables in a relational database. The name and location of tables are specified as part of the table declaration. The following paragraphs describe how the entities within the transportation LP model correspond with the related tables in the Access database. The techniques for reading and writing data into the tables are also described.

The AMPL transportation model contains set for origins (ORIG), destinations (DEST) and products (PROD). Each of these sets corresponds to a table in the relational

database. The `table` statements for linking these sets with corresponding tables are given below:

```
     table Orig IN "ODBC"
"c:\ampl\amplcml\transportation.mdb": ORIG <- [ORIG];
     table Dest IN "ODBC"
"c:\ampl\amplcml\transportation.mdb": DEST <- [DEST];
     table PROD IN "ODBC"
"c:\ampl\amplcml\transportation.mdb": PROD <- [PROD];
```

Each `table` declaration has two parts. The part before the column provides general information, such as the table name, such as Orig, which is the name by which the table will be known in AMPL. The keyword “IN” states that all the non-key columns will be read-only. The strings “ODBC” and the path of the database specify the location of the table. After the colon, the `table` declaration gives details of correspondence between AMPL entities and the relational table columns. The key columns are specified by enclosing the square brackets [...]. The arrow indicates that the value from the key column is read into the AMPL set such as ORIG.

Next, the `table` statements to read the two parameters namely, supply and demand are described. The statements are given below:

```
     table Supply1 IN "ODBC"
"c:\ampl\database\transportation.mdb": [ORIG, PROD],
supply;
```

```
     table Demand1 IN "ODBC"
"c:\ampl\database\transportation.mdb": [DEST, PROD], demand
~ Demand;
```

Since both supply and demand are indexed over two sets, a composite primary key is required to store them in relational tables. The parameter supply is stored in a table called Supply1, which has a composite key consisting of columns ORIG and PROD. Supply from every origin-product pair is stored in a column called supply. For simplicity's sake, the same of the parameter in AMPL and the name of the columns in the table are kept the same. The same concepts apply to the `table` statement for reading in the parameter demand.

Finally, the statement relating to reading the transportation costs is explained. A table called TransportationCosts stores the unit transportation cost from all origins to all the destinations per product. It also stores the optimum transportation amount to be transported between each pair to minimize cost.

```
     table TransportationCosts "ODBC"
"c:\ampl\database\transportation.mdb": [ORIG,DEST,PROD],
cost IN, Trans OUT;
```

The above statement contains both an "IN" keyword and an "OUT" keyword. The "OUT" keyword ensures that after the LP problem is solved, the optimum transportation amounts are written for each node within the transportation problem.

After defining the connection between the sets, parameters and variables and their respective tables, the records from the tables are read using a `read` statement. The read statements for all the tables are as follows:

```
read table Orig;
read table Dest;
read table Prod;
read table Supply1;
read table Demand1;
read table TransportationCosts;
read table Limit;
```

After the model and data have been read into AMPL, the transportation problem is solved using the `solve` command. Upon receiving the `solve` command AMPL, solves the linear problem using the mathematical solver, specified in the file that houses all the commands mentioned in the previous paragraphs. Having such a file automates the process of reading the model and data, solving the problem and finally writing the optimum values in output tables. Writing the output to related tables in the database is performed using the `write` command. In this project, the `write` command was used for writing the optimum transportation costs back to the table – Transportation costs.

Solving the AMPL model

The AMPL model, along with its data, is solved using the `solve` command. The `solve` command displays the following information after its execution:

a) Name and version of the mathematical solve used for solving the problem.
b) Whether or not the problem could be solved. If problem cannot be solved, the phrase “infeasible solution” is displayed. If the problem can be solved, the value of the objective function is displayed, which in the case of this project is the total minimum transportation cost.
c) The number of iterations needed to solve the problem.

AMPL provides several commands that cause input to be taken from a file. One such command used in this project is the `include` command. The command `include` *filename*

is replaced by the contents of the named file. In this project, the named file contains all the table, model, read and write commands. This obviates the necessity of entering these commands separately and also allows the “include” file to be called from a user interface. In this project a graphical user interface has been developed (GUI) to allow easy entry of the model parameters and data related to the model.

Chapter Five: User interface for AMPL Transportation model

This chapter describes the technical design and implementation history of the graphical user interface (GUI) for this project. The requirements of the GUI were listed in Chapter 3. This chapter describes the principles of good GUI design and how they were applied to the GUI designed this project.

Principles of good GUI design

Following are the principles of a good GUI design:

a) The audience for a given GUI application should be known. In other words the designer should aware of the technical knowledge level of the expected users when designing a screen.

b) A common metaphor should be chosen for all the screens in the application. It is not always necessary. In many cases, the natural function of the software itself is easier to comprehend than any real-world analog of it (Tognazzini, B. (1991)).

c) Users' opinion should be actively sought when deciding the layout of fields and other controls.

d) While the GUI need not be a work of art, it should be appealing to the eye of the user.

e) If certain controls do not apply in a certain context, they should be disabled in that mode or context, to prevent confusion in the mind of the user.

f) The principle of safety should be applied through out the application. In other words, the user should be able to abort or exit from a screen/process, if he/she feels that it might compromise the data.

g) The GUI should be coherent and easily understandable. The user should be able to be able to use the controls and navigate from screen to screen intuitively.

h) Short cuts on the keyboard should be provided for controls, in case the mouse is not available.

Technical design of the GUI

The rapid prototyping method was adopted for designing and developing the GUI for this project. One of the main goals of the project was to demonstrate how LP techniques could be used to solve real world problems in the area of optimization. The tools used for developing the GUI for this project viz., Visual C# and MS Access are conducive for bringing ideas into reality in a rapid manner. In future revisions of the project, formal object-oriented techniques could be used to design the screens for entering the data and parameters related to the LP model and solving it. Figure 1 illustrates the high level view architecture of the graphical user interface developed for this project and the workflow between the various screens. Each of the screens in the graphical user interface and the C# controls used to deliver the functionality are described in detail in the following sections.

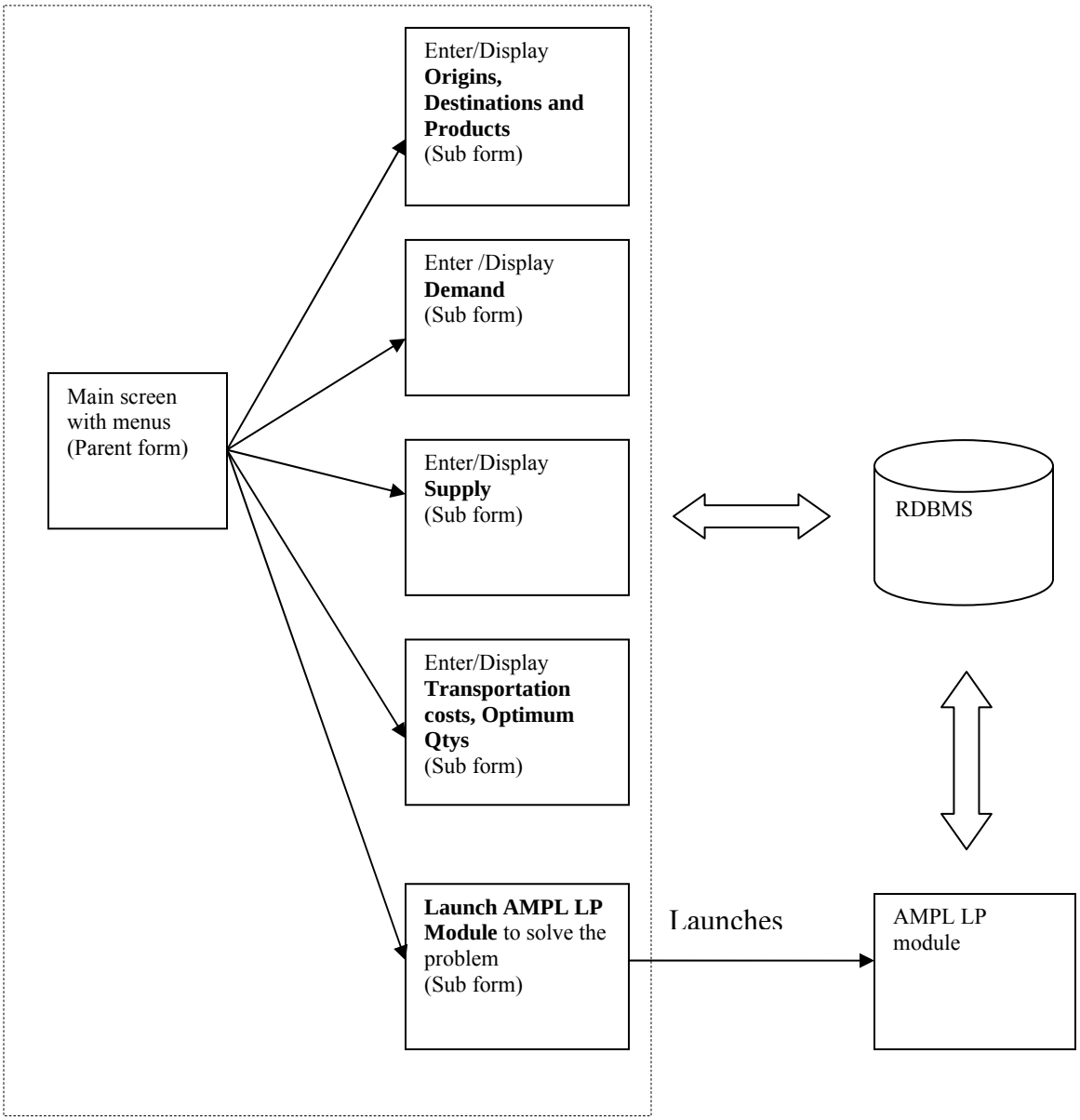


Figure 1 High level view architecture of the graphical user interface

Main screen

This screen is the main entry point to the application. It meets requirement a) mentioned in the list of requirements. It has a parent form, from where other "child" forms are launched. It has a menu, listing the names of other forms for performing other operations related to entering data and parameters for the LP model. The menu was created using the Visual C# control `MainMenu.` The individual menu items under `MainMenu` are used to launch the various sub forms for entering data and parameters for the LP model and solving it. The `button_clicked` event of each menu item has the logic to display the respective child form.

The GUI application for the AMPL LP module has been developed as a Multiple Dialog Interface (MDI). MDI applications present the user with a menu and have the ability to hold multiple windows open at the same time (Watson K., 2003). This is a requirement for the application developed for this project, as the user may have to switch between screens to check/enter various parameters related to the LP model, without opening and closing the screens each time. An MDI application is created as a windows application within Visual Studio .NET. The main screen that is the entry point to the GUI interface is designated as an MDI container, by setting the `IsMDIContainer` to be true. The other forms are configured as child forms, by setting the `MdiParent` property of the child screen to a reference to the main form. This property has to be set programmatically.

To display the child form from the parent (container) form, it is necessary to create an instance of the child form to be displayed and then display using the `Show()` function within C#. This logic is present in the `button_clicked` event of the menu items within the `MainMenu` control. This event is executed when the user selects a menu item, thus launching the selected form. Also the constructor function of the child form should be configured to have the parent form as a parameter. This establishes the parent-child relationship between the main form and the "called" form. The logic for establishing the parent child relationship, and launching the child form on selection of the menu item, is illustrated by the following code snippet (Fig 2).

```
private void menuItem3_Click(object sender,
                             System.EventArgs e)
{
     WindowsApplication1.frmCosts costsForm =
          new WindowsApplication1.frmCosts(this);
     costsForm.MdiParent = this;
     costsForm.Show();
}
```

Figure 2 The logic for establishing the parent child relationship

The screen shot for the design mode (Fig. 3) displays the different menu options.

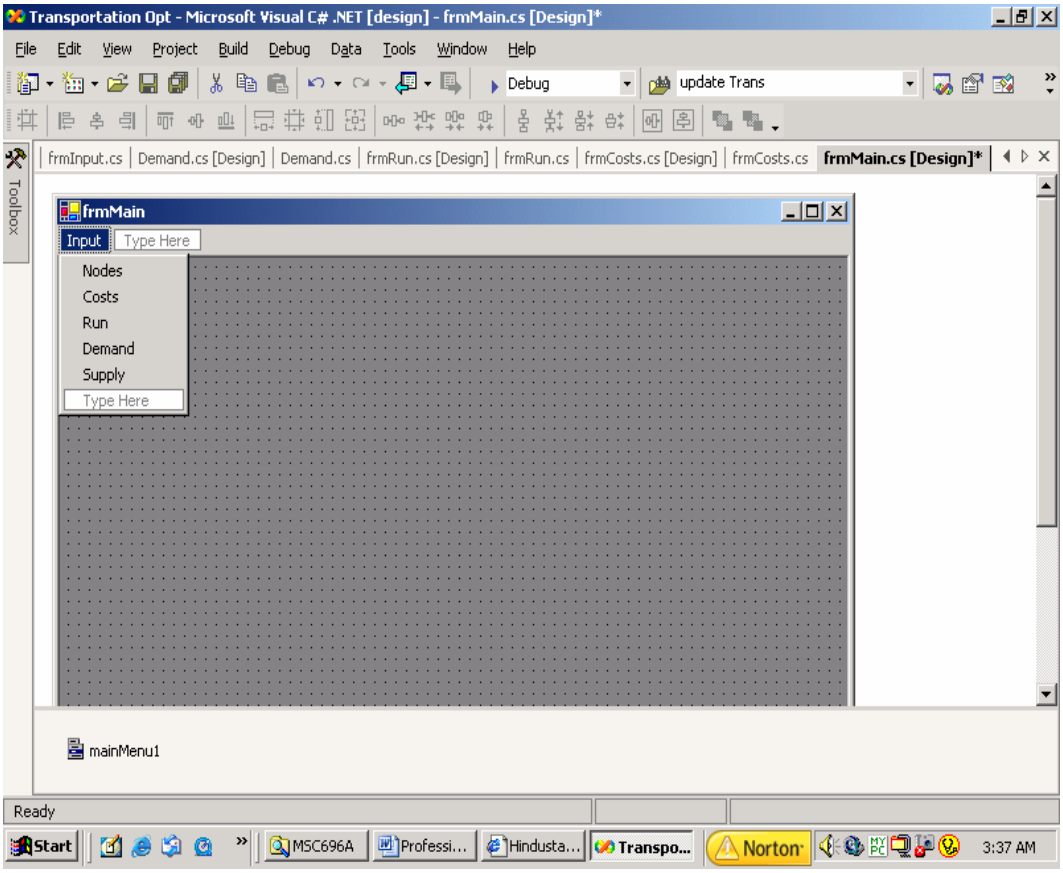


Figure 3 Screen shot of the design mode

Screen for entering/displaying origins, destinations and products

This form allows the user to enter and display the origins, destinations and products that are parameters of the transportation problem. Each of these parameters is entered and displayed, using the `ListBox` controls. `ListBox` control displays a list of strings, from which one or more can be selected at a time. There are buttons for adding and removing items from the List box. There is a separate field where the items to be added to the list can be entered. Then the user clicks the "Add" button to add the item to the list. The `button_clicked` event of each of these buttons contains logic for

adding and removing items from the list using the methods and properties provided with the `ListBox` control.

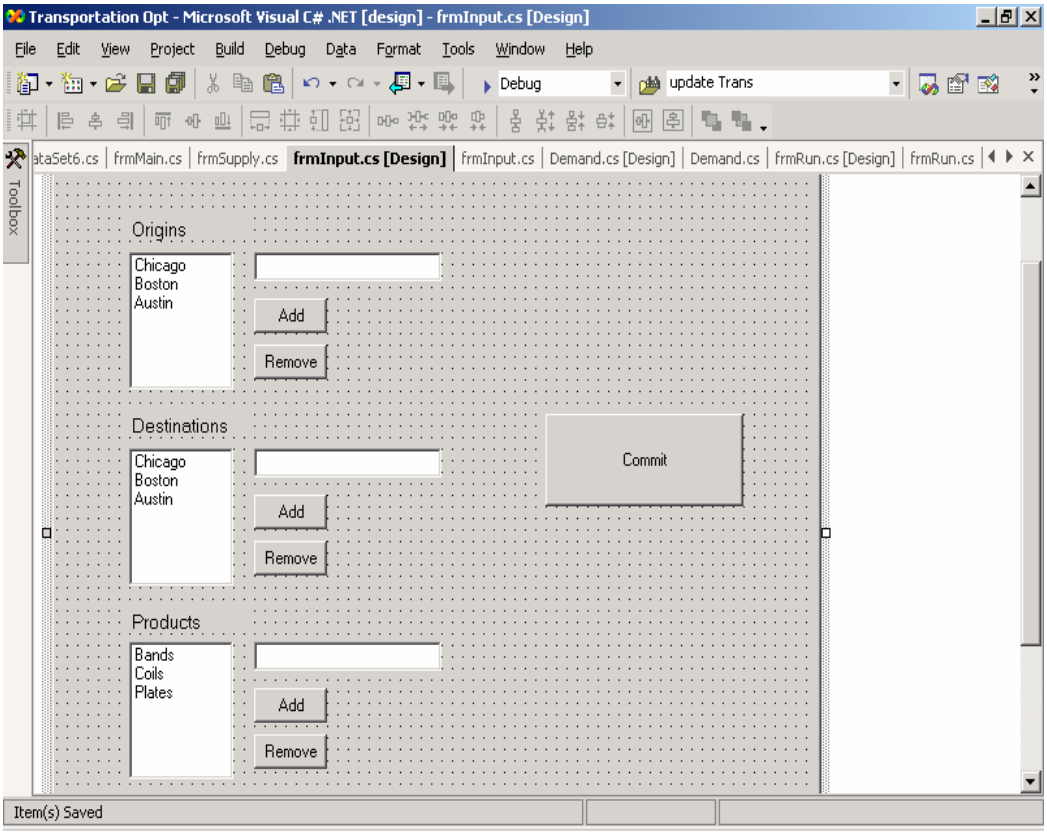


Figure 4 The form for entering origins, destinations and products, as displayed in design mode

After all the origins, destinations and products have been added (and finalized) to their respective ListBoxes, they are added to the relational tables by clicking on the "Commit" button. Each parameter is added to its respective table by using ADO.NET. ADO stands for Active Data Objects. ADO.NET is a set of classes provided within C# and the .NET framework to access data in a relational, data oriented format. This includes relational databases such as Microsoft Access and Microsoft SQL Server.

The code snippet in Fig 5 displays the logic for saving the contents of the `ListBox` for origins to the database. To ensure that the latest entries in the control are saved in the related tables, all the records in the table deleted. Then the contents of the `ListBox` are saved to the table. The same logic has been used to save the contents of the `ListBoxes` for destinations and products.

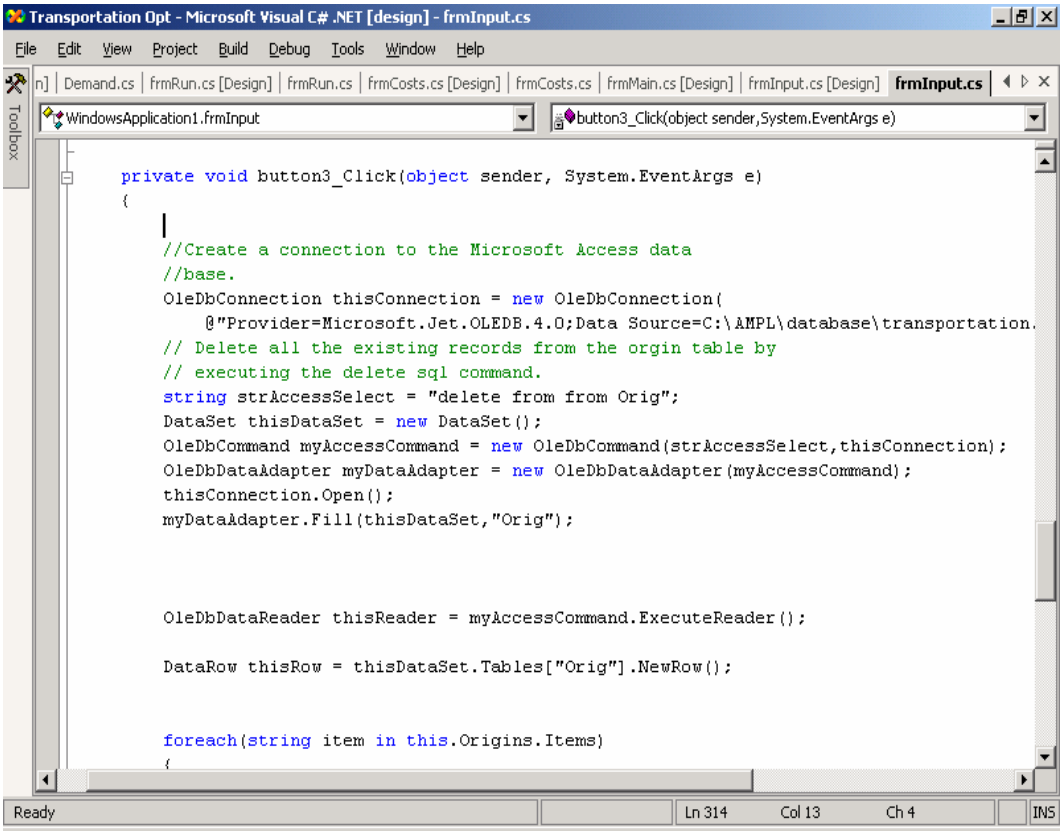


Figure 5 Snippet to show the logic for saving the contents of the `ListBox` for origins to the database

Screen for entering/displaying supply

The screen for entering and displaying supply, from the various origins that are defined in the transportation problem, is described in this section. This form displays supply information from the supply table in the relational database, via a DataGrid class within the .NET framework. DataGrid displays from data from a relational table, in a scrollable grid using ADO.NET. The procedure of binding a table to a DataGrid involves the following steps:

a) Connecting to the data source: This is accomplished by using the oleDBConnection class within the .NET Framework.
b) Opening the connection: The connection, established by using the oleDBConnection class, is opened by using the open method within the class.
c) Read/Write data from/to the table in the data source: This is accomplished by a class called OleDBDataAdapter. This class enables reading and writing data from/to a database via SQL commands. The data read from the data source is stored in an instance of a DataSet class.
d) Displaying the data in a DataGrid: This is accomplished by using the DataSet class within the .NET framework. The DataSet class stores the data read, using SQL commands and displays it in the DataGrid.

The fields on the DataGrid are Origin, Product and Supply for the Origin-Product combination. These are displayed in the screen shot of form in design mode in Fig 6.

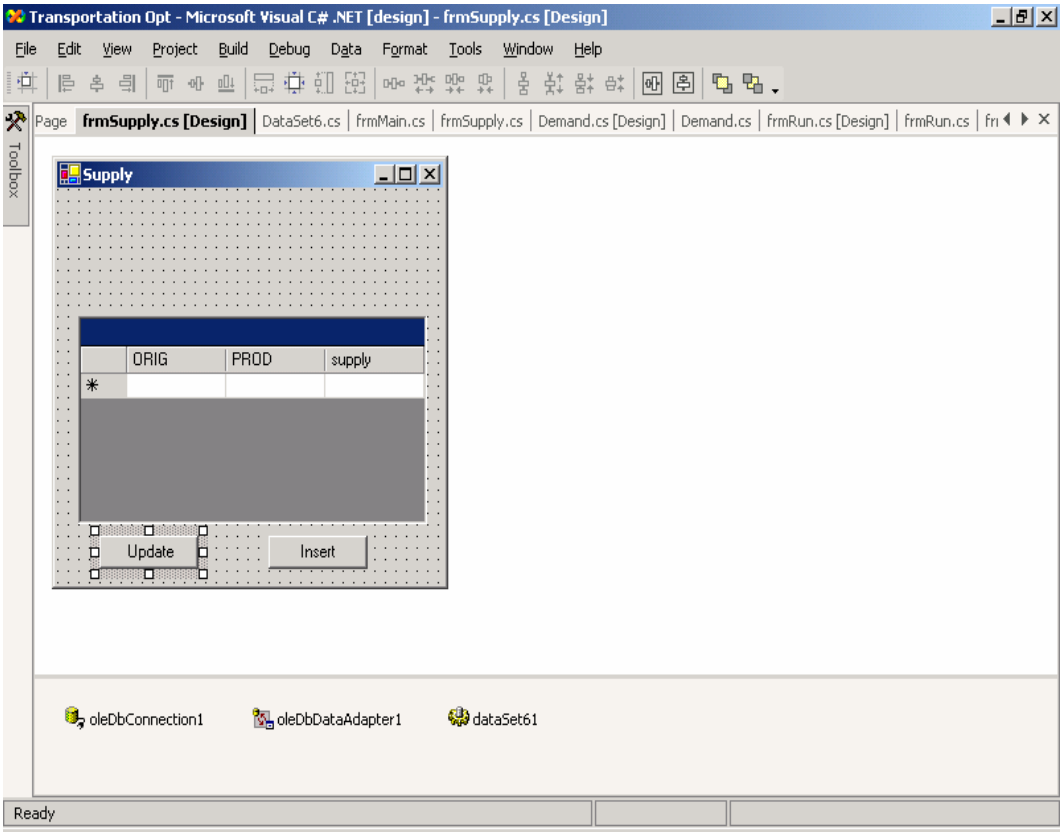


Figure 6 Origin, Product and Supply for the Origin-Product combination

There are two buttons for updating and inserting new records in the grid respectively. There is logic in the `button_clicked` event of each of these buttons to perform the respective functions on the DataGrid. The logic within the `button_clicked` event of the update button reads the changes made to the DataGrid, using the "GetChanges" method. These changes are loaded into a DataSet (as described in an earlier section). If this DataSet is not null, then the changes are saved to the database. Then the DataGrid is refreshed to display the latest information. The code snippet for this logic is displayed in figure 7.

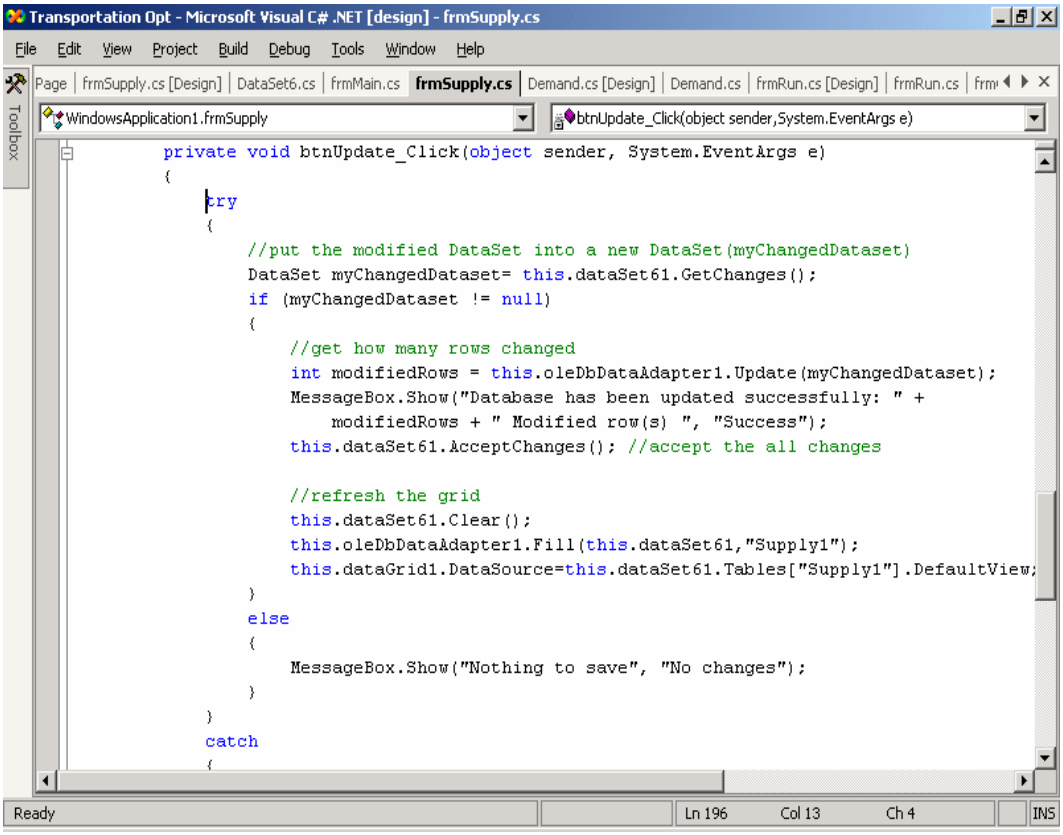


Figure 7 Code snippet to refresh the DataGrid

When the user clicks the Insert button, a message is displayed instructing the user to add the new row at the end of the grid and then click the Update button. Then the logic adds the row to the DataSet attached to the DataGrid. When the user clicks the update button, the newly added row is saved in the database.

Screen for entering/displaying Demand

The form for entering/displaying demand is very similar to the form for displaying supply. This form reads and displays data from the demand table within the relational database. This form also contains a DataGrid and associated classes within the .NET framework for entering/displaying data. The form has buttons to insert and update records from the DataGrid respectively.

This form is a child form to the main form explained earlier. This form is called from one of the menu items in the main form. The section on the main form explains how the main form is configured as a parent in this Multi-dialog Interface (MDI) application. In the demand form, this relationship is established by specifying the main form as a parent in the form constructor method. The code snippet displaying this logic is given in figure 8.

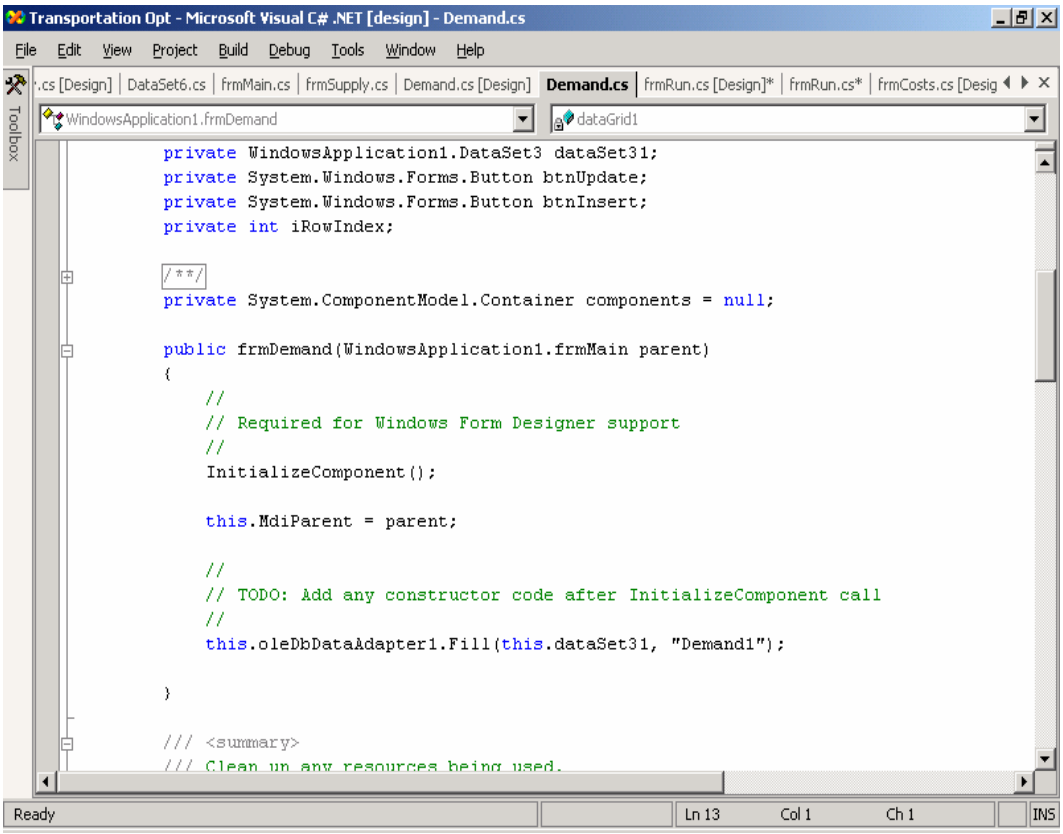


Figure 8 Code snippet for specifying the main form as a parent

Screen for entering/displaying transportation costs and quantities

This screen entering and displaying transportation costs for each origin-destination-product combination also contains a DataGrid. This DataGrid is linked to the table transportation costs in the relational database. This screen has buttons to add and delete records from the grid. When the user clicks on any of these buttons, logic related to performing the corresponding action on the grid (add, delete or update) is executed. The changes are then committed to the databases automatically, because of the link between the DataGrid and the database via ADO.NET.

Screen for solving the transportation problem by launching the AMPL module

The purpose of this screen is to enable the user to launch and execute the AMPL LP module and solve the transportation problem. This form is a child form, whose parent is the main form described earlier. When the AMPL module is launched it reads the data and parameters for the problem from the relational database, via the OBBC connection established between the AMPL model and the relational database, as explained in Chapter 2. After the transportation problem is solved, the optimum transportation costs are displayed in a grid for each origin-destination-product combination. Fig 9 displays the screen in design mode.

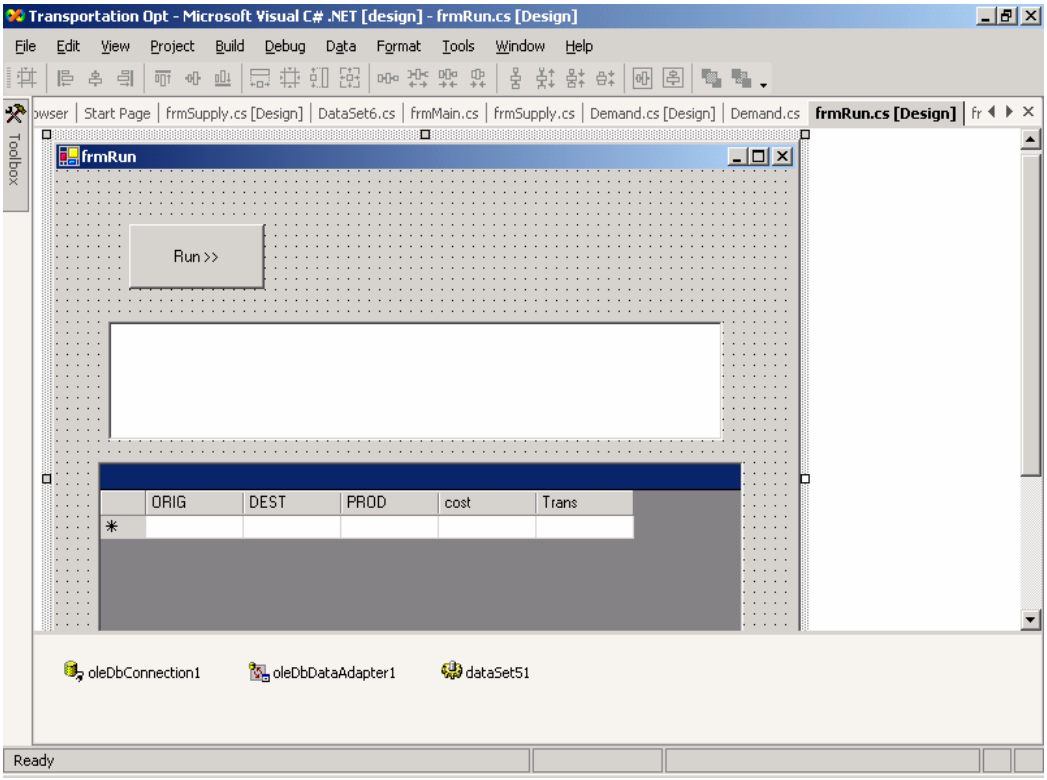


Figure 9 Screen for solving the transportation problem by launching the AMPL module

This screen contains a button to launch the AMPL module to solve the transportation module. The AMPL module is launched in the form of a Microsoft batch (*.bat) file by using the ProcessStartInfo class within the .NET framework. ProcessStartInfo is used in conjunction with the Process component. A Process component provides access to a process that is running on a computer. The Process component is a useful tool for starting, stopping, controlling, and monitoring applications[3]. Using ProcessStartInfo enables the user to launch and run the batch file in the local operation system. The path variable containing the full (and not relative) path of the application must be specified in the double quotes. After the process ends, the output of the process started by the ProcessStartInfo command is redirected to a field on the

screen. This informs the user if the AMPL LP module found a feasible solution for the transportation problem. If a feasible solution is found, the optimum transportation cost is displayed in the results box.

When the user launches the screen for the first time, the grid in the screen displays the transportation cost for each node (origin-destination-product combination) in a grid. The optimum transportation amount for each node is initialized to zero. After the transportation problem is solved by clicking the solve button (as described in the previous paragraph), the grid is refreshed and the optimum transportation amounts for each node are displayed in the grid. The logic for launching the process and refreshing the grid are displayed in the code snippet shown in figure 10 below.

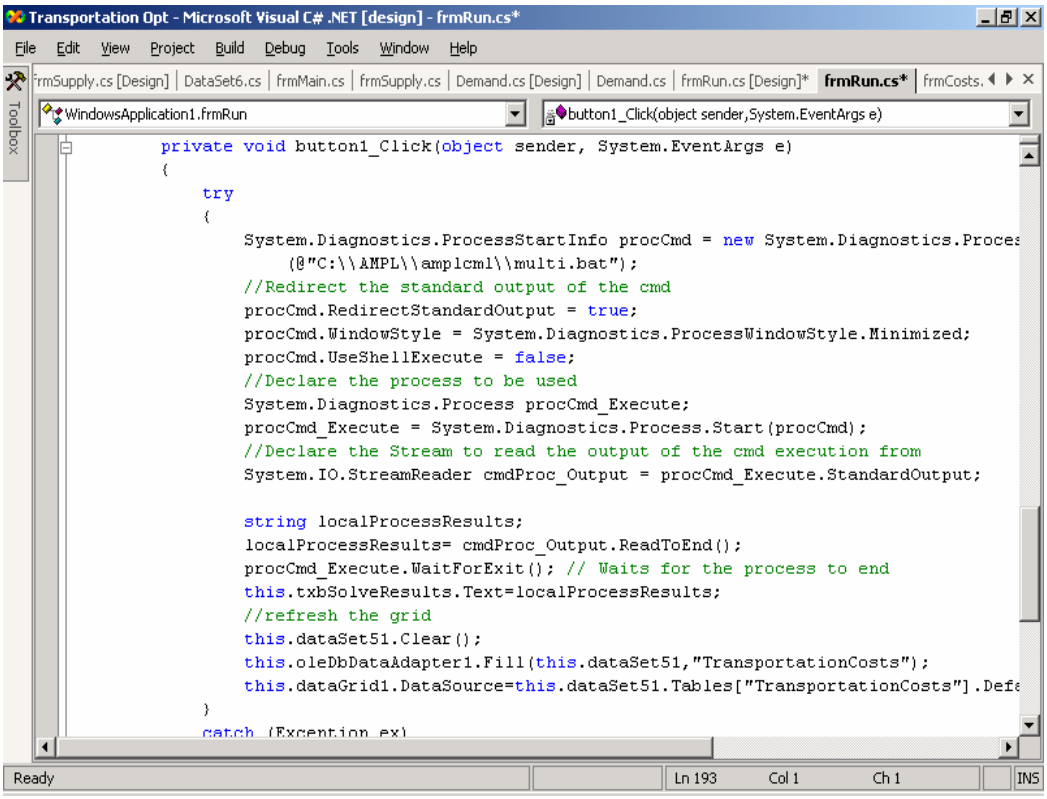


Figure 10 Code snippet for the logic for launching the process and refreshing the grid

Chapter Five: Testing and validating the solution

This chapter describes the testing of the different modules developed as part of this project as a complete solution. The testing and validation process involved using the GUI and the AMPL LP module to solve a transportation problem involving multiple origins, destinations and products.

In the problem solved as part of this project, a company has 3 product types – bands, coils and plates.

Data for the Transportation problem

The list of origins (manufacturing plants) and the respective amounts of each product which they supply are given below.

Table 1 Supply

	Gary	Cleveland	Pittsburgh
Bands	400	700	800
Coils	800	1600	1800
Plate	200	300	300

Table 2 Destinations (distribution centers) and the respective demand levels

	Franklin	Detroit	Langley	Winchester	Seattle	Fremont	LA
Bands	300	300	100	75	650	225	250
Coils	500	750	400	250	950	850	500
Plate	100	100	0	50	200	100	250

Table 3 The transportation costs for each origin destination pair is displayed below per product.

Bands	Franklin	Detroit	Langley	Winchester	Seattle	Fremont	LA
Gary	30	10	8	10	11	71	6
Cleveland	22	7	10	7	21	82	13
Pittsburgh	19	11	12	10	25	83	15

Coils	Franklin	Detroit	Langley	Winchester	Seattle	Fremont	LA
Gary	39	14	11	14	16	82	8
Cleveland	27	9	12	9	26	95	17
Pittsburgh	24	14	17	13	28	99	20

Plate	Franklin	Detroit	Langley	Winchester	Seattle	Fremont	LA
Gary	41	15	12	16	17	86	8
Cleveland	29	9	13	9	28	99	18
Pittsburgh	26	14	17	13	31	104	20

Testing the graphical user interface (GUI)

The GUI was tested by a) using it to enter the data for the transportation problem and b) solving the transportation problem by launching the AMPL module. The data for the transportation problem were entered using the various screens mentioned in the previous chapter. This section will demonstrate the functionality of each of these screens.

Testing the main screen

The main screen is the first one to be displayed when launching the application. It has menu options to launch the various other screens. First, the menu option "nodes" was chosen to launch the screen to enter the origins, destinations and products. Figure 11 displays the form with this menu option chosen. The menu in the main form was used to launch other forms to enter other parameters and variables for the LP model.

Testing the "nodes" screen

The "nodes" screen was launched using the nodes menu item in the main screen. It was used to enter origins, destinations and products, using list boxes. After entering the values, they were saved to the database, by clicking on the "Commit" button. Figure 12 shows the screen shot of the "nodes" screen, with the values for origins, destinations and products entered.

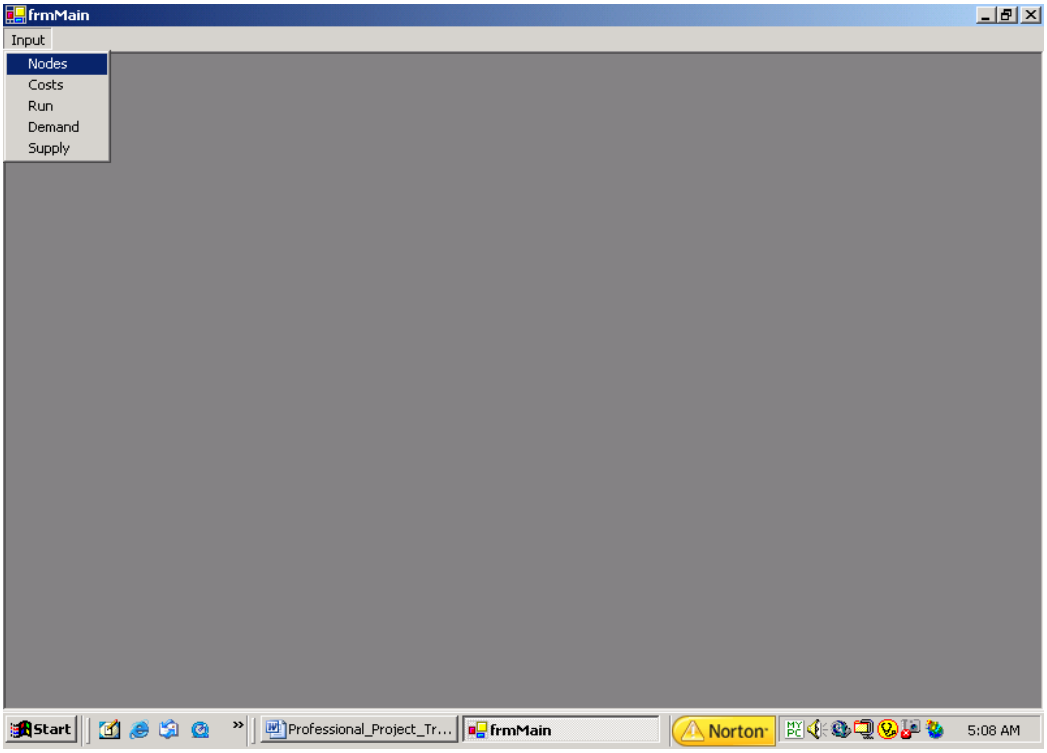


Figure 11 Menu screen

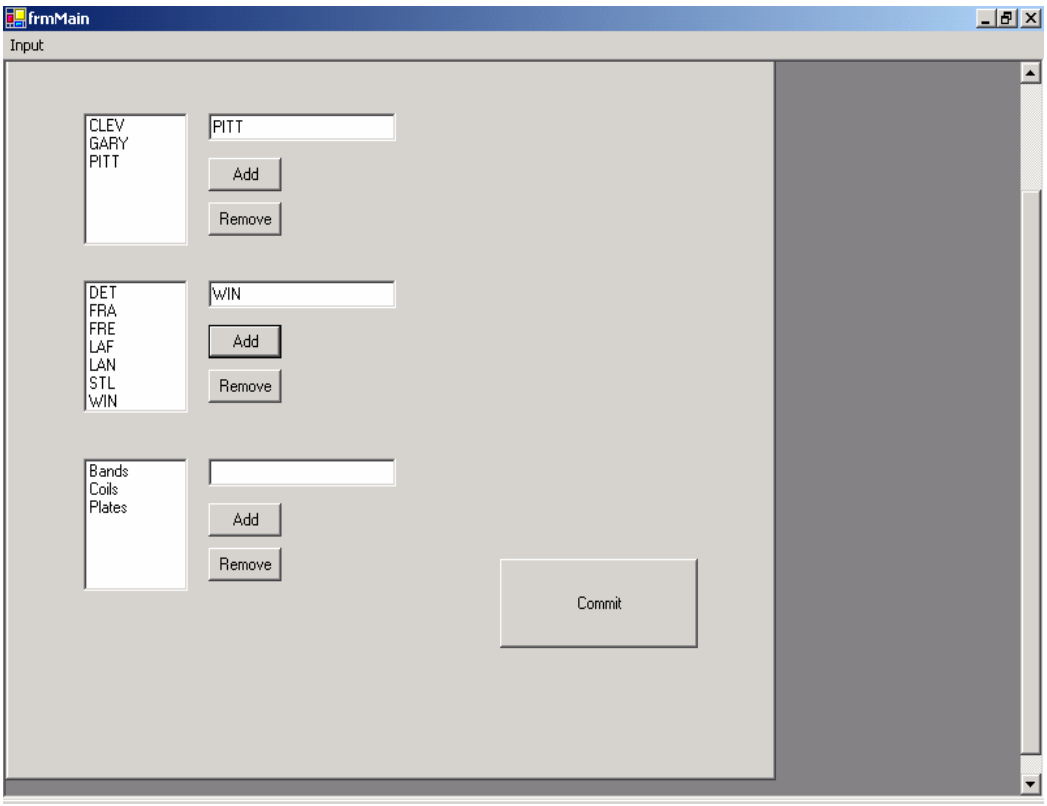


Figure 12 Nodes screen

Testing the Supply screen

As described in the previous chapter, the user can enter the supply from each origin using the grid on the screen. Figure 13 shows the screen with supply entered for each product from all the origins. The origin, product and supply information was entered in a grid line and the insert button was used to save that information in the database. When the screen was closed and launched again, the information previously entered was displayed in the grid. The supply for any product from an origin can be changed by entering the information in the grid and clicking on the update button to save the changes to the relational database.

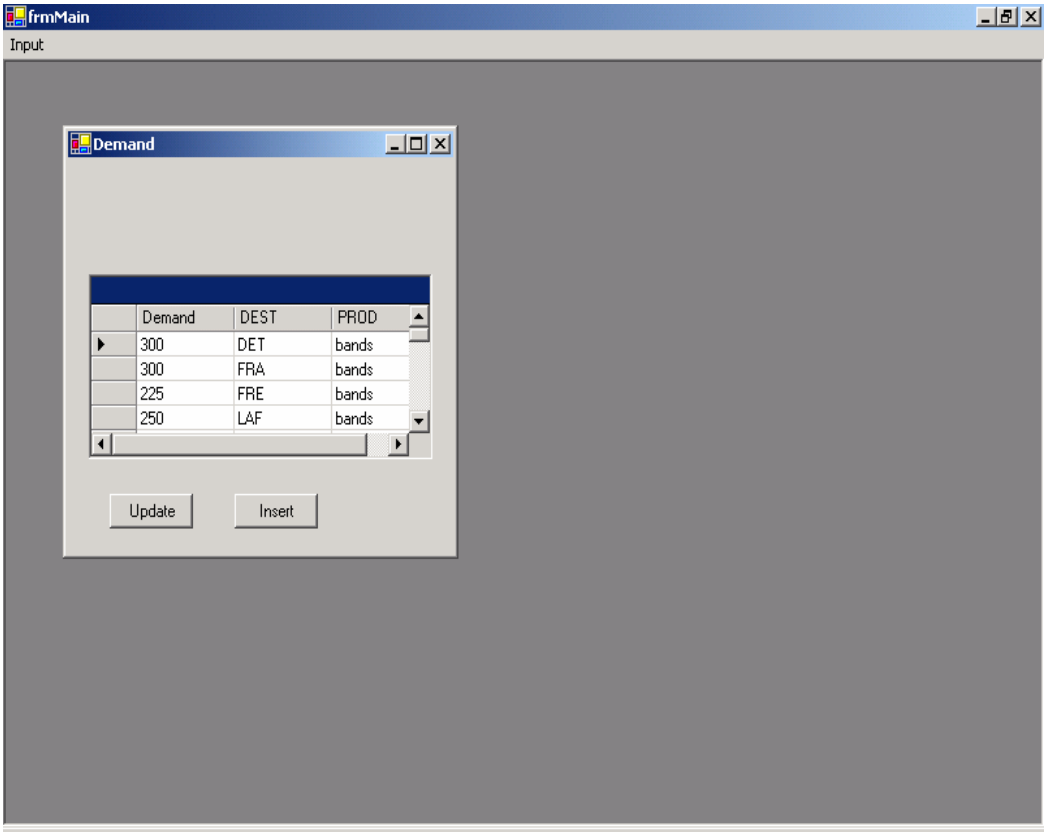


Figure 13 Supply screen

Testing the Demand screen

The user can enter the demand at each destination, using the grid on this screen. Figure 14 displays the screen with demand information entered for each product at all the destinations. The origin, product and demand information was entered in a grid line and the insert button was used to save that information in the database. When the screen was closed and launched again, the information previously entered was displayed in the grid. The demand for any product at a destination can be changed by entering the information in the grid and clicking on the update button to save the changes to the demand table in the relational database.

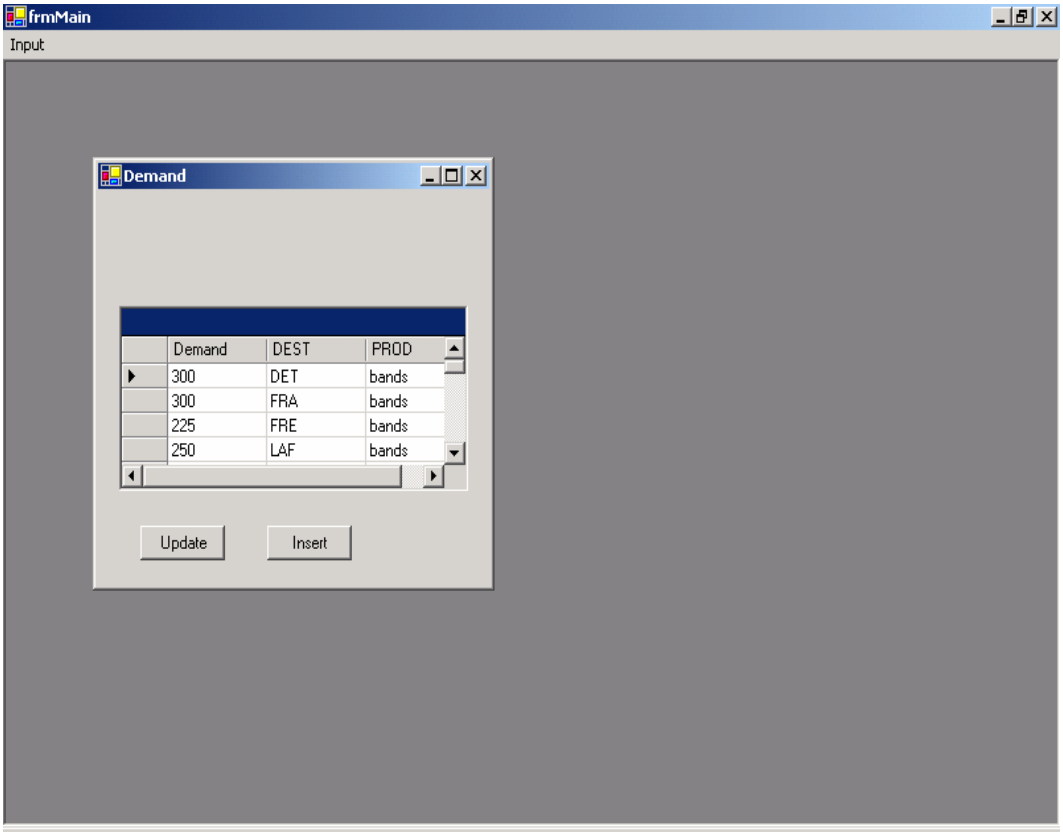


Figure 14 Demand screen

Testing the Costs Screen

Transportation costs were entered for each origin, destination and product combination (node). The user can enter the transportation cost for each node using the grid on this screen. Figure 15 displays the screen with cost information entered for each node. The information was entered in a grid line and the insert button was used to save that information to the database. When the screen was closed and launched again, the information previously entered was displayed in the grid. The transportation cost for any node can be changed by entering the information in the grid and clicking on the update button to save the changes to the TransportationCosts table in the relational database.

This table also contains a field to store the optimum amount to be transported along each node. This field is blank until the LP model is solved.

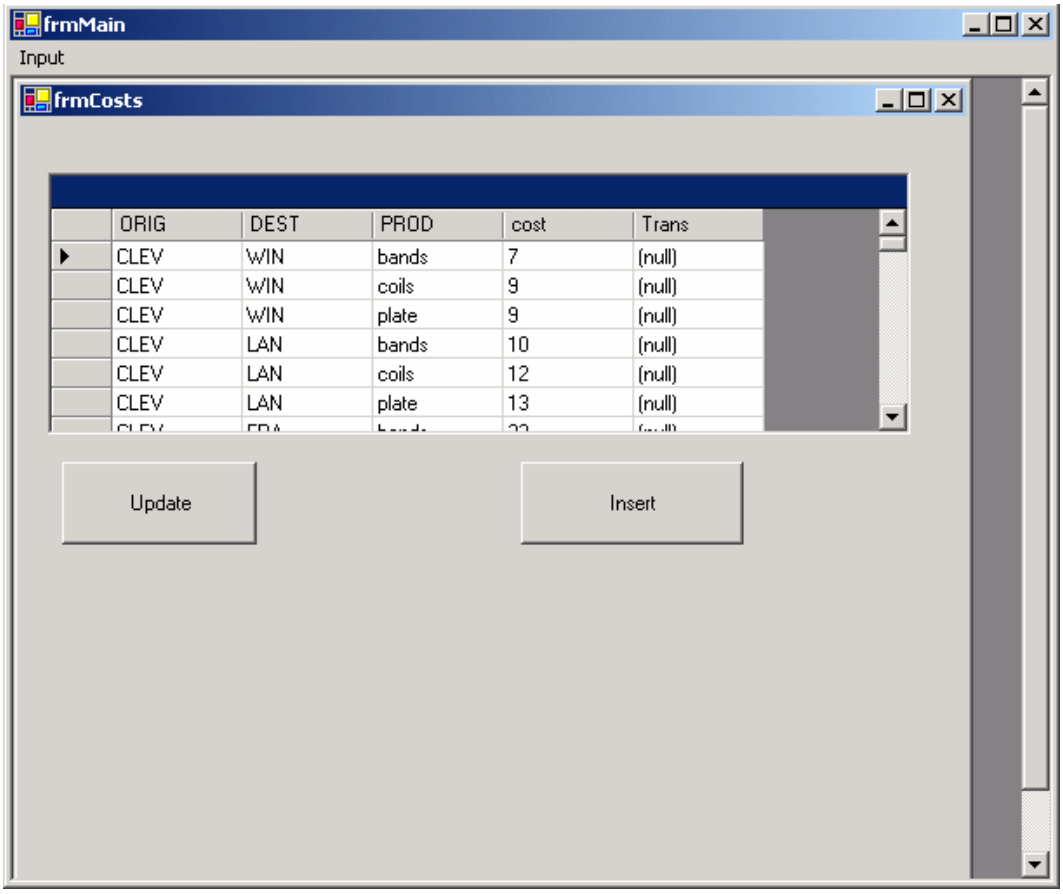
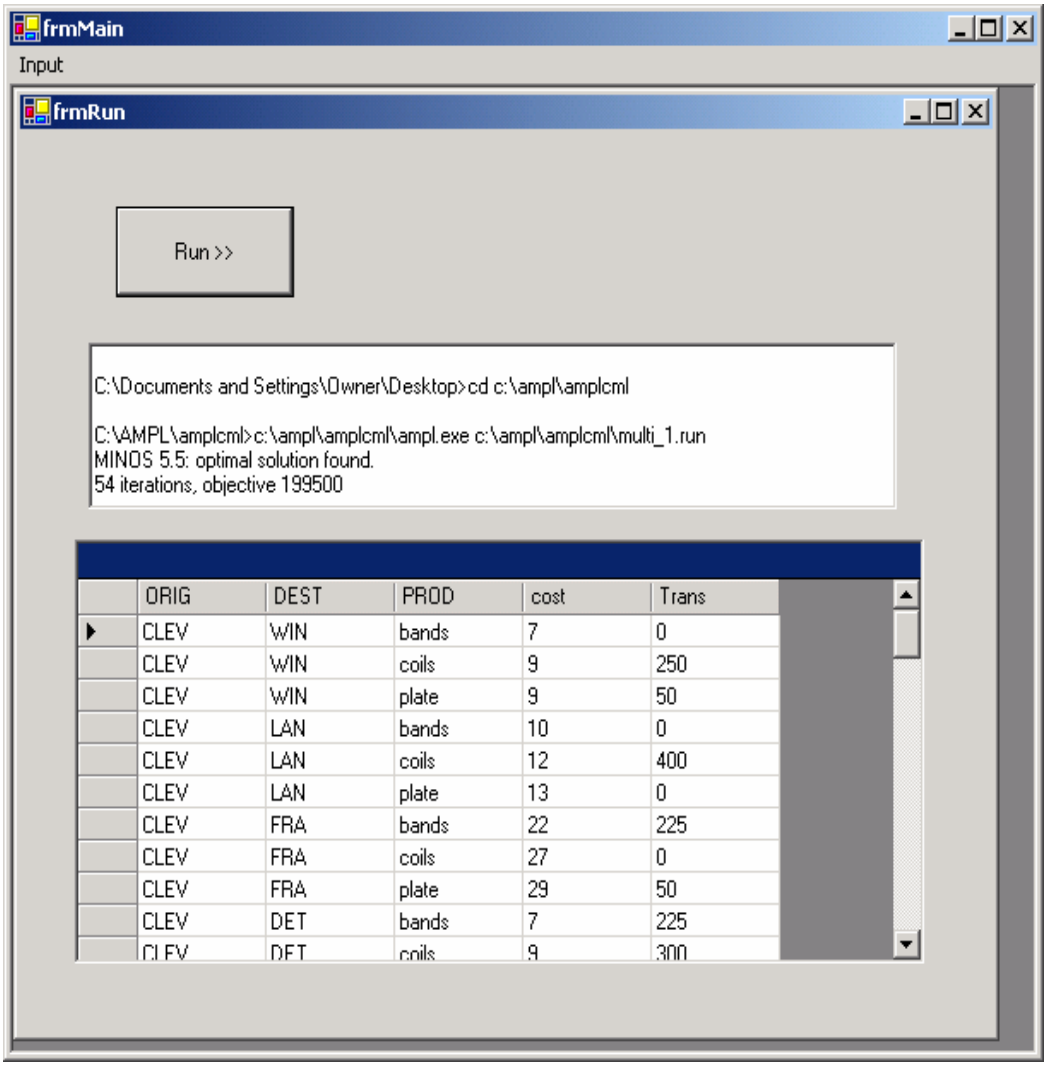


Figure 15 Costs screen

Testing the screen to solve the transportation problem

The screen to solve the transportation problem is used to launch the AMPL LP module. This screen also displays the optimum transportation costs for each node, required to minimize the total transportation costs after solving the problem. Clicking on the solve button launches the AMPL module. The AMPL module reads all the data and parameters entered in the previously described screens, using the ODBC interface built between the AMPL module and the relational database. Figure 16 displays the “solve”

screen after the problem was solved and the minimum total transportation cost was determined.



ORIG	DEST	PROD	cost	Trans
CLEV	WIN	bands	7	0
CLEV	WIN	coils	9	250
CLEV	WIN	plate	9	50
CLEV	LAN	bands	10	0
CLEV	LAN	coils	12	400
CLEV	LAN	plate	13	0
CLEV	FRA	bands	22	225
CLEV	FRA	coils	27	0
CLEV	FRA	plate	29	50
CLEV	DET	bands	7	225
CLEV	DET	coils	9	300

Figure 16 Solve screen showing a run

The optimum total transportation cost is $199500. The optimum transportation amounts along each node were written back to the TransportationCosts table and also displayed in the grid.

Table 4 The optimum transportation amounts

Origin	Destination	Product	cost	Amount Transported
CLEV	WIN	bands	7	0
CLEV	WIN	coils	9	250
CLEV	WIN	plate	9	50
CLEV	LAN	bands	10	0
CLEV	LAN	coils	12	400
CLEV	LAN	plate	13	0
CLEV	FRA	bands	22	225
CLEV	FRA	coils	27	0
CLEV	FRA	plate	29	50
CLEV	DET	bands	7	225
CLEV	DET	coils	9	300
CLEV	DET	plate	9	100
CLEV	STL	bands	21	250
CLEV	STL	coils	26	300
CLEV	STL	plate	28	0
CLEV	FRE	bands	82	0
CLEV	FRE	coils	95	125
CLEV	FRE	plate	99	100
CLEV	LAF	bands	13	0
CLEV	LAF	coils	17	225

CLEV	LAF	plate	18	0
GARY	WIN	bands	10	0
GARY	WIN	coils	14	0
GARY	WIN	plate	16	0
GARY	LAN	bands	8	0
GARY	LAN	coils	11	0
GARY	LAN	plate	12	0
GARY	FRA	bands	30	0
GARY	FRA	coils	39	0
GARY	FRA	plate	41	0
GARY	DET	bands	10	0
GARY	DET	coils	14	0
GARY	DET	plate	15	0
GARY	STL	bands	11	400
GARY	STL	coils	16	25
GARY	STL	plate	17	200
GARY	FRE	bands	71	0
GARY	FRE	coils	82	625
GARY	FRE	plate	86	0
GARY	LAF	bands	6	0
GARY	LAF	coils	8	150
GARY	LAF	plate	8	0
PITT	WIN	bands	10	75

PITT	WIN	coils	13	0
PITT	WIN	plate	13	0
PITT	LAN	bands	12	100
PITT	LAN	coils	17	0
PITT	LAN	plate	17	0
PITT	FRA	bands	19	75
PITT	FRA	coils	24	500
PITT	FRA	plate	26	50
PITT	DET	bands	11	75
PITT	DET	coils	14	450
PITT	DET	plate	14	0
PITT	STL	bands	25	0
PITT	STL	coils	28	625
PITT	STL	plate	31	0
PITT	FRE	bands	83	225
PITT	FRE	coils	99	100
PITT	FRE	plate	104	0
PITT	LAF	bands	15	250
PITT	LAF	coils	20	125
PITT	LAF	plate	20	250

Chapter Six: Lessons Learned and Conclusion

Lessons learned

The main lesson learned from this project was determining how solve a hypothetical transportation problem related to minimizing total transportation cost, by developing disparate software modules and interfacing them. This included the following

a) Modeling the hypothetical transportation problem as a linear problem. This project involved using mathematical concepts to model the problem as a linear problem with an objective function and constraints. Therefore, practical knowledge of using mathematical techniques to solve business problems was gained. Although the problem solved was a hypothetical one, it could be modified in future revisions to model a real world problem related to transporting finished goods from origins to destinations.

b) Developing a software model for the linear problem using the AMPL modeling software and solving it. In the process, knowledge of using AMPL modeling commands and programming constructs was gained. This knowledge, along with the knowledge of linear programming, could be used for modeling other real world problems related to optimization of processes in business areas of distribution and manufacturing.

c) Interfacing the AMPL model with a relational database to store the data for the linear problem. The advantages of using a relational database for storing data and parameters for a linear program were explained in chapter 4. This knowledge could be useful, especially when modeling larger problems with hundreds of variables and parameters. Knowledge was gained of applying relational database

principles to organize the LP model data into normalized tables with simple and compound primary keys and foreign keys.

The AMPL commands to interface the relational database had to be learnt as part of this project. Then they had to be executed in the correct order to ensure that data dependencies were taken into account when entering the data for the LP model.

Designing and developing the various screens for entering and maintaining data related to the AMPL model.

d) Adding the various controls, such as List boxes, DataGrids to the screens to enable data entry and maintenance.
e) Adding event driven logic to the controls to ensure that the appropriate actions are taken when the user tries to enter data using the controls
f) Interfacing the GUI with the relational database using ADO.NET interface within the .NET framework.

The knowledge gained by accomplishing tasks d, e, f and g was in the area of designing and developing an event-driven Windows application using the .NET framework. In this process, knowledge was gained of developing applications using an object-oriented programming language (C#). Knowledge was also gained of advanced features of the C# programming language, such as DataGrid, ADO.NET and interfacing with another program (AMPL module). This knowledge would be very useful in future enhancements of this project, which might involve modeling larger problems with higher number of variables and constraints.

Linear programming techniques have been used to solve business problems in the past. However, this project was used to demonstrate how computer modeling techniques, along with modern software tools, such as object oriented languages/frameworks and relational databases, could be used to quantify and solve business problems. By utilizing a combination of modern software tools, this project delivered a solution that is efficient and amenable to future enhancements.

Was this project a success?

This project was a success because a prototype module for solving a hypothetical transportation problem was developed. This module consisted of different components viz., a LP component, a GUI component and a relational database. The transportation problem was first modeled as a linear problem. Then it was modeled using a linear programming software language. A user-friendly interface was developed for the linear model to enable data to be entered for the LP model and solving it. The prototype could be used as a basis for solving larger problems. The project can also be considered a success because of all the lessons learned, which have been mentioned in the earlier section.

Challenges faced during the project

One of the major challenges faced during the project was in developing the interface between the AMPL model and the relational database. The commands for interfacing the AMPL LP model and the relational database had to be configured to map the variables and data in the relational database. Since the parameters and variables in the LP model had multiple indexes, they had to be mapped to tables with composite

primary keys. AMPL commands had to be carefully researched to achieve this. Another major challenge faced was in the area of developing the GUI screens for the module, especially in the area of grid processing. Here, the challenge was to establish the connection between the grid fields and related tables in the relational database to ensure that the grid was loaded with records at the time of launching the form. A significant amount of time was also spent in trying to add logic to add and delete records from the DataGrid. The final challenging aspect of the GUI development was adding ListBoxes and related logic for adding and removing items.

Directions for further improvements to this project

This software module, developed as part of this project, could be further enhanced in the following ways in future projects to provide greater feature/functionality to users.

a) The transportation problem selected could be from the real world, i.e., a case study could be made of the optimization problems faced by the shipping department of an actual firm and modeled using the techniques used in this project.
b) The graphical user interface developed for this project could be further enhanced to provide greater visual aids to model the transportation problem.
c) Formal object-oriented techniques could be used in designing and developing the GUI for this project.
d) While the GUI developed in this project was a standalone application, it could be enhanced to interface other business systems used in a business, such as the ERP system, data warehouse or other network optimization systems.

List of References

Dantzig, G. B. (1963). *Linear programming and extensions.* Princeton, NJ: Princeton University Press.

Microsoft Solution Development Network documentation. ©Microsoft Corporation 2006.

Fourer, R., Gay, D. M., & Kernigan, B. W. (2003). *AMPL – A modeling language for mathematical programming*. Brooks-Cole-Thomson Learning.

Watson, K. (2003). *Beginning C#.* Wiley Publishing Inc.

Tognazzini, B. (1991). *Tog On Interface*. Addison-Wesley.

Laurel, B. (1991). *The Art of Human Computer Interface Design*. Addison-Wesley.