AUTOMATED TRANSPORTATION MODEL USING NORTH WEST CORNER METHOD

BY

ADIGO Y. ATTAH
PGD/MCPS/1183/05/06

DEPARTMENT OF MATHS AND COMPUTER SCIENCE
FEDERAL UNIVERSITY OF TECHNOLOGY
MINNA, NIGER STATE

MARCH, 2007

AUTOMATED TRANSPORTATION MODEL USING NORTH WEST CORNER METHOD

BY

ADIGO Y. ATTAH
PGD/MCPS/1183/05/06

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF POST GRADUATE DIPLOMA (PGD) DEGREE IN COMPUTER SCIENCE

DEPARTMENT OF MATHS AND COMPUTER SCIENCE
FEDERAL UNIVERSITY OF TECHNOLOGY
MINNA, NIGER STATE

MARCH, 2007

CERTIFICATION

I certify that this project was carried out by Adigo Y. Attah (PGD/MPS/1183/05/06) of the Department of Computer Science, Federal University of Technology, Minna, under my supervision and approval.

Dr. Y.M. Aiyesimi	Date
Dr. N.I. Akinwade	Date
External Examiner	Date

DEDICATION

To God almighty who has given me the strength to complete this work. God is Great!

ACKNOWLEDGMENT

The researching and writing of a project of this nature could hardly be a one person endeavour. A lot of individuals assisted me in various ways and at different stages of this project.

Therefore it becomes difficult to enumerate and thank all of them but a few whose contributions proved to be valuable to the production of this work.

I express my profound gratitude to God Almighty, who has given me the strength to complete this work.

With every sense of sincerity, I most recognize and acknowledge the support I received from my immediate family especially my parents, brothers and sisters, for all their sacrifices support and understanding. I deeply appreciate all they did for me by way of moral, financial and prayerful support.

My words of appreciation go to my supervisor, Dr. Aiyesimi for his efforts, and his corrections here and there.

Further, I am indebted to Mr. Jiya, His contribution towards this project has proved to be valuable.

I also owe much thanks to the others lecturers of the computer science department.

Finally, thanks are due to Emeka Awagu, for all his efforts.

3.4	Sample and sampling techniques	14
3.5	Instrument for data collection	20
3.6	Methods of data Collection	22

CHAPTER FOUR - SOFTWARE DESIGN AND IMPLEMENTATION

4.1	Construction of the Problem	25
4.2	Analysis of the Northwest Corner/Stepping Stone Method	27
4.3	Implementing the transportation algorithm	28
4.4	Description of statistical tool used	29
4.5	The least square method of estimating trend	29
4.6	Statement of Hypothesis	30
4.7	Uses of the analysis of time series	30
4.8	Method of time series analysis	31
4.9	Validity checks of the model	34
4.10	Reliable prediction of System performance	35
4.11	System flow chart	36
4.12	Program flow chart	37

CHAPTER FIVE – CONCLUSIONS AND RECOMMENDATIONS

5.1	Conclusions	40
5.2	Recommendations	41
	References	
	Program Listing	

LIST OF TABLES

2.1 Problems of transportation

2.2 Northwest corners/stepping stone method

4.1 Initial solution result

4.2 Optimal solution

4.2.1 Manufacturer shipping schedule

4.8.1 Showing least square method using time series

4.8.2 Showing chi- square calculated

4.10.1 Comparing and examining the model for system performance

CHAPTER 1 – INTRODUCTION.

Transportation can be defined as the conveyance or the movement of goods and materials from one place to another. Transportation has been a vital activity of man over the ages of development. The need for the movement or conveyance of goods arises from the fact that they have to be taken to where desired after being produced. The transportation of goods arises from the need to satisfy basic needs be it social, cultural, or business.

In Nigeria, and most other African countries, transportation is operated in four main areas namely; Air, Road, Rail and Water. This research is concerned with the road transportation. All of this transportation means serve a common purpose and that is movement of people and goods from one place to another. Transportation is faced with various types of problems which affect the movement of people and goods from one location to another. The road transport system is faced with numerous problems namely: High fares, scarcity of fuel which leads to the delay of distributing goods from the source to their various destinations, disregard of traffic rules by road users which lead to accidents, armed robbery, lack of good roads, lack of traffic light and traffic facilities, traffic accident analysis. All of these lead to the road transportation problem in the country.

Transportation improvement primarily focuses on overcoming a common obstacle, which is the distance between places. This is seen as a barrier in terms of time and cost and will solve to an extent the problem of safety, convenience and comfort. In effect, any improvement in transportation that leads to reduction in either time or cost or both overcomes the major barrier presented by distance. More often than not, transportation routes are straight lines between two

places with the exception of mountain or water bodies, posing as obstacles to a straight line.

1.1 – Background Of The Study

Transportation model is concerned basically with the transportation of homogenous products or service from a number of sources, often called origins that have specific quantities of supply to a number of destinations with certain quantities of demand.

Transportation problems are often used in transportation planning, for instance, where goods are at a warehouse, the problem might be to assign warehouses as the source, the customers as destinations and the costs to represent the per unit transportation cost.

Coca-Cola is the most successful foreign company transacting business, especially in Asia, according to an expatriate business executive in a recent survey published in Hong Kong in 2005 by a pharmacist Dr. John Styth Pemberton in his hour Africa Georgia USA.

Dr. Pemberton's partner, Mr. Frank M. Robinson gave coca-cola its fancy name that distinguishes the famous trademark. Coca-cola's content remains secret has it has for over a hundred years. The formula known as "MERCHANDISE 7X", is kept in a special security vault in a bank in United States.

Firstly, coca-cola is distributed in two different ways: Depot and Basic

- *Basic*

  This deals with sales trucks and how they are being loaded, according to their request from the customer. Their request depends on the sale root, they load on a daily basis after which a settlement sheet is printed for each truck. This settlement sheet records what the salesman has taken out and what he returns

with. A case where the salesman returns without 5 crates would indicate that he sold both minerals and bottles.

- *Depot*

  It deals with the distribution in different form unlike Basic, it deals with the daily stock position they distribute on daily basis and based on what the customer request, the produce way bill / invoice for goods transfer, which keeps track of the goods that have been transported to the depot and the name of the driver, number of truck, the storekeeper will also sign with time and date. He will also record the quantity, unit price and extended amount and when the goods is delivered to its destination. They receive would have to send an acknowledge letter to confirm that he/she has received the goods.

Manufacturers are the ones that produce the mineral in the company. In the production planning they have their own hours of production, which deals with the number of crates that will be produced in an hour e.g. 1008 per hour, to meet optimal solution they work harder so that they can produce 1200 per hour, the scheduling time stipulates the number of hours and cases expected in production figures.

1.2 – Statement Of The Problems

Transportation problems involve finding the cheapest way of shipping goods / products from certain sources to certain locations or destinations. The numerical label on a segment is the cost of shipping one unit of coca-cola truck along the route. Sometimes shipment route which are actually used are drawn. Cases abound where supply may be less that the demand or may exceed the demand. In this case the model is said to be unbalanced.

1.3 – Purpose Of The Study

- To design a model that will minimize the total transportation cost consequently maximizing the total transportation profit.
- To schedule vehicles for the distribution of goods to essential route with the aim of transporting goods from certain sources of supply to certain destinations at minimum cost. This would be done such that the transportation schedule would be worked out to minimize the transport cost by determining an initial basic feasible solution.
- To provide a means of determining a unique allocation that satisfies the demand and supply requirements simultaneously.
- To know the number of trips estimated in the trip generation phase for each traffic analysis, the zone is matched with a trip attracted to the zone by purpose though a zone to zone impedance matrix (time distance cost) in the trip distributed step.
- To investigate the full social and environmental cost of road freight.

1.4 – Significance / Motive Of Study

The transportation model helps this researcher to know the number of routes expected to be used where (m = number of rows, n = number of columns). It is usually summarized in tabular or matrix form. Columns represent the destination while rows represent the source. The transportation model is to determine, j, such that the total transportation cost is minimized. It mainly schedules vehicles for the distribution of goods / products from a certain source supply to a certain destination at a minimum cost. It also determines the number of units transported from source i to location j.

1.5 – Research Questions

- How do the activities of the product demanded at various destinations affect the quantities supplied from the source?
- How cost effective is the method the company uses in delivering their product or service?
- Is the optimal solution obtained in the method the company adopts in delivery of their product at various destinations?

1.6 – Scope Of Study

This research project is designed to examine or focus on the activities of transportation model distribution of goods from source to destination with the aim of achieving the optimal solution.

1.7 – Definition Of Terms

Automated: This is a system of processing, designed to extend the capacity of computer to perform certain tasks formerly done by humans, and control sequence of operation with human intervention.

Transportation: Is concerned basically with the movement of certain products from a source to it numerous localities or destinations.

Model: Is defined as an idealized or simplified representation of a real life situation, sometimes the system may already be in existence or an idea awaiting execution.

Route: Is a way, or course taken in getting from starting point to a destination, a round traveled in delivery, and selling or collecting goods. Or is a link or part that links the source and destination.

Distribution: Is the process whereby the products of the company is transported and supplied to its customer, i.e. moving of goods from one company to different sets of customer or people.

Destination: The place where the goods are supposed to reach i.e. the customer.

Turn Around Time: Is a time known to the coca-cola drivers, this means they are expected to return from distributing the products.

Manufacturer: These are the people that produce the products / goods.

Sales Representative: These are the people that follow the truck and distribute the product to the customers.

Stock: Is a document that contains the quantity of products needed by the customer.

Load Out: Is the number of crates that the sales man goes out with in a trip.

Load Out Return: Is the number of crates that the sales man returns with in a trip.

Overnight: This means that the trucks are being loaded according to their request.

Settlement Sheet: Is used to record what the salesman has taken out of the company and how many he has returned with, and the customer to whom the crates have been sold to has to sign to confirm that he/she has paid or not.

CHAPTER 2 – LITERATURE REVIEW

A Literature review is an essential part of the research work that will help the researcher achieve the desired goal. In any research work, there is a need to review a new related literature that is know and tested. Analysis of such existing literature or knowledge provides fumigation needed to build upon an existing system.

2.1 – The Basic Transportation Model Problem

The transportation model involves the manufacturers transporting a number of homogenous products from several warehouses (origin) to a number of refined stores (destination). Each store requires b_j by the j^{th} store a certain number of units of the product, while each warehouse a_i by the j^{th}, ils warehouse can supply up to a certain number amount. The cost of transporting a unit from the i^{th} origin to the j^{th} destination is C_{ij} and is known for all combination (I, J). The problem is determining the amount X_{ij} to be transported over to source (I, J), so as to minimize the total cost of transportation. The above problem can be expressed in a tabular form.

Table 2.1 – Destinations

	1	2	...	J	...	N	a_i
i	x_i	X_{12}		X_i		X_{in}	a_1
2	X_{21}	X_{22}		X_{21}		X_{2n}	a_2
.							
i	X_{ij}	X_{12}		X_i		X_{in}	a_n
.							
m	X_{m1}	X_{m2}		X_{mi}		X_{mn}	a_m
b_i	b_1	b_2		b_J		B_n	

The amount shipped from origin i to destination j is x_{ij}. The total shipped from origin i is $a > 0$ and the total received by destination j is $b = a$, we impose the restriction that the total amount shipped is equal to the total amount received i.e. $\sum a_i = \sum b_i$. The total cost of shipping x_{ij} unit is $c_{ij}*x_{ij}$ since negative shipment has no valid interpretation for the problem, one restriction is that each $x_{ij} \geq 0$. We can state the problem mathematically as follows:

Find the value of x_i which minimize the total cost of transportation

m, n

$\sum_{i=1}^{n} \sum_{j=1}^{m} c_i, x_i$, total cost of transportation

Supply to n $x_{ij} \leq a_i$ I=1 --------m (supply)

$\sum x_i \geq b_i$ J=1 ----------n (demand)

$X_{ii} \geq 0$ (non negative)

2.2 – The Northwest Corner / Stepping Stone Method

This method involves working diagonally across the routes from the top left hand to the bottom right hand (south-east) corner. In order words, one can start allocation from the upper left hand corner of the given transportation matrix, satisfying the individual destination requirements, and exhausting the origin capabilities on at a time. Then move towards the lower right corner until all the requirements are satisfied. When the northwest corner rule is applied, there is no need to pay attention to the relative cost of the different routes while making the first assignment (that is the initial solution)

Table 2.2

From \ To	A	B	C	Capacity
X	500 [8]	[10]	[7]	500
Y	100 [5]	150 [4]	[8]	250
Z	[4]	250 [8]	500 [4]	750
demand	600	400	500	1500

Total cost: ₦9,100.00

Check: m+n-1 = The required number of routes expected to be in use

3 + 3 – 1 = 5. (condition satisfied)

Note that the initial cost = E(unit allocated * unit cost in each occupied cell)

((500*8) + (100*5) + (150*4) + (250*8) + (500*4)) = 9,100.00.

Step 2 – Test table 1 for optimality (iteration 1) using the stepping stone method procedure.

Stepping stone method is based on evaluating the empty cells by carefully stepping on the occupied cell (clockwise or anti-clockwise). If an empty cell is stepped on, which is being evaluated, has a positive actual cost, while the next occupied cell which can make us turn leftward, rightward, up-ward or down-ward take a negative actual cost. The next turn interchanges sign.

XB = 10-4+5-8 = 3 (Additional cost of N3/unit).

XC = 7-4+8-4+5-8 = 4 (Additional cost of N4/unit).

YC = 8-4+8-4 = 8. (Additional cost of N8/unit).

ZA = 6-5+4-8 = -3 (Additional saving of N3/unit).

2.3 – Analysis Of the Existing System

In this analysis, this research attempts a review of the existing system to give an exposition of both input requirement and the output requirement or expectation. In order to appreciate where we are heading to, the system in review is the Nigerian Bottling Company Coca-Cola Kaduna Plc. The system is manual, using word processing and spread sheet packages for manipulation of text and calculation. This is a very tedious process to be used in continuous repetition of the process in creating the structure for record reports and in storing data information. In order to maintain its position with the present technology and to win over completeness and challenges with other producing industries, the company has to improve in its technology by fully computerizing all its systems in the company.

2.4 – Comparative Analysis Of The Transportation Model

Transportation model is classified into three types:

- The Northwest Corner Method Rule.
- The Least Cost Method.
- The Vogel's approximation Method (VAM).

In all these transportation models, we can find the following solution

1 Finding an initial solution, which is feasible from the point of availabilities and requirement of sources

2 Examine the solution of optimality and examine if an improved transportation schedule lower cost is possible.

3 Repeat step (ii) until no further improvement is feasible.

2.5 – Problem Of The Existing System

The manual system of computing distribution of goods gives room for inefficiency, time wastage, profit manipulation, misleading and / or unreliable reports. Below are some of the major setbacks of the manual distribution process:

- Humans often have an inherent dislike to work. There is a loss of concentration and routine work due to one thing or the other. Delayed information retrieval among other things constitutes unnecessary decision.
- Profit manipulation end of year purchasing policy can be used to manipulate profit.
- Lack of security departments: routine documents are not properly secured and are exposed to all sorts of hazard.

2.6 – Solution To The Existing Problem

The major objective of the project is to design software that can be implemented to alleviate the problem associated with the manual system of computing distributed system. It will modify the existing system, which in most cases has led to poor report processing. The implementation of the automated system will among other things give rise to timely decision making, accurate computation of optimum quality of goods distributed as required.

CHAPTER 3 – DESIGN PROCEDURE

Research encompasses activities that increase the sum of human knowledge [OECD Definition].

Research and experimental development comprises:

- creative work undertaken on a systematic basis in order to increase the stock of knowledge, including knowledge of humanity, culture and society, and the use of this stock of knowledge to devise new applications; [OECD Definition]

- any activity classified as research and experimental development is characterised by originality; it should have investigation as a primary objective and should have the potential to produce results that are sufficiently general for humanity's stock of knowledge (theoretical and/or practical) to be recognisably increased. Most higher education research work would qualify as research and experimental development. [DEST 2002, HERDC Specifications for the collection of 2002 data]

Research carries with it a professional and ethical responsibility to disseminate and apply the results of research activity and to conduct research in a manner consistent with the Statement and Guidelines on Research Practice. An essential characteristic is that it leads to publicly verifiable outcomes which are open to peer appraisal.

The complementary activity of scholarship refers to possession of an extensive and profound knowledge of an academic discipline and the analysis and interpretation of existing knowledge aimed at improving, through teaching or by other means of communication, the depth of human understanding. This chapter introduces various design

techniques, instrument for data collection, instrument for data analysis and administration of instrument as may be necessary to achieve the design goal.

3.1 – Research Design

Research includes pure basic research, strategic basic research, applied research and experimental development [ABS 1998, Australian Standard Research Classification]

Pure basic research is experimental and theoretical work undertaken to acquire new knowledge without looking for long-term benefits other than the advancement of knowledge.

Strategic basic research is experimental and theoretical work undertaken to acquire new knowledge directed into specified broad areas in the expectation of useful discoveries. It provides the broad base of knowledge necessary for the solution of recognized practical problems.

Applied research is original work undertaken primarily to acquire new knowledge with a specific application in view. It is undertaken either to determine possible uses for the findings of basic research or to determine new ways of achieving some specific and predetermined objectives.

Experimental development is systematic work, using existing knowledge gained from research or practical experience that is directed to producing new materials, products or devices, to installing new processes, systems and services, or to improving substantially those already produced or installed.

3.2 – Area Of Study

The area of study is the transportation department of Nigerian Bottling Company, Coca-Cola Kaduna, and the study is basically on transportation model of the company's sales and distributing activities. The study focuses on the analysis of the distribution of three major products of the company.

3.3 – Population Of The Study

In the cause of collecting data for this project, 63 questionnaires were given out to the company staff of the company. Out of these, 57 people responded, 11 of which were senior staff. In addition to this, one principle staff and distribution manager were interviewed. Also the researcher scrutinized distribution sales journal and annual distribution sales reports.

3.4 – Sample and Sampling Techniques

If you survey every person or a whole set of units in a population you are taking a census. However, this method is often impracticable; as it's often very costly in terms of time and money. For example, a survey that asks complicated questions may need to use trained interviewers to ensure questions are understood. This may be too expensive if every person in the population is to be included.

Sometimes taking a census can be impossible. For example, a car manufacturer might want to test the strength of cars being produced. Obviously, each car could not be crash tested to determine its strength!

To overcome these problems, samples are taken from populations, and estimates made about the total population based on information

derived from the sample. A sample must be large enough to give a good representation of the population, but small enough to be manageable. In this section the two major types of sampling, random and non-random, will be examined.

RANDOM SAMPLING

In random sampling, all items have some chance of selection that can be calculated. Random sampling technique ensures that bias is not introduced regarding who is included in the survey. Five common random sampling techniques are:

- Simple random sampling,
- Systematic sampling,
- Stratified sampling,
- Cluster sampling, and
- Multi-stage sampling.

SIMPLE RANDOM SAMPLING

With simple random sampling, each item in a population has an equal chance of inclusion in the sample. For example, each name in a telephone book could be numbered sequentially. If the sample size was to include 2,000 people, then 2,000 numbers could be randomly generated by computer or numbers could be picked out of a hat. These numbers could then be matched to names in the telephone book, thereby providing a list of 2,000 people.

A Tattslotto draw is a good example of simple random sampling. A sample of 6 numbers is randomly generated from a population of 45, with each number having an equal chance of being selected.

The advantage of simple random sampling is that it is simple and easy to apply when small populations are involved. However, because every person or item in a population has to be listed before the

corresponding random numbers can be read, this method is very cumbersome to use for large populations.

SYSTEMATIC SAMPLING

Systematic sampling, sometimes called interval sampling, means that there is a gap, or interval, between each selection. This method is often used in industry, where an item is selected for testing from a production line (say, every fifteen minutes) to ensure that machines and equipment are working to specification.

Alternatively, the manufacturer might decide to select every 20th item on a production line to test for defects and quality. This technique requires the first item to be selected at random as a starting point for testing and, thereafter, every 20th item is chosen.

This technique could also be used when questioning people in a sample survey. A market researcher might select every 10th person who enters a particular store, after selecting a person at random as a starting point; or interview occupants of every 5th house in a street, after selecting a house at random as a starting point.

It may be that a researcher wants to select a fixed size sample. In this case, it is first necessary to know the whole population size from which the sample is being selected. The appropriate sampling interval, I, is then calculated by dividing population size, N, by required sample size, n, as follows:

$I = N/n$

If a systematic sample of 500 students were to be carried out in a university with an enrolled population of 10,000, the sampling interval would be:

$I = N/n = 10{,}000/500 = 20$

Note: if is not a whole number, then it is rounded to the nearest whole number.

All students would be assigned sequential numbers. The starting point would be chosen by selecting a random number between 1 and 20. If

this number was 9, then the 9th student on the list of students would be selected along with every following 20th student. The sample of students would be those corresponding to student numbers 9, 29, 49, 69, ... 9929, 9949, 9969 and 9989.

The advantage of systematic sampling is that it is simpler to select one random number and then every 'Ith' (e.g. 20th) member on the list, than to select as many random numbers as sample size. It also gives a good spread right across the population. A disadvantage is that you may need a list to start with, if you wish to know your sample size and calculate your sampling interval.

STRATIFIED SAMPLING

A general problem with random sampling is that you could, by chance, miss out a particular group in the sample. However, if you form the population into groups, and sample from each group, you can make sure the sample is representative.

In stratified sampling, the population is divided into groups called strata. A sample is then drawn from within these strata. Some examples of strata commonly used by the ABS are States, Age and Sex. Other strata may be religion, academic ability or marital status.

The committee of a school of 1,000 students wishes to assess any reaction to the re-introduction of Pastoral Care into the school timetable. To ensure a representative sample of students from all year levels, the committee uses the stratified sampling technique.

In this case the strata are the year levels. Within each strata the committee selects a sample. So, in a sample of 100 students, all year levels would be included. The students in the sample would be selected using simple random sampling or systematic sampling within each strata

Stratification is most useful when the stratifying variables are simple to work with, easy to observe and closely related to the topic of the survey.

An important aspect of stratification is that it can be used to select more of one group than another. You may do this if you feel that responses are more likely to vary in one group than another. So, if you know everyone in one group has much the same value, you only need a small sample to get information for that group; whereas in another group, the values may differ widely and a bigger sample is needed.
If you want to combine group level information to get an answer for the whole population, you have to take account of what proportion you selected from each group (see 'Bias in Estimation' on page 186).

CLUSTER SAMPLING

It is sometimes expensive to spread your sample across the population as a whole. For example, travel can become expensive if you are using interviewers to travel between people spread all over the country. To reduce costs you may choose a cluster sampling technique.
Cluster sampling divides the population into groups, or clusters. A number of clusters are selected randomly to represent the population, and then all units within selected clusters are included in the sample. No units from non-selected clusters are included in the sample. They are represented by those from selected clusters. This differs from stratified sampling, where some units are selected from each group.
Examples of clusters may be factories, schools and geographic areas such as electoral sub-divisions. The selected clusters are then used to represent the population.
Suppose an organization wishes to find out which sports Final Year students are participating in across Nigeria. It would be too costly and

take too long to survey every student, or even some students from every school. Instead, 100 schools are randomly selected from all over Nigeria.

These schools are considered to be clusters. Then, every Year 11 student in these 100 schools is surveyed. In effect, students in the sample of 100 schools represent all Final Year students in Nigeria.

Cluster sampling has several advantages: reduced costs, simplified field work and administration is more convenient. Instead of having a sample scattered over the entire coverage area, the sample is more localised in relatively few centres (clusters).

Cluster sampling's disadvantage is that less accurate results are often obtained due to higher sampling error (see section Information - Problems with Using) than for simple random sampling with the same sample size. In the above example, you might expect to get more accurate estimates from randomly selecting students across all schools than from randomly selecting 100 schools and taking every student in those chosen.

MULTI-STAGE SAMPLING

Multi-stage sampling is like cluster sampling, but involves selecting a sample within each chosen cluster, rather than including all units in the cluster. Thus, multi-stage sampling involves selecting a sample in at least two stages. In the first stage, large groups or clusters are selected. These clusters are designed to contain more population units than are required for the final sample.

In the second stage, population units are chosen from selected clusters to derive a final sample. If more than two stages are used, the process of choosing population units within clusters continues until the final sample is achieved.

An example of multi-stage sampling is where, firstly, electoral sub-divisions (clusters) are sampled from a city or state. Secondly, blocks of houses are selected from within the electoral sub-divisions and, thirdly, individual houses are selected from within the selected blocks of houses.

The advantages of multi-stage sampling are convenience, economy and efficiency. Multi-stage sampling does not require a complete list of members in the target population, which greatly reduces sample preparation cost. The list of members is required only for those clusters used in the final stage. The main disadvantage of multi-stage sampling is the same as for cluster sampling: lower accuracy due to higher sampling error

3.5 – Instrument For Data Collection

To adequately address the research questions a variety of data collection methods and instruments will be used. These methods and instruments are not specific to any one question but provide data that when used in combinations will address the research questions. Examples of each of these instruments, with a description of data fields and administration, are available in Appendix B.

The data gathering instruments are:

• User Request Survey: a series of questions to be answered by users who ask to use an Experimental Title in storage. This instrument accompanies all Study Titles or copies of articles from Study Titles requested from Storage Locations.

- **Return Request Forms:** copies of standard forms completed by library staff or users when making a request for returns or copies from experimental Study Titles.

- **Experimental Use Worksheets:** quarterly activity reports from Storage Facilities summarizing, by Study Title, return requests and fulfillment details.

- **Control Use Worksheets:** quarterly activity reports from Control Sites summarizing, by Study Title, the number of uses as measured by reshelving counts.

- **Control Use Data Slips:** slips, inserted in each piece of a Study Title, for staff at Control Sites to record all reshelving instances by date.

- **Digital Use Reports:** transaction data from Providers for electronic access to Digital Study Titles from Experimental and Control Sites.

- **Phase I Cost Worksheets:** Financial reports from campuses and storage facilities on actual costs and staff effort in establishing the Experimental and Control Treatment.

- **Phase II Special Cost Studies:** a series of campus focused examinations of cost increases or decreases in subscriptions, licensing fees, binding, copying, handling, or other areas that may be impacted by the Experimental Treatment when applied over an extended time.

- **Formative Interviews:** interviews with select users to inform the design of a system wide survey on user preferences and behaviors.

- **Structured Interviews:** interviews with a broad range of users and library staff familiar with the Study Titles to examine preferences and behaviors with in the specific context of a particular Study Title.

• User Preference Surveys: a widely administered survey to users of both print and digital forms of journals, incorporating questions developed in the Formative Interviews on characteristics of journals, users, purpose of use and the user environment that influence the acceptability of digital or print.

• Comment Cards: cards at service locations in Control and Experimental Sites and a web site comment function offering users and staff a vehicle to contribute reactions, suggestions and ideas on the Experimental Treatment.

• Storage Reports: reports from storage facilities with measurements of volumes and linear feet used by each Study Title.

• Study Title Characteristics Worksheet: an examination of each Study Title in both digital and print form against a standard list of typography and content features drawn from literature and tested in the Formative Interviews.

• Consultation Process Survey: a survey administered to CMI Campus Liaisons on the effectiveness of the consultation and decision making processes needed to achieve the experimental condition.

3.6 – Methods Of Data Collection

Common methods are:

- Interviewing
- Questionnaires
- Observation
- Repertory Grids
- Concept Mapping
- Joint Application Design
- Contextual Design

Interviewing is the most widely used technique in requirements engineering.

Analysts interview future users of the system individually to find out

- what the present system does and
- what changes are needed.

The information gathered during the interviews enables the analysts to design a new system that will eliminate the shortcomings of the current one.

An alternative to interviewing individuals is the group interview conducted by an impartial leader.

A representative method is **Joint Application Design** (JAD), developed and taught by IBM.

The **advantages** of JAD are that

- the analysis phase of the life cycle is shortened and
- the specifications document produced are better accepted by the users.

Requirements analysis should not start until there is a clear statement of scope and objectives of the project.

The next phase is to build a **logical model** of the system (one which does not depend on details of implementation), usually by one of the following methods:

- **Interviewing** users to build a physical model of the present system, abstracting this to a logical model of the present system, and then incorporating desirable changes so as to create the logical model of the future system.
- **Organizing** a workshop with group methods such as JAD to create the logical model of the new system directly, without the intermediate steps of the physical and logical models of the present system.

The logical model of the new system becomes the requirements specification.

Typically, it contains:

- An **introduction** containing the scope, goals, and objectives of the system.
- **Data flow diagrams** depicting the workflow of the new system, including the clear identification of business transactions.
- A project **data model** (e.g. ER model).
- **Information needs**: which types of questions will be asked of the system and which data will support the answers.
- **Interfaces** with other systems.
- **Sample** report, screen, and form layouts, if required.
- **Operational information** such as processing frequencies and schedules, response time requirements, resource utilization constraints.
- Measurable **criteria** for judging the implementation.

The final test of the system, the **user acceptance test**, should be conducted as a benchmark against this document. Therefore, all those factors that will determine whether the users are satisfied with the system should be documented and agreed upon.

There may be a final step in the requirements analysis--to have the users and top management sign off on the specification document. (In most cases, if sufficient contact is maintained with the users during the work, this step is not necessary.)

We need to **review the present system** before designing a new one, because:

- it gives the design team a good grasp of the business problems
- it lets them understand the environment in which the new system will operate
- it helps make sure everything needed is considered
- it helps to relate the new design to the old one for ease of use
- it enhances cooperation between project team and users

CHAPTER 4 – SOFTWARE DESIGN AND IMPLEMENTATION

4.1 – Construction of the Problem

From table 2.2 in chapter 2, we show how the Northwest Corner Method/Stepping Stone Method is shown below.

Note

- Negative values computed implies that the route *Z has a saving of N3 per unit. Therefore it is considered wise if if ZA is put into use. Given the above position, we revisit table 2.2 and show the required movement, which will aid the re-allocation of units in Table 4.1
- The maximum quantity must be moved into ZA without changing the existing row and column totals. A plus sign is inserted into ZA and plus and minus signs in other various occupied cells according to their actual pattern

Table 4.1: Initial solution revisited (showing the required movement use of route ZA)

To / From	A	B	C	Capacity
X	500 [8]	[10]	[7]	500
Y	100 [5]	150 [4]	[8]	250
Z	+ [6]	250 [8]	500 [4]	750
demand	600	400	500	1500

Step 3: Reshuffle the unit to be allocated according to the dictate of the plus and minus signs by the movement in table 4.1. In other words the units to be transferred to ZA are those of the route with the smallest quantity and has a minus sign. The route YA goes out of use in table 4.2, and is replaced by ZA.

Table 4.2: Optimal Solution

To / From	A	B	C	Capacity
X	500 [8]	[10]	[7]	500
Y	[5]	250 [4]	[8]	250
Z	100 [6]	150 [8]	500 [4]	750
demand	600	400	500	1500

Total cost: ₦8,800

Step 4: Test table 4.2 for optimality.

XB = 10 – 8 + 6 - 8 = 0 (No additional cost or saving)

XC = 7 – 4 + 6 – 8 = 1 (Additional cost of N1/unit)

YA = 5 – 4 + 8 – 6 = 3 (Additional cost of N3/unit)

YC = 8 – 4 + 8 – 4 = 8 (Additional cost of N8/unit)

Given that the outcome of step 4 are all positive, it shows that no other route would give us further saving. We, therefore, conclude that the allocation in Table 3 is optimal. Hence we recommend the following shipping schedule to the manufacturer.

From	To	Unit	Unit Cost	Total Cost
X	A	500	8	4,000
Y	B	250	4	1,000
Z	A	100	6	600

Z	B	150	8	1,200
Z	C	500	4	2,000
Total Transportation Cost				8,800

4.2 – Analysis of the Northwest Corner Method

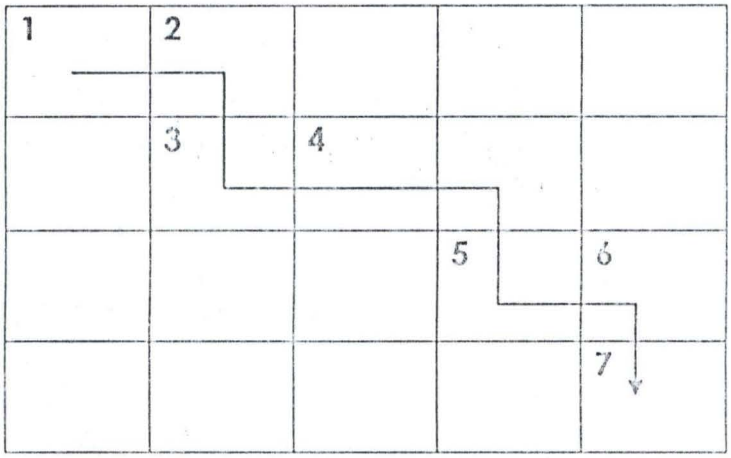
The Nigerian Bottling Company Coca-Cola Kaduna Plc, has various numbers of trucks which are in working condition and ply the various routes daily between the hours of 6:00am and 10:00pm (16 hours daily). We therefore base the research on the Kaduna metropolis. Every morning the transport officer schedules the number of trucks per day from it different types of trucks to five of the following routes: Zaria, Igabi, Makarfi, Ikara and Kauru.

Though the Northwest Corner rule is a simple procedure for obtaining a basic feasible solution of the transportation problem, the solution may however, be different from the optimum solution, since the transportation costs are not taken into consideration like the Least Cost Method.

To / From	Ikara	Kauru	Zaria	Igabi	Markafi	Capacity
Q	200 [40]	50 [60]	[70]	[70]	[20]	250
R	[55]	[100]	150 [90]	60 [90]	70 [20]	280
S	50 [40]	[50]	[60]	50 [70]	[15]	100
T	[35]	[30]	[40]	[10]	20 [10]	20
demand	250	50	150	110	90	132

The associated transportation cost is N35,600.

The staircase pattern is shown in the model below and shows that the solution is valid.



4.3 – Implementing the Transporting Algorithm

The computer has become an increasingly important tool of operation research. Management Information Systems (MIS) allows the researcher to use operation research to manipulate, computerize, compute, analyze and monitor company performance against projects and budget.

Decision support system allows an analyst to manipulate the variables of a computerized model of a real system to test and evaluate the consequences of alternative programs before they are implemented.

Hence the use of mathematical programming which is the theoretical tool of management science and economics, in which management operations are described as LINEAR PROGRAMMING, an example of the mathematical model given for units transported

from a particular source to a destination. If more complex forms are required, the term non-linear programming is applied.

Mathematical programming is used for planning, production schedule, military logistics, calculating economic growth and transportation problems. This is possible by inserting assumed values for variables in the equations and solving for the unknown.

4.4 – Description of the statistical tool used.

The statistical tool employed in this study is the time series analysis as explained below:

- ***Time Series Analysis***: The time series analysis is defined as statistical data which are collected, observed and recorded at successive interval of time. It is also defined as recording observation of a variable, which is the function of time that results in a set of numbers. Time series data are set of data from quantitative events that are recorded over a period of time. The analysis of time series data on these circumstances usually focuses on two types of problems.
    - Attempting to estimate the factors of components that produces the pattern on the time series.
    - Using this estimate in forecasting the future behavior of the time series.

4.5 – The least square method estimating trend

The satisfactory method to describe a trend is to use a mathematical equation. The type of equation chosen depends on the nature of the phenomenon under study. If the values in the time series are

expected to increase of decrease at a constant rate, a straight line fit to the data is used. But if a constant percentage rate of change is expected, an exponential curve may be used. Various curves may be used depending on the circumstance, but a straight line and exponential curve are the most common used.

Let the variable x represents the independent variable of the time series, which is time and y represents the value of dependent variable. x stands for the number of years and y stands for the number of products/crates. Years(x) are coded in such a way that the sum is zero, then the linear trend equation can be given as:

$Y_i = a + bx_i + e_i$

For i = 1, 2, ..., n

xy is the coded years and number of products/crates

4.6 – Statement of Hypothesis

Hypothesis is a conjectural proposition, informed and intelligent about the solution to the problem whose voracity and significance is to be established.

Hypothesis

H_o: a = b = 0 (Estimates are not significant)

H_1: at least on differ from zero

Note: If H_1 holds, then estimates are reliable or accepted.

4.7 – Uses of the analysis of time series

- It helps in understanding the past behavior of variables and also in determining the rate of growth, the extent and direction of periodic fluctuations. The transportation manager who wishes to know whether the product distributed is fair, when and why if fluctuates, and how it is compared with the number of products needed. To the economist, the desire to trace the trend of goods and examine the upward and downward movement of arrival time, such analysis is of great help.

- To study the past behavior of variables enables us to predict the future tendencies. To business executives who are to plan their production programmes such as analysis, it is of great assistance. With the help of this analysis, the approximate estimate of the future demand can be made.

- Knowledge of the behavior of the variable enables us to iron out intra-year variation. These make effective adversitements to reduce seasonal ups and downs in transporting goods/products.

- The determination of the impact of various forces influencing different variables to facilitate their comparison.

4.8 – Method of time series analysis

These include:

- Secular method
- Free hand method
- The semi-average method
- The moving average method

- The least square method

We shall limit this analysis to Least Square Method of the Time Series Analysis.

Table showing Least Square Method using Time Series

Years(x)	No of Crates	X	X^2	Xy
1995	116000	-5	25	-580000
1996	122500	-4	16	-490000
1997	126500	-3	9	-379500
1998	133500	-2	4	-267000
1999	140000	-1	1	-140000
2000	148000	1	1	148000
2001	155000	2	4	310000
2002	162000	3	9	486000
2003	169000	4	16	676000
2004	174500	5	25	872500
	$\sum y=1447000$	$\sum x=0$	$\sum x^2=110$	$\sum=636000$

$$b = \frac{\sum xy}{\sum x^2} = \frac{636000}{110} = 5781.82$$

$$a = \frac{\sum y}{N} = \frac{1447000}{10} = 144{,}700$$

Therefore, $y_i = 144{,}700 + 5781.82x_i$ 1

To predict the number of crates that will be supplied in 2005, 2006 and 2007, we put x to be 6, 7 and 8 respectively in equation 1.

When x = 6, then $y_i = 179{,}391$.

When x = 7, then $y_i = 185{,}172$.

When x = 8 then $y_i = 190{,}955$.

Table showing chi-square calculated

Destination	Observed Supply	Expected Supply	$(O-e)^2/e$
Igabi	116,000	143,800	5374041
Makarfi	126,500	143,800	2081.29
Kauru	140,100	143,800	100.42
Ikara	162,000	143,800	2303.48
Zaria	174,500	143,800	45340.17
			$\sum$ = 16,413.77

Hypothesis

H0: Supply is independent of the destinations.

H1: Supply depends on the destination.

H0 stands for null hypothesis while H1 stands for alternative hypothesis.

Decision

If the Chi-Square calculated is less than the Chi-Square tabulated, then H0 is accepted, otherwise H0 is rejected.

Calculated = 16,413.77

Tabulated = 9.48

Since calculated > tabulated, we reject H0 and conclude that supply depends on the destination.

4.9 – Validity checks of the model

A model is only realistic from a particular point of view. The appropriateness of a model is dependent on the context where it will be used. A model should be tested continuously while it is been constructed. If it's testing is not carried out on in parallel with its construction, the model tends to acquire sanctity during development that makes it difficult to evaluate the model objective after its completion.

When a model is completed, it should be tested as a whole, if it fails such testing, the nature of its deficiency should be determined and corrected. The kinds of deficiency that a model can suffer are as follows:

- It may include irrelevant variables.
- It may exclude relevant variables.
- One or more relevant variables may be evaluated incorrectly.
- It's structure (i.e. the function that relates performance to control and uncontrolled variables) may be in error.

The technique that can be used in testing for these types of deficiencies is statistical in nature. A common method for testing the validity of a model is to compare its performance with PSS + data for the actual system. Unfortunately in our own case it was not possible to obtain reliable historical data of Coca-Cola Kaduna. Our manual solution was thus compared with the computed solution. With all this in mind and with the fact that our problem was using the model, we could correctly say that our model is valid.

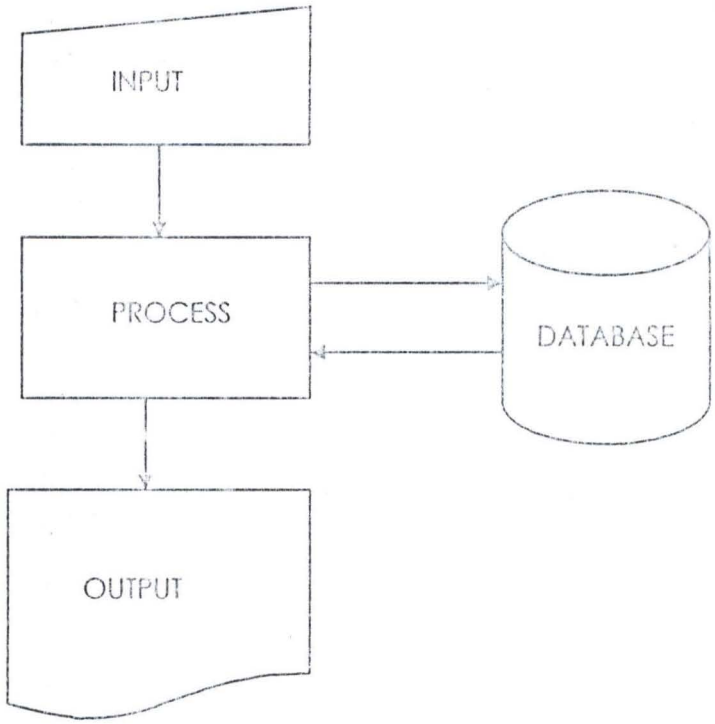
A model is then said to be valid if it can give a reliable prediction of its system performance. It was achieved as it helped us to find an optimum way of scheduling the truck of our five destinations as well as minimizing our transportation cost.

4.10 – Reliable prediction of system performance

Comparing and examining the models we can deduce these results from below:

Type of Truck	Destination	No of Crates taken
Truck Q	Ikara	200
Q	Kaura	50
Lorry R	Zaria	150
R	Igabi	60
R	Makarfi	70
Bus S	Ikara	50
S	Zaria	50
J-Five T	Markafi	20

4.11 – System Flowchart



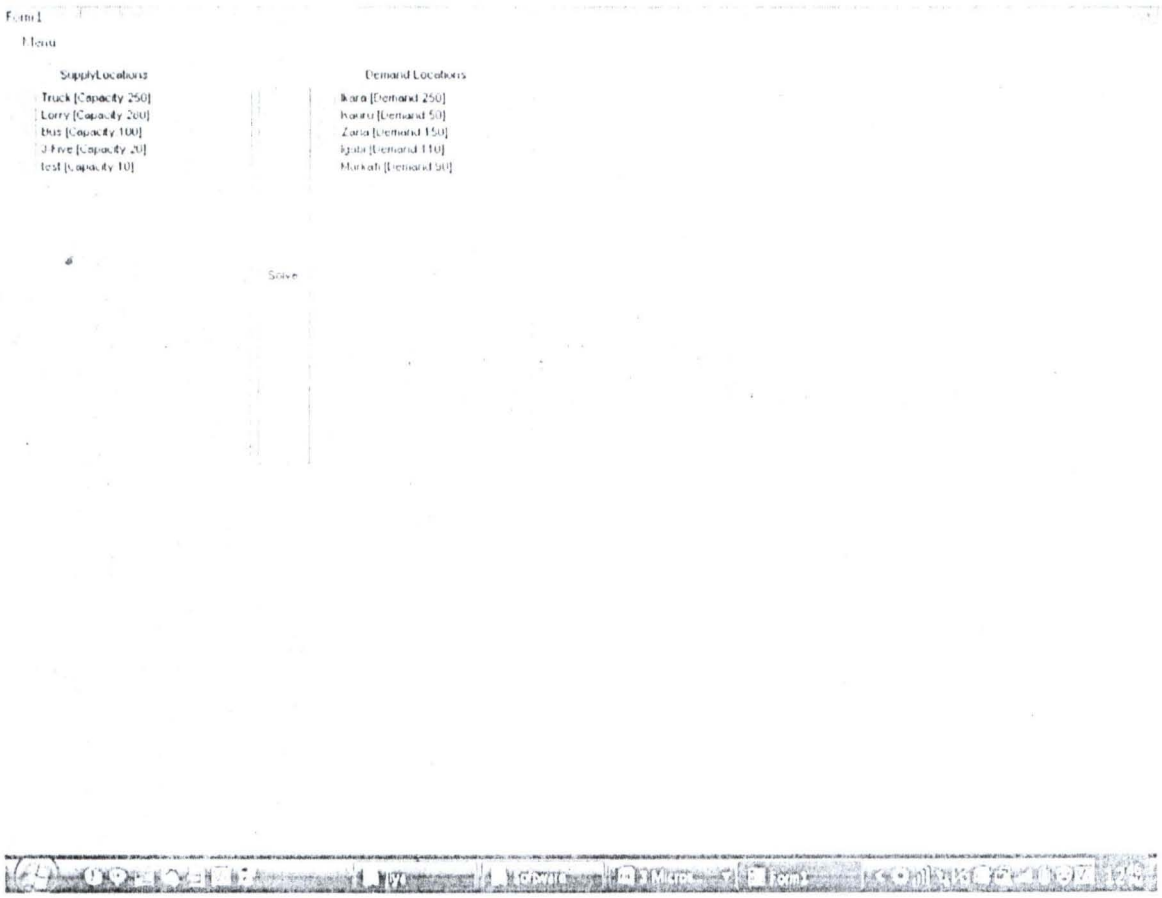


Fig. 1. Main page

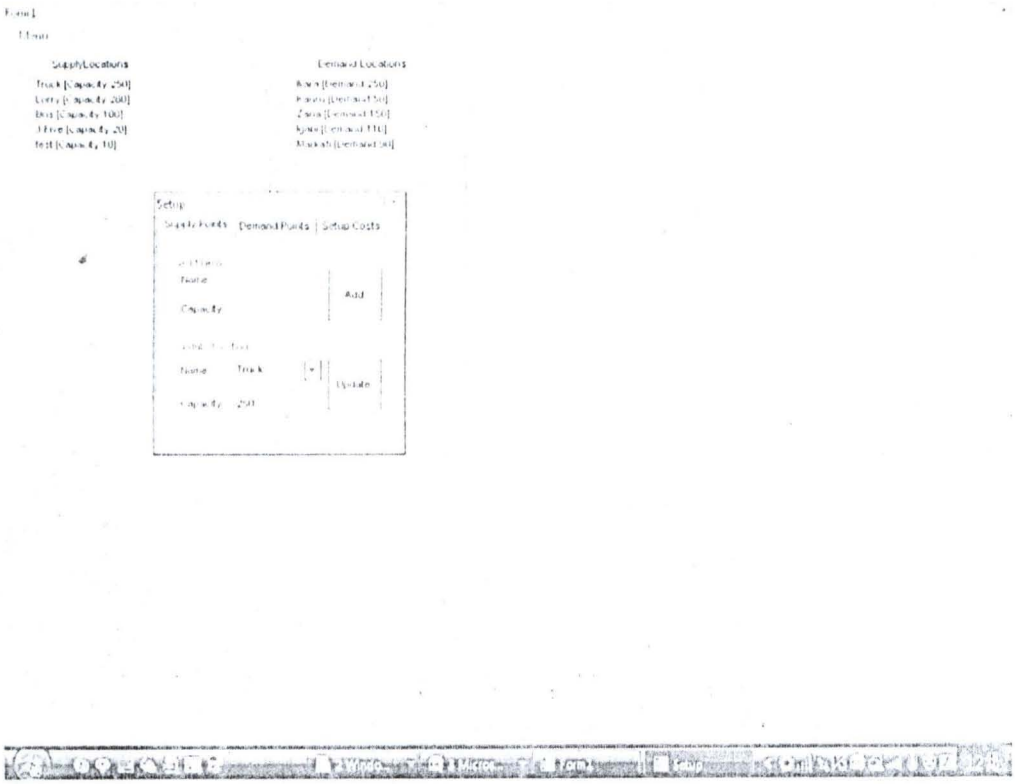


Fig. 2. Setting up Demand Locations

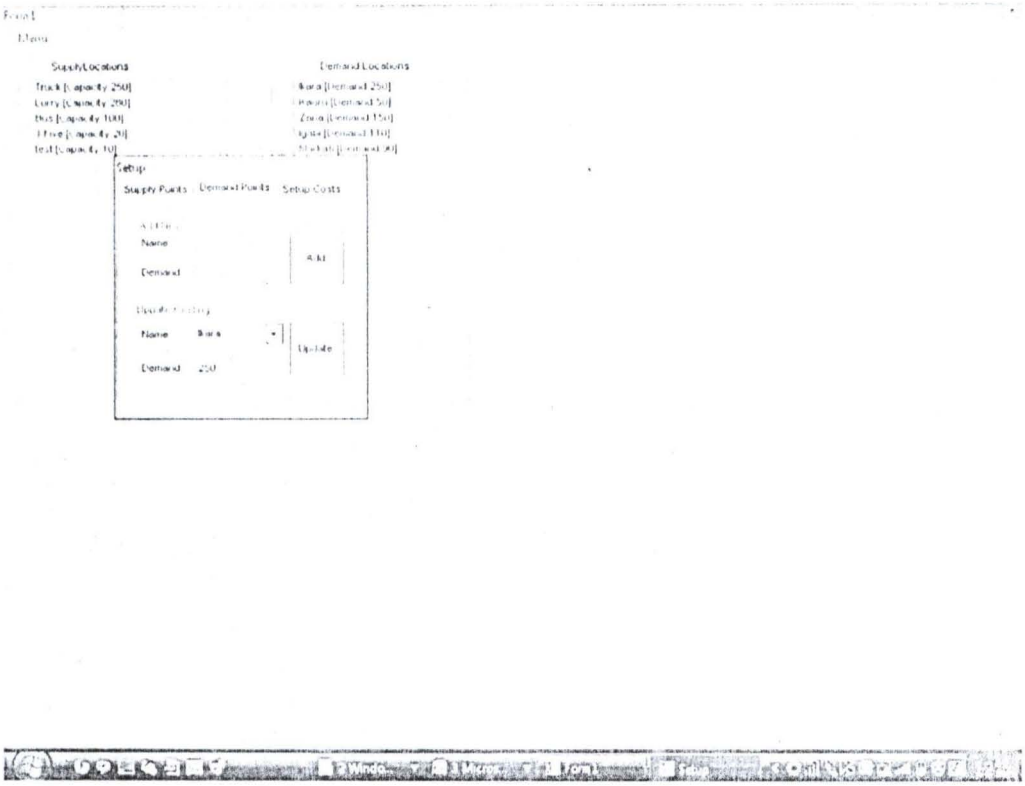


Fig. 3. Setting up transport cost

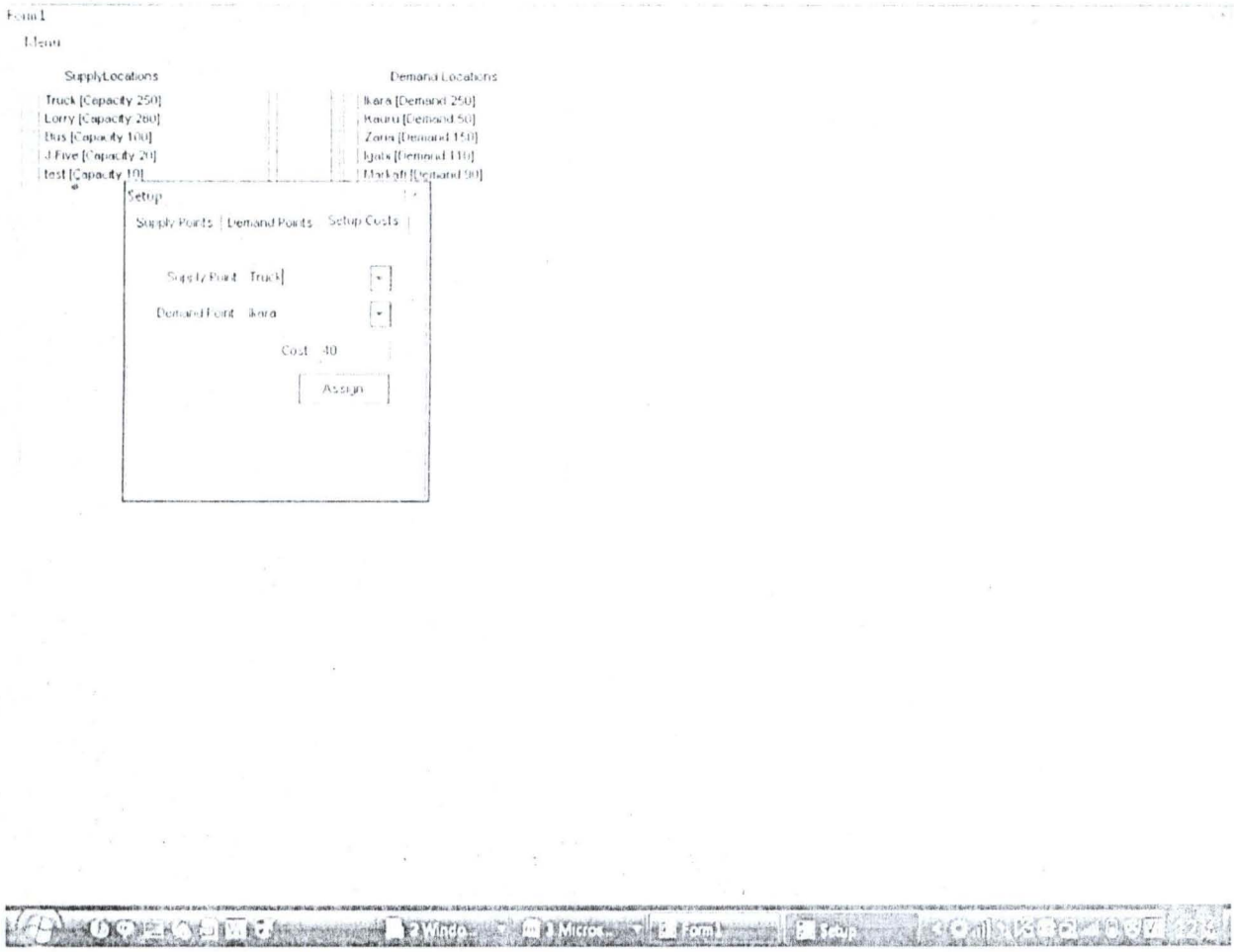


Fig. 4.Setting up Transport Costs

CHAPTER 5 – CONCLUSIONS AND RECOMMENDATIONS

5.1 – Conclusions

This project demonstrates some of the potential benefit which the transportation industry can derive from the use of operation research techniques. This project research was introduced in the project by defining terms like lorries, transportation and trucks of the Nigerian Bottling Company.

The method of solving the "Transportation Problem" was enumerated therefore, and a pratical and physical problem was solved using these methods.

It was observed that the Northwest Corner Method was the simplest method to apply and the only method to have a basic feasible solution, but did not have an optimal starting solution. As against our objective of study, the Northwest Corner Method has the highest transportation cost in its solution.

The Vogel's Approximation Method and the Least Cost Method showed the optimal starting solution while the Northwest Corner Method shows the basic feasible solution.

Goods/Products are transported to five different Local Government Areas from Kaduna (origin) each day. Other data collection considered – types of trucks, capacity, number of trucks, maintenance cost, time taken for each trip was necessary to determine the average cost. From this data our solution was derived and tested and the results of the model were implemented to achieve a desired result.

There was a challenge in getting data from Coca-Cola Kaduna as they did not have the proper necessary records.

The time series model was used to forcast future time series values. We began by realizing it could be useful to think of a series as consisting of trend, seasonal, cyclical and irregular components. If these components are remaining constant over time, then it is an appropriate series regression model. We discussed using such models to describe trend, a linear trend, a quadratic trend, and constant seasonal variation, and we saw that if the error terms exhibit first order

autocorrelation, then we can model this by using a first-order auto-regressive process. As an alternative to using a transformation and dummy variable to model increasing seasonal variation, to model increasing seasonal variation, we can use multiple decomposition method.

Discussion with people could only give ideas on how to go about the missing data variables and why it was necessary to do a research on the "Transportation Modeling", not giving concrete information to help with the research. The method of multiplication and the modified distribution method could only be mentioned and enumerated or work with because of insufficient time and limited resources. It is thus very necessary at this juncture to give some recommendation in the light of the preceding discussion.

5.2 – Recommendations

The management of Coca-Cola Kaduna should try to keep record of all transactions, be it data of products, cost of maintenance, fare of fuel(at different times) etc. This will help in determining the scheduling of lorries/trucks and other advices given based on the prediction of the model.

An improvement should be made on the project research using the method extensively as we have done with the Northwest Corner Method. This would further evolve possibly because of a better scheduling pattern, minimizing the transportation cost.

The management should adopt the transportation models for transporting their products for source to destinations. This will reduce the transportation cost and maximize profits.

Finally the software for the transportation model can be modified, updated and upgraded to suit any organization wishing to adopt the transportation model.

References

- Alan Altshular (1994): Current Issuses in Transportation Policy.
- Bruce Bowerman & Richard T. O. Connell (1997): Applied Statistics, Improving Business Process Models.
- Cyprian A. Oyeka (1990): An Introduction to Applied Statistical Method in the Science.
- Robert J. Thiekauf: An Introductrory Approach to Operation Research.
- Richard Bronson (1997): Operation Research.

Soji Olokoyo (2001): Quantitative Technique for Business Decision, An Operations Research Science Approach.

System Flowchart

```
namespace CocaCola.BusinessEntities
{
  public class AxesCollection : List<Axis>
  {
    public AxesCollection() { }
    public new Axis this[int index]
    {
      get
      {
        foreach (Axis ax in this)
          if (ax.Index == index)
            return ax;
        return null;
      }
    }
    public AxesCollection SupplyAxes
    {
      get
      {
        AxesCollection axc = new AxesCollection();
        foreach (Axis ax in this)
          if (ax.AxisType == AxisType.SupplyRow)
            axc.Add(ax);
        if (axc.Count == this.Count)
        {
          axc = null;
          return this;
        }
        return axc;
      }
    }
    public AxesCollection DemandAxes
    {
      get
      {
        AxesCollection axc = new AxesCollection();
        foreach (Axis ax in this)
          if (ax.AxisType == AxisType.DemandColumn)
            axc.Add(ax);
        if (axc.Count == this.Count)
        {
```

```
            axc = null;
            return this;
          }
          return axc;
        }
      }
      public bool AllAxesHasValues
      {
        get
        {
          foreach (Axis ax in this)
            if (!ax.HasValue)
              return false;

          return true;
        }
      }
    }//end class AxesCollection
    public class Axis
    {
      #region Private Fields
      //
      int index;
      TransportCellsCollection cells;
      public TransportCellsCollection Cells
      {
        get { return cells; }
        set { cells = value; }
      }
      AxisType axisType;
      //
      #endregion
      public Axis(int idx, AxisType type)
      {
        this.index = idx;
        this.axisType = type;
        this.cells = new TransportCellsCollection();
        this.value = null;
      }
      #region Properties
      //
      public int Index
      {
```

```
        get { return this.index; }
        set { this.index = value; }
    }
    public AxisType AxisType
    {
        get { return this.axisType; }
        set { this.axisType = value; }
    }

    private float? value;
    public float? Value
    {
        get { return this.value; }
        set { this.value = value; }
    }
    public bool HasValue
    {
        get { return this.value.HasValue; }
    }
    //
    #endregion
}//end class Axis

public enum AxisType { DemandColumn, SupplyRow }
/// <summary>
/// This keeps track of the current current cell in the transport
/// </summary>
public class CellPosition
{

    public CellPosition()
    {
        //this.dCol = new DemandColumn();
        //this.sCol = new SupplyColumn();
    }

    Axis dAxis;
    Axis sAxis;

    public Axis DemandAxis
    {
        get { return this.dAxis; }
        //set
```

```
    //{
    //    this.dAxis = value;
    //    value.Cells.Add(this);
    //}
  }

  public Axis SupplyAxis
  {
    get { return this.sAxis; }
    //set
    //{
    //    this.sAxis = value;
    //    value.Cells.Add(this);
    //}
  }

  public void AssignAxis(TransportCell tc, Axis ax)
  {
    this.AssignAxis(ax, false);

    ax.Cells.Add(tc);
  }

  public void AssignAxis(Axis ax, bool createNew)
  {
    if (createNew)
    {
      if (ax.AxisType == AxisType.DemandColumn)
      {
        if (this.dAxis == null)
          this.dAxis = new Axis(ax.Index, ax.AxisType);
        else
          this.dAxis.Index = ax.Index;
      }

      else
      {
        if (this.sAxis == null)
          this.sAxis = new Axis(ax.Index, ax.AxisType);
        else
          this.sAxis.Index = ax.Index;
      }
    }
```

```
        else
        {
            if (ax.AxisType == AxisType.DemandColumn)
                this.dAxis = ax;

            else
                this.sAxis = ax;
        }
    }
    public override string ToString()
    {
        return "S:" + this.SupplyAxis.Index.ToString() + ", D:" + this.DemandAxis.Index.ToString();
    }
}//end class CellPosition
public enum RelativeCellPosition { Top, Left, Right, Bottom, Invalid }

public class DemandPoint
{
    private int demand;
    private DemandPointType demandType;
    private int allocation;
    private AllocationStatus allocationStatus;

    /// <summary>
    /// Demand
    /// </summary>
    public int Demand
    {
        get { return demand; }
//        set { demand = value; }
    }

    public int Requirement
    {
        get { return this.Demand - this.Allocation; }
    }


    public AllocationStatus AllocationStatus
    {
        get { return this.allocationStatus; }
        set { this.allocationStatus = value; }
    }
```

```
    public int Allocation
    {
        get { return this.allocation; }
        set { this.allocation = value; }
    }

    public DemandPoint(string nme, int dmd)
    {
        this.demand = dmd;
        this.name = nme;
        this.DemandType = DemandPointType.Actual;
        this.allocationStatus = AllocationStatus.None;
    }

    public DemandPointType DemandType
    {
        get { return this.demandType; }
        set { this.demandType = value; }
    }

    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public override string ToString()
    {
//      return this.Name + " (" + this.demand.ToString() + ", Alloc: " + this.Allocation.ToString() + ")";
        return this.Name + " [Demand:" + this.demand.ToString() + "]";
    }
}//end class DemandPoint

public enum DemandPointType { Actual, Dummy }
public class DemandPointsCollection : List<DemandPoint>
{
    public DemandPointsCollection() { }

    /// <summary>
    /// Gets the total demand
```

```
    /// </summary>
    public int TotalDemand
    {
        get
        {
            int total = 0;
            foreach (DemandPoint dp in this)
                total += dp.Demand;
            return total;
        }
    }
}//end class DemandPointsCollection
public class SupplyPoint
{
    private int capacity;
    private string name;
    private int constCapacity;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    /// <summary>
    /// Capacity of supply depot
    /// </summary>
    public int Capacity
    {
        get { return capacity; }
        set { capacity = value; }
    }

    public int ConstCapacity
    {
        get { return this.constCapacity; }
    }

    public SupplyPoint(string nme, int cpty)
    {
        this.name = nme;
        this.capacity = this.constCapacity = cpty;
    }
```

```
    public override string ToString()
    {
//        return this.Name + " (" + this.constCapacity.ToString() + ", Rem: " + this.Capacity.ToString() + ")";
        return this.Name + " [Capacity:" + this.constCapacity.ToString() + "]";
    }
}//end class SupplyPoint
public class SupplyPointsCollection : List<SupplyPoint>
{
    public SupplyPointsCollection() { }

    /// <summary>
    /// Gets the total capacity
    /// </summary>
    public int TotalCapacity
    {
        get
        {
            int total = 0;
            foreach (SupplyPoint sp in this)
                total += sp.Capacity;
            return total;
        }
    }
public class TransportCell
{
    private float cost;

    /// <summary>
    /// Cost of transporting
    /// </summary>
    public float Cost
    {
        get { return cost; }
        set { cost = value; }
    }

    CellPosition cPosition;

    public CellPosition CellPosition
    {
        get { return this.cPosition; }
        set { this.cPosition = value; }
```

```
}

List<PathDirections> triedPaths;

public List<PathDirections> TriedPaths
{
    get
    {
        return this.triedPaths;
    }
}

/// <summary>
/// Gets or sets the allocation for that cell
/// </summary>
public int DemandSourceAllocation
{
    get { return this.demandSource.Allocation; }
    set { this.demandSource.Allocation = value; }
}

private int cellAllocation;

public int CellAllocation
{
    get { return this.cellAllocation; }
    set { this.cellAllocation = value; }
}

public TransportCell()
{
//    this.position = new TransportMatrixPosition();
    this.cPosition = new CellPosition();
    this.triedPaths = new List<PathDirections>();
}

private SupplyPoint supplySource;

/// <summary>
/// Supply point of the cell
/// </summary>
public SupplyPoint SupplySource
{
```

```
        get { return supplySource; }
        set { supplySource = value; }
    }
    private DemandPoint demandSource;

    /// <summary>
    /// Demand point of the cell
    /// </summary>
    public DemandPoint DemandSource
    {
        get { return demandSource; }
        set { demandSource = value; }
    }

    private bool isAllocated = false;

    public bool IsAllocated
    {
        get { return this.isAllocated; }
        set { this.isAllocated = value; }
    }


    public void Allocate()
    {
        if (this.supplySource.Capacity >= this.demandSource.Reqirement)
        {
            this.supplySource.Capacity -= this.demandSource.Reqirement;
            this.CellAllocation = this.demandSource.Reqirement;
            this.DemandSourceAllocation += this.CellAllocation;
            this.AllocationStatus = AllocationStatus.Complete;
        }
        else
        {
            this.CellAllocation = this.supplySource.Capacity;
            this.DemandSourceAllocation += this.CellAllocation;
            this.supplySource.Capacity = 0;
            this.AllocationStatus = AllocationStatus.Partial;
        }
        this.isAllocated = true;
    }

    public AllocationStatus AllocationStatus
```

```
    {
        get { return this.demandSource.AllocationStatus; }
        set { this.demandSource.AllocationStatus = value; }
    }

    private CellSign sign = CellSign.Empty;

    public CellSign Sign
    {
        get { return this.sign; }
        set { this.sign = value; }
    }

    public override string ToString()
    {
        string sign;
        if (this.Sign == CellSign.Positive)
            sign = "+";
        else if (this.sign == CellSign.Negative)
            sign = "-";
        else
            sign = "";

        return "S: " + this.supplySource.Name + ", D: " + this.demandSource.Name + ", All: " + this.CellAllocation.ToString()
+ sign;
    }

    public bool IsDummy
    {
        get
        {
            bool rtn = false;

            rtn = this.supplySource.Name == "Dummy";
            rtn = rtn ? rtn : this.demandSource.Name == "Dummy";

            return rtn;
        }
    }

    private float tmpCellAllocation;

    public float TmpCellAllocation
    {
```

```
        get { return tmpCellAllocation; }
        set { tmpCellAllocation = value; }
    }

    public RelativeCellPosition GetRelativePositionFromCell(TransportCell tc)
    {
        if (this.CellPosition.DemandAxis.Index == tc.CellPosition.DemandAxis.Index)
        {
            if (this.CellPosition.SupplyAxis.Index > tc.CellPosition.SupplyAxis.Index)
                return RelativeCellPosition.Bottom;
            else
                return RelativeCellPosition.Top;
        }
        else if (this.CellPosition.SupplyAxis.Index == tc.CellPosition.SupplyAxis.Index)
        {
            if (this.CellPosition.DemandAxis.Index > tc.CellPosition.DemandAxis.Index)
                return RelativeCellPosition.Right;
            else
                return RelativeCellPosition.Left;
        }
        else
            return RelativeCellPosition.Invalid;
    }

    public float TransportCost
    {
        get
        {
            return this.Cost * this.CellAllocation;
        }
    }
}//end class TransportCell
public enum AllocationStatus { Complete, Partial, None }
public enum CellSign { Positive, Negative, Empty }
public enum PathDirections { Top, Bottom, Left, Right };
    public TransportCellsCollection() { }

    #region Properties
    //
    public TransportCell this[Axis sc, Axis dc]
    {
        get
        {
```

```
        foreach (TransportCell tc in this)
            if (tc.CellPosition.SupplyAxis.Index == sc.Index && tc.CellPosition.DemandAxis.Index == dc.Index)
                return tc;
        return null;
    }
}

public float TotalCost
{
    get
    {
        float total = 0.0F;
        foreach (TransportCell tc in this.AllocatedCells)
            total += tc.TransportCost;
        return total;
    }
}

public TransportCellsCollection AllocatedCells
{
    get
    {
        TransportCellsCollection tcc = new TransportCellsCollection();

        foreach (TransportCell tc in this)
            if (tc.CellAllocation > 0)
                tcc.Add(tc);

        if (tcc.Count == this.Count)
        {
            tcc = null;
            return this;
        }

        return tcc;
    }
}

public TransportCellsCollection UnAllocatedCells
{
    get
    {
        TransportCellsCollection tcc = new TransportCellsCollection();
```

```
        foreach (TransportCell tc in this)
            if (tc.CellAllocation == 0)
                tcc.Add(tc);

        if (tcc.Count == this.Count)
        {
            tcc = null;
            return this;
        }

        return tcc;
    }
}

public TransportCell MostNegativeTmpAllocatedCell
{
    get
    {
        TransportCell tc = this[0];
        foreach (TransportCell t in this)
            if (t.CellAllocation == 0 && t.TmpCellAllocation < tc.TmpCellAllocation)
                tc = t;

        return tc;
    }
}

public TransportCell LeastNegativeSignAllocatedCell
{
    get
    {
        TransportCellsCollection tcc = this.AllocatedCells;
        TransportCell least = null;
        foreach (TransportCell t in tcc)
        {
            if (t.Sign == CellSign.Negative)
            {
                least = t;
                break;
            }
        }
```

```
            foreach (TransportCell t in tcc)
                if (t.Sign == CellSign.Negative && t.CellAllocation < least.CellAllocation)
                    least = t;
            return least;
        }
    }
    //
    #endregion

}//end class TransportCellsCollection

public class NoTransportCostException : Exception
{
    public NoTransportCostException(string message) : base(message) { }
}

public class TransportMatrix
{
    #region Private fields
    //
    private SupplyPointsCollection supplyPoints;
    private DemandPointsCollection demandPoints;
    private AxesCollection axes;
    private TransportCellsCollection transportCells;
    private CellLocator cLocator;
    //
    #endregion

    public TransportMatrix()
    {
        this.DemandPoints = new DemandPointsCollection();
        this.SupplyPoints = new SupplyPointsCollection();
        this.axes = new AxesCollection();
    }

    #region Properties
    //
    /// <summary>
    /// List of supply points
    /// </summary>
    public SupplyPointsCollection SupplyPoints
    {
        get { return supplyPoints; }
```

```
    set { supplyPoints = value; }
  }

  public CellLocator CellLocator
  {
    get { return this.cLocator;  }
  }

  /// <summary>
  /// List of demand points
  /// </summary>
  public DemandPointsCollection DemandPoints
  {
    get { return demandPoints; }
    set { demandPoints = value; }
  }

  /// <summary>
  /// List of cells that make up the transport matrix
  /// </summary>
  public TransportCellsCollection TransportCells
  {
    get
    {
      if (this.transportCells == null)
        this.transportCells = new TransportCellsCollection();
      return transportCells;
    }
  }

  public Axis HighestAllocationAxis
  {
    get
    {
      //Get axis when highest allocations
      Axis targetAxis = this.axes[1];
      foreach (Axis ax in this.axes)
      {
        if (ax.Cells.AllocatedCells.Count > targetAxis.Cells.AllocatedCells.Count)
          targetAxis = ax;
      }

      //Get all axes with same number of allocations with the highest allocation axis
```

```
        AxesCollection tmp = new AxesCollection();
        tmp.Add(targetAxis);
        foreach (Axis ax in this.axes)
        {
            if (ax.Cells.AllocatedCells.Count == targetAxis.Cells.AllocatedCells.Count)
                tmp.Add(ax);
        }

        //get the axis closest to the origin
        foreach (Axis ax in tmp)
        {
            if (ax.Index < targetAxis.Index)
                targetAxis = ax;
        }

        return targetAxis;
    }
}

public AxesCollection Axes
{
    get { return axes; }
    set { axes = value; }
}

public bool IsOptimum
{
    get
    {
        foreach (TransportCell tc in this.transportCells.UnAllocatedCells)
            if (tc.TmpCellAllocation <= 0)
                return false;
        return true;
    }
}

public bool HasDegeneracy
{
    get
    {
        int totalAllocations = 0;
        foreach (TransportCell tc in this.transportCells)
            if (tc.CellAllocation > 0)
```

```
            totalAllocations++;

        if (totalAllocations == this.axes.Count - 1)
            return false;

        return true;
    }
}

public TransportCell this[CellPosition cPosition]
{
    get
    {
        return this.transportCells[cPosition.SupplyAxis, cPosition.DemandAxis];
    }
}

public TransportCell CurrentCell
{
    get
    {
        if (this.cLocator.LocationValid)
            return this[this.cLocator.Position];

        return null;
    }
}

//
#endregion

#region Public Methods
//
private void SetUpColumns()
{
    for (int supplyIndex = 0; supplyIndex < this.SupplyPoints.Count; supplyIndex++)
        this.axes.Add(new Axis(supplyIndex + 1, AxisType.SupplyRow));

    for (int demandIndex = 0; demandIndex < this.DemandPoints.Count; demandIndex++)
        this.axes.Add(new Axis(demandIndex + 1, AxisType.DemandColumn));

    this.cLocator = new CellLocator(this.demandPoints.Count, this.supplyPoints.Count);
}
```

```
public void SetUpTransportMatrix()
{
    int totalSupply = this.SupplyPoints.TotalCapacity;
    int totalDemand = this.DemandPoints.TotalDemand;
    if (totalSupply != totalDemand)
    {
        if (totalSupply > totalDemand)
        {
            DemandPoint dp = new DemandPoint("Dummy", totalSupply - totalDemand);
            this.DemandPoints.Add(dp);
        }
        else
        {
            SupplyPoint sp = new SupplyPoint("Dummy", totalDemand - totalSupply);
            this.SupplyPoints.Add(sp);
        }
    }

    this.SetUpColumns();

    using (TransportCostDAL tcDAL = new TransportCostDAL())
    {
        for (int supplyIndex = 0; supplyIndex < this.SupplyPoints.Count; supplyIndex++)
        {
            for (int demandIndex = 0; demandIndex < this.DemandPoints.Count; demandIndex++)
            {
                TransportCell tc = new TransportCell();
                tc.SupplySource = this.SupplyPoints[supplyIndex];
                tc.DemandSource = this.DemandPoints[demandIndex];

                tc.CellPosition.AssignAxis(tc, this.axes.DemandAxes[demandIndex + 1]);
                tc.CellPosition.AssignAxis(tc, this.axes.SupplyAxes[supplyIndex + 1]);

                if (!tc.IsDummy)
                {

                    float cost = tcDAL.GetTransportCost(tc.SupplySource.Name, tc.DemandSource.Name);

                    if (cost == 0)
                        throw new NoTransportCostException("Transport cost between " + tc.SupplySource.Name + " and "
                            + tc.DemandSource.Name + " has not been set. Cannot continue.");
```

```
                tc.Cost = cost;
            }

            this.TransportCells.Add(tc);
        }
    }
}

public void GetInitialFeasibleSolution()
{
    foreach (Axis sc in this.axes.SupplyAxes)
    {
        foreach (Axis dc in this.axes.DemandAxes)
        {
            TransportCell tc = this.TransportCells[sc, dc];

            if (tc.AllocationStatus == AllocationStatus.Complete)
                continue;
            if (tc.SupplySource.Capacity == 0)
                break;

            tc.Allocate();
        }
    }
}

public void Display(TableLayoutPanel tablePanel)
{
    tablePanel.Controls.Clear();
    tablePanel.RowCount = 0;
    tablePanel.ColumnCount = 0;

    tablePanel.Visible = false;

    tablePanel.ColumnCount = this.demandPoints.Count + 1;
    tablePanel.RowCount = this.supplyPoints.Count + 1;

    int supplyIndex = 1;
    foreach (SupplyPoint sp in this.supplyPoints)
    {
        Label lbl = new Label();
        lbl.Size = System.Drawing.Size.Empty;
```

```
        lbl.AutoSize = true;
        lbl.Font = new System.Drawing.Font(lbl.Font, System.Drawing.FontStyle.Bold);
        lbl.ForeColor = System.Drawing.Color.Blue;
        if(sp.Name == "Dummy")
            lbl.ForeColor = System.Drawing.Color.Red;
        lbl.Text = sp.Name + "\n" + sp.ConstCapacity;
        tablePanel.Controls.Add(lbl, 0, supplyIndex++);
    }

    int demandIndex = 1;
    foreach (DemandPoint dp in this.demandPoints)
    {
        Label lbl = new Label();
        lbl.Size = System.Drawing.Size.Empty;
        lbl.AutoSize = true;
        lbl.Font = new System.Drawing.Font(lbl.Font, System.Drawing.FontStyle.Bold);
        lbl.ForeColor = System.Drawing.Color.Blue;
        if (dp.Name == "Dummy")
            lbl.ForeColor = System.Drawing.Color.Red;
        lbl.Text = dp.Name + "\n" + dp.Demand;
        tablePanel.Controls.Add(lbl, demandIndex++, 0);

    }

    foreach (TransportCell tc in this.transportCells)
    {
        Label lbl = new Label();
        StringBuilder txt = new StringBuilder();

        lbl.Size = System.Drawing.Size.Empty;
        lbl.AutoSize = true;

        txt.Append("Cost: ");
        txt.Append(tc.Cost.ToString());

        if (tc.CellAllocation > 0)
        {
            txt.Append(Environment.NewLine);
            txt.Append("------");
            txt.Append(Environment.NewLine);
            txt.Append("Alloc: ");
            txt.Append(tc.CellAllocation.ToString());
```

```
            lbl.ForeColor = System.Drawing.Color.LightSeaGreen;

            lbl.Font = new System.Drawing.Font(lbl.Font, System.Drawing.FontStyle.Bold);
          }

          if(tc.IsDummy)
            lbl.ForeColor = System.Drawing.Color.Red;

          lbl.Text = txt.ToString();

          tablePanel.Controls.Add(lbl, tc.CellPosition.DemandAxis.Index, tc.CellPosition.SupplyAxis.Index);
        }

        tablePanel.Visible = true;

      }

      public string ShowHigestAllocatedAxis()
      {
        string name;
        Axis ax = this.HighestAllocationAxis;
        if (ax.AxisType == AxisType.DemandColumn)
          name = this.demandPoints[ax.Index - 1].ToString();
        else
          name = this.supplyPoints[ax.Index - 1].ToString();

        return name;
      }

      public void SolveForAxesValues()
      {
        foreach (TransportCell tc in this.transportCells.AllocatedCells)
        {
          if (!tc.CellPosition.DemandAxis.HasValue || !tc.CellPosition.SupplyAxis.HasValue)
          {
            if (tc.CellPosition.DemandAxis.HasValue)
              tc.CellPosition.SupplyAxis.Value = tc.Cost - tc.CellPosition.DemandAxis.Value;
            else if (tc.CellPosition.SupplyAxis.HasValue)
              tc.CellPosition.DemandAxis.Value = tc.Cost - tc.CellPosition.SupplyAxis.Value;
          }
        }
      }
```

```
public void SolveForUnOccupiedCells()
{
    foreach (TransportCell tc in this.transportCells)
        tc.TmpCellAllocation = 0.0F;

    foreach (TransportCell tc in this.transportCells.UnAllocatedCells)
        tc.Tmp_CellAllocation = tc.Cost - (tc.CellPosition.DemandAxis.Value.Value + tc.CellPosition.SupplyAxis.Value.Value);
}

public TransportCell FindCell(TransportCell currentCell, RelativeCellPosition rPosition)
{
    return this.FindCell(currentCell, rPosition, true);
}

public TransportCell FindCell(TransportCell currentCell, RelativeCellPosition rPosition, bool search)
{
    TransportCell tc = null;

    this.cLocator.SetCurrentPosition(currentCell);

    switch (rPosition)
    {
        case RelativeCellPosition.Top:
            this.cLocator.MoveUp();
            while (this.CurrentCell != null && search)
            {
                if (this.CurrentCell.CellAllocation > 0)
                {
                    tc = this.CurrentCell;
                    break;
                }
                this.cLocator.MoveUp();
            }
            break;
        case RelativeCellPosition.Bottom:
            this.cLocator.MoveDown();
            while (this.CurrentCell != null && search)
            {
                if (this.CurrentCell.CellAllocation > 0)
                {
                    tc = this.CurrentCell;
                    break;
                }
```

```
            this.cLocator.MoveDown();
        }
        break;
    case RelativeCellPosition.Right:
        this.cLocator.MoveRight();
        while (this.CurrentCell != null && search)
        {
            if (this.CurrentCell.CellAllocation > 0)
            {
                tc = this.CurrentCell;
                break;
            }
            this.cLocator.MoveRight();
        }
        break;
    case RelativeCellPosition.Left:
        this.cLocator.MoveLeft();
        while (this.CurrentCell != null && search)
        {
            if (this.CurrentCell.CellAllocation > 0)
            {
                tc = this.CurrentCell;
                break;
            }
            this.cLocator.MoveLeft();
        }
        break;
    }

    return tc;
}

public TransportCellsCollection GetPath(TransportCell leastCostCell)
{
    TransportCellsCollection path = new TransportCellsCollection();

    TransportCellsCollection triedCells = new TransportCellsCollection();

    TransportCell currentCell = this.GetStartUpCell(leastCostCell, triedCells);

    RelativeCellPosition [] expectedEndPositions;
    RelativeCellPosition startPosition = currentCell.GetRelativePositionFromCell(leastCostCell);
    switch (startPosition)
```

```
{
    case RelativeCellPosition.Left:
    case RelativeCellPosition.Right:
        expectedEndPositions = new RelativeCellPosition [] { RelativeCellPosition.Top, RelativeCellPosition.Bottom };
        break;
    case RelativeCellPosition.Top:
    case RelativeCellPosition.Bottom:
        expectedEndPositions = new RelativeCellPosition[] { RelativeCellPosition.Left, RelativeCellPosition.Right };
        break;
    default:
        expectedEndPositions = null;
        throw new Exception("Start cell in invalid position");
}

CellSign currentSign = CellSign.Positive;
leastCostCell.Sign = currentSign;
path.Add(leastCostCell);
this.SwitchSign(ref currentSign);

path.Add(currentCell);
currentCell.Sign = currentSign;
this.SwitchSign(ref currentSign);

bool triedLeft = false;
bool triedRight = false;
bool triedTop = false;
bool triedDown = false;

while (!this.IsInExpectedPosition(currentCell.GetRelativePositionFromCell(leastCostCell), expectedEndPositions))
{
    if (!triedDown)
    {
        this.CurrentCell.TriedPaths.Add(PathDirections.Bottom);
        this.cLocator.MoveDown();
        if (this.CurrentCell != null && !this.IsInList(path, this.CurrentCell) && this.CurrentCell.CellAllocation > 0)
        {
            currentCell = this.CurrentCell;
            currentCell.Sign = currentSign;
            this.SwitchSign(ref currentSign);
            path.Add(currentCell);
            triedLeft = false;
            triedRight = false;
            triedTop = true;
```

```
            currentCell.TriedPaths.Add(PathDirections.Top);
            triedDown = false;
            continue;
        }
        else
        {
            if (this.cLocator.LocationValid)
                this.cLocator.MoveUp();
            else
                this.cLocator.LocationValid = true;
            triedDown = true;
        }
    }

    if (!triedLeft)
    {
        this.CurrentCell.TriedPaths.Add(PathDirections.Left);
        this.cLocator.MoveLeft();
        if (this.CurrentCell != null && !this.IsInList(path, this.CurrentCell) && this.CurrentCell.CellAllocation > 0)
        {
            currentCell = this.CurrentCell;
            currentCell.Sign = currentSign;
            this.SwitchSign(ref currentSign);
            path.Add(currentCell);
            triedLeft = false;
            triedRight = true;
            currentCell.TriedPaths.Add(PathDirections.Right);
            triedTop = false;
            triedDown = false;
            continue;
        }
        else
        {
            if (this.cLocator.LocationValid)
                this.cLocator.MoveRight();
            else
                this.cLocator.LocationValid = true;
            triedLeft = true;
        }
    }

    if (!triedRight)
    {
```

```
        this.CurrentCell.TriedPaths.Add(PathDirections.Right);
        this.cLocator.MoveRight();
        if (this.CurrentCell != null && !this.IsInList(path, this.CurrentCell) && this.CurrentCell.CellAllocation > 0)
        {
            currentCell = this.CurrentCell;
            currentCell.Sign = currentSign;
            this.SwitchSign(ref currentSign);
            path.Add(currentCell);
            triedLeft = true;
            currentCell.TriedPaths.Add(PathDirections.Left);
            triedRight = false;
            triedTop = false;
            triedDown = false;
            continue;
        }
        else
        {
            if (this.cLocator.LocationValid)
                this.cLocator.MoveLeft();
            else
                this.cLocator.LocationValid = true;
            triedRight = true;
        }
    }

    if (!triedTop)
    {
        this.CurrentCell.TriedPaths.Add(PathDirections.Top);
        this.cLocator.MoveUp();
        if (this.CurrentCell != null && !this.IsInList(path, this.CurrentCell) && this.CurrentCell.CellAllocation > 0)
        {
            currentCell = this.CurrentCell;
            currentCell.Sign = currentSign;
            this.SwitchSign(ref currentSign);
            path.Add(currentCell);
            triedLeft = false;
            triedRight = false;
            triedTop = false;
            triedDown = true;
            currentCell.TriedPaths.Add(PathDirections.Bottom);
            continue;
        }
        else
```

```
        {
            if (this.cLocator.LocationValid)
                this.cLocator.MoveDown();
            else
                this.cLocator.LocationValid = true;
            triedTop = true;
        }
    }
    if (triedTop && triedRight && triedLeft && triedDown)
    {
        TransportCell deadEnd = this.CurrentCell;
        path.Remove(deadEnd);
        this.SwitchSign(ref currentSign);
        if (path.Count == 1)
            throw new PathNotFoundException();
        this.cLocator.SetCurrentPosition(path[path.Count - 1]);

        triedLeft = false;
        triedRight = false;
        triedTop = false;
        triedDown = false;

        foreach (PathDirections paths in this.CurrentCell.TriedPaths)
        {
            switch (paths)
            {
                case PathDirections.Top:
                    triedTop = true;
                    break;
                case PathDirections.Bottom:
                    triedDown = true;
                    break;
                case PathDirections.Left:
                    triedLeft = true;
                    break;
                case PathDirections.Right:
                    triedRight = true;
                    break;
            }
        }
    }

}
```

```
        return path;
    }

    public void PeformAddSubtractOperationOnPath(TransportCellsCollection path)
    {
        int value = path.LeastNegativeSignAllocatedCell.CellAllocation;
        foreach (TransportCell tc in path)
        {
            if (tc.Sign == CellSign.Positive)
                tc.CellAllocation += value;
            else
                tc.CellAllocation -= value;
        }
    }
    //
    #endregion

    #region Private Methods
    //
    private void SwitchSign(ref CellSign sign)
    {
        if (sign == CellSign.Negative)
            sign = CellSign.Positive;
        else
            sign = CellSign.Negative;
    }

    private bool IsInList(TransportCellsCollection triedCells, TransportCell tc)
    {
        foreach (TransportCell t in triedCells)
            if (t == tc)
                return true;
        return false;
    }

    private TransportCell GetStartUpCell(TransportCell leastCostCell, TransportCellsCollection triedCells)
    {
        TransportCell startCell = this.FindCell(leastCostCell, RelativeCellPosition.Top);

        if (startCell == null || this.IsInList(triedCells, startCell))
            startCell = this.FindCell(leastCostCell, RelativeCellPosition.Right);
```

```
        if (startCell == null || this.IsInList(triedCells, startCell))
            startCell = this.FindCell(leastCostCell, RelativeCellPosition.Bottom);

        if (startCell == null || this.IsInList(triedCells, startCell))
            startCell = this.FindCell(leastCostCell, RelativeCellPosition.Left);

        if (startCell == null)
            throw new Exception("Could not find a start cell for path");

        if (this.IsInList(triedCells, startCell))
            throw new Exception("No valid path could be found");

        return startCell;
    }

    private bool IsInExpectedPosition(RelativeCellPosition position, RelativeCellPosition[] expectedEndPositions)
    {
        foreach (RelativeCellPosition rcp in expectedEndPositions)
            if (rcp.ToString() == position.ToString())
                return true;

        return false;
    }
    //
    #endregion

}//end class TransportMatrix

#region CellLocator Class
//
/// <summary>
/// Moves around the transport matrix, while keeping track of the it current position
/// using the <cref>CellPosition</cref> class
/// </summary>
public class CellLocator
{
    #region Private fields
    //
    CellPosition position;
    bool locationValid = true;
    int maxDemandIndex;
    int maxSupplyIndex;
    //
```

```
#endregion

public CellLocator(int mDIdex, int mSIndex)
{
    this.position = new CellPosition();
    this.position.AssignAxis(new Axis(1, AxisType.DemandColumn), true);
    this.position.AssignAxis(new Axis(1, AxisType.SupplyRow), true);
    this.maxDemandIndex = mDIdex;
    this.maxSupplyIndex = mSIndex;
}

#region Properties
//
public bool LocationValid
{
    get {return this.locationValid; }
    set {this.locationValid = value; }
}

public CellPosition Position
{
    get { return this.position; }
}
//
#endregion

#region Methods
//

public override string ToString()
{
    return "S:" + this.position.SupplyAxis.Index.ToString() + ", D:" + this.position.DemandAxis.Index.ToString();
}

public void SetCurrentPosition(TransportCell tc)
{
    this.position.AssignAxis(tc.CellPosition.DemandAxis, true);
    this.position.AssignAxis(tc.CellPosition.SupplyAxis, true);
}

public void MoveLeft()
{
    int index = this.position.DemandAxis.Index;
```

```
        index--;
        if (index < 1)
            this.locationValid = false;
        else
        {
            this.position.DemandAxis.Index--;
            this.locationValid = true;
        }
    }

    public void MoveRight()
    {
        int index = this.position.DemandAxis.Index;
        index++;
        if (index > this.maxDemandIndex)
            this.locationValid = false;
        else
        {
            this.position.DemandAxis.Index++;
            this.locationValid = true;
        }
    }

    public void MoveUp()
    {
        int index = this.position.SupplyAxis.Index;
        index--;
        if (index < 1)
            this.locationValid = false;
        else
        {
            this.position.SupplyAxis.Index--;
            this.locationValid = true;
        }
    }

    public void MoveDown()
    {
        int index = this.position.SupplyAxis.Index;
        index++;
        if (index > this.maxSupplyIndex)
            this.locationValid = false;
        else
```

```
            {
                this.position.SupplyAxis.Index++;
                this.locationValid = true;
            }
        }
        //
        #endregion

    }//end class CellLocator
    //
    #endregion
}

namespace CocaCola.DataAccess
{
    public class DALBase : IDisposable
    {
        private OleDbConnection conn;

        public DALBase()
        {
            this.conn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=data.mdb");
        }

        protected void ExecuteNonQuery(OleDbCommand cmd)
        {
            try
            {
                if(this.conn.State != System.Data.ConnectionState.Open)
                    this.conn.Open();
                cmd.ExecuteNonQuery();
            }
            finally
            {
                this.conn.Close();
            }
        }

        protected OleDbCommand GetCommand(string commandText)
        {
            OleDbCommand cmd = new OleDbCommand(commandText, this.conn);
            return cmd;
        }
```

```
protected OleDbDataReader GetReader(OleDbCommand cmd)
{
    if (this.conn.State != System.Data.ConnectionState.Open)
        this.conn.Open();
    OleDbDataReader reader = cmd.ExecuteReader();
    return reader;
}

protected object ExecuteScalar(OleDbCommand cmd)
{
    object value = null;
    try
    {
        if (this.conn.State != System.Data.ConnectionState.Open)
            this.conn.Open();
        value = cmd.ExecuteScalar();
    }
    finally
    {
        this.conn.Close();
    }
    return value;
}

protected void CloseConnection()
{
    this.conn.Close();
}

protected void ExecuteNonQueryWithTransaction(OleDbCommand[] cmds)
{
    OleDbTransaction trans = null;
    try
    {
        if (this.conn.State != System.Data.ConnectionState.Open)
            this.conn.Open();
        trans = this.conn.BeginTransaction();
        foreach (OleDbCommand cmd in cmds)
        {
            cmd.Transaction = trans;
            cmd.ExecuteNonQuery();
        }
```

```
            trans.Commit();
        }
        catch
        {
            trans.Rollback();
            throw;
        }
    }
    #region IDisposable Members

    public void Dispose()
    {
        this.conn.Dispose();
    }
    #endregion
}
public class DemandPointsDAL :DALBase
{
    public DemandPointsDAL() { }

    public DemandPointsCollection GetDemandPoints()
    {
        DemandPointsCollection dpc = new DemandPointsCollection();
        try
        {
            OleDbDataReader reader = base.GetReader(base.GetCommand("Select * from tbl_DemandPoints"));
            while (reader.Read())
            {
                string name = reader["Location"].ToString();
                int demand = (int)reader["Demand"];
                dpc.Add(new DemandPoint(name, demand));
            }
            reader.Close();
        }
        finally
        {
            base.CloseConnection();
        }
        return dpc;
    }

    public int GetDemand(string demandPoint)
    {
```

```
    }

    public void AssignDemand(string demandPoint, int dmd)
    {
        StringBuilder cmdText = new StringBuilder();
        int demand = this.GetDemand(demandPoint);

        if (demand == 0)
        {
            cmdText.Append("Insert into tbl_DemandPoints (Location, Demand) values ('");
            cmdText.Append(demandPoint);
            cmdText.Append("', '");
            cmdText.Append(dmd);
            cmdText.Append("')");
        }
        else
        {
            cmdText.Append("Update tbl_DemandPoints set Demand = ");
            cmdText.Append(dmd);
            cmdText.Append(" where Location = '");
            cmdText.Append(demandPoint);
            cmdText.Append("'");
        }

        base.ExecuteNonQuery(base.GetCommand(cmdText.ToString()));
    }
}//end class DemandPointsDAL

public class SupplyPointsDAL : DALBase
{
```

```
public SupplyPointsDAL() { }

public SupplyPointsCollection GetSupplyPoints()
{
    SupplyPointsCollection spc = new SupplyPointsCollection();
    try
    {
        OleDbDataReader reader = base.GetReader(base.GetCommand("Select * from tbl_SupplyPoints"));
        while (reader.Read())
        {
            string name = reader["Location"].ToString();
            int capacity = (int)reader["Capacity"];
            spc.Add(new SupplyPoint(name, capacity));
        }
        reader.Close();
    }
    finally
    {
        base.CloseConnection();
    }
    return spc;
}

public int GetCapacity(string supplyPoint)
{
    int rtn = 0;

    StringBuilder cmdText = new StringBuilder();

    cmdText.Append("Select Capacity from tbl_SupplyPoints where Location = '");
    cmdText.Append(supplyPoint);
    cmdText.Append("'");

    object capacity = base.ExecuteScalar(base.GetCommand(cmdText.ToString()));
    if (capacity != null)
        rtn = int.Parse(capacity.ToString());

    return rtn;
}

public void AssignCapacity(string supplyPoint, int cpty)
{
    StringBuilder cmdText = new StringBuilder();
```

```
        int capacity = this.GetCapacity(supplyPoint);

        if (capacity == 0)
        {
            cmdText.Append("Insert into tbl_SupplyPoints (Location, Capacity) values ('");
            cmdText.Append(supplyPoint);
            cmdText.Append("', '");
            cmdText.Append(cpty);
            cmdText.Append("')");
        }
        else
        {
            cmdText.Append("Update tbl_SupplyPoints set Capacity = ");
            cmdText.Append(cpty);
            cmdText.Append(" where Location = '");
            cmdText.Append(supplyPoint);
            cmdText.Append("'");
        }

        base.ExecuteNonQuery(base.GetCommand(cmdText.ToString()));
    }

}
public class TransportCostDAL : DALBase
{
    public TransportCostDAL() { }

    public float GetTransportCost(string supplyPoint, string demandPoint)
    {
        float rtn = 0.0F;

        StringBuilder cmdText = new StringBuilder();

        cmdText.Append("Select Cost from tbl_TransportCosts where SupplyPoint = '");
        cmdText.Append(supplyPoint);
        cmdText.Append("'");
        cmdText.Append(" and DemandPoint  = '");
        cmdText.Append(demandPoint);
        cmdText.Append("'");

        object cost = base.ExecuteScalar(base.GetCommand(cmdText.ToString()));
        if (cost != null)
            rtn = float.Parse(cost.ToString());
```

```
        return rtn;
    }

    public void AssignCost(string supplyPoint, string demandPoint, float cost)
    {
        StringBuilder cmdText = new StringBuilder();
        float cst = this.GetTransportCost(supplyPoint, demandPoint);

        if (cst == 0)
        {
            cmdText.Append("Insert into tbl_TransportCosts (SupplyPoint, DemandPoint, Cost) values ('");
            cmdText.Append(supplyPoint);
            cmdText.Append("', '");
            cmdText.Append(demandPoint);
            cmdText.Append("', '");
            cmdText.Append(cost);
            cmdText.Append("')");
        }
        else
        {
            cmdText.Append("Update tbl_TransportCosts set Cost = ");
            cmdText.Append(cost);
            cmdText.Append(" where SupplyPoint = '");
            cmdText.Append(supplyPoint);
            cmdText.Append("' and DemandPoint = '");
            cmdText.Append(demandPoint);
            cmdText.Append("'");
        }

        base.ExecuteNonQuery(base.GetCommand(cmdText.ToString()));
    }

  }//end class TransportCostDAL
}

namespace CocaCola
{
  public partial class mainForm : Form
  {
    TransportMatrix transportMatrix;

    TransportCellsCollection currPath;
```

```
}

public mainForm()
{
    InitializeComponent();
    this.transportMatrixTableLayoutPanel.RowStyles.Add(new RowStyle(SizeType.AutoSize));
}

private void mainForm_Load(object sender, EventArgs e)
{
    this.RefreshSupplyAndDemandPoints();
}

public void RefreshSupplyAndDemandPoints()
{
    this.supplyPointsCheckedListBox.Items.Clear();
    this.demandPointsCheckedListBox.Items.Clear();
    using (SupplyPointsDAL spDAL = new SupplyPointsDAL())
    {
        SupplyPointsCollection spc = spDAL.GetSupplyPoints();
        foreach (SupplyPoint sp in spc)
            this.supplyPointsCheckedListBox.Items.Add(sp);
    }

    using (DemandPointsDAL dpDAL = new DemandPointsDAL())
    {
        DemandPointsCollection dpc = dpDAL.GetDemandPoints();
        foreach (DemandPoint dp in dpc)
            this.demandPointsCheckedListBox.Items.Add(dp);
    }
}

private void calcButton_Click(object sender, EventArgs e)
{
    try
    {
        this.Reset();

        foreach (SupplyPoint sp in supplyPointsCheckedListBox.CheckedItems)
            transportMatrix.SupplyPoints.Add(sp);

        foreach (DemandPoint dp in demandPointsCheckedListBox.CheckedItems)
            transportMatrix.DemandPoints.Add(dp);
```

```
        transportMatrix.SetUpTransportMatrix();

        transportMatrix.GetInitialFeasibleSolution();

        transportMatrix.Display(this.transportMatrixTableLayoutPanel);

        if (this.transportMatrix.HasDegeneracy)
            throw new Exception("Degeneracy Encountered");

        this.label3.Text = transportMatrix.ShowHigestAllocatedAxis();

        //  transportMatrix.Axes.DemandAxes[2].Value = 0;

        this.transportMatrix.HighestAllocationAxis.Value = 0;

        while (!transportMatrix.Axes.AllAxesHasValues)
            transportMatrix.SolveForAxesValues();


        foreach (Axis ax in transportMatrix.Axes)
        {
            string text;
            if (ax.AxisType == AxisType.SupplyRow)
                text = "U";
            else
                text = "V";
            text += ax.Index.ToString() + "=" + ax.Value.ToString();
            this.listBox1.Items.Add(text);
        }

        transportMatrix.SolveForUnOccupiedCells();

        foreach (TransportCell tc in transportMatrix.TransportCells.UnAllocatedCells)
        {
            string text = "D(";
            text += tc.CellPosition.SupplyAxis.Index.ToString() + ", " + tc.CellPosition.DemandAxis.Index.ToString() + ") =
";
            text += tc.TmpCellAllocation.ToString();
            this.listBox2.Items.Add(text);
        }

        this.CheckForOptimality(false);

    }
```

```
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Northwest Corner Method");
        }
    }

    private void ClearLabels()
    {
        this.listBox1.Items.Clear();
        this.listBox2.Items.Clear();
        this.listBox3.Items.Clear();

        this.statusLabel.Text = "";

        this.label3.Text = "";
    }

    private void CheckForOptimality(bool rethrowException)
    {
        try
        {
            if (transportMatrix.IsOptimum)
                this.statusLabel.Text = "Solution Optimum";
            else
            {
                this.leastCostLabel.Text = this.transportMatrix.TransportCells.MostNegativeTmpAllocatedCell.ToString();

                try
                {
                    this.currPath =
this.transportMatrix.GetPath(this.transportMatrix.TransportCells.MostNegativeTmpAllocatedCell);
                    foreach (TransportCell tc in this.currPath)
                        this.listBox3.Items.Add(tc);
                    this.statusLabel.Text = "Not Optimal";
                }
                catch (PathNotFoundException)
                {
                    if (rethrowException)
                        throw;
                    MessageBox.Show("Cannot move forward");
                    this.statusLabel.Text = "Solution Optimum";
                }
            }
        }
```

```
        catch
        {
            throw;
        }
        finally
        {
            this.totalCostLabel.Text = this.transportMatrix.TransportCells.TotalCost.ToString();
        }
    }

    private void Reset()
    {
        this.transportMatrix = new TransportMatrix();

        this.ClearLabels();

        foreach (SupplyPoint sp in supplyPointsCheckedListBox.Items)
            sp.Capacity = sp.ConstCapacity;

        foreach (DemandPoint dp in demandPointsCheckedListBox.Items)
        {
            dp.Allocation = 0;
            dp.AllocationStatus = AllocationStatus.None;
        }
    }

    private void setupTransportCostsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        transportCostSetUpForm tcsf = new transportCostSetUpForm();
        tcsf.OnSupplyOrDemandChanged += new
transportCostSetUpForm.SupplyDemandChangedEventHandler(tcsf_OnSupplyOrDemandChanged);
        tcsf.Show();
        tcsf.SetupTabControl.SelectedTab = tcsf.TransportCosts;
    }
    void tcsf_OnSupplyOrDemandChanged(object sender, EventArgs e)
    {
        this.RefreshSupplyAndDemandPoints();
    }

    private void optimizeButton_Click(object sender, EventArgs e)
    {
        this.ClearLabels();

        foreach (Axis ax in this.transportMatrix.Axes)
```

```
    ax.Value = null;

//foreach (TransportCell tc in this.transportMatrix.TransportCells)
//    tc.TmpCellAllocation = 0.0F;

this.transportMatrix.PeformAddSubtractOperationOnPath(this.currPath);

transportMatrix.Display(this.transportMatrixTableLayoutPanel);

if (this.transportMatrix.HasDegeneracy)
    throw new Exception("Degeneracy Encountered");

this.label3.Text = transportMatrix.ShowHigestAllocatedAxis();

//  transportMatrix.Axes.DemandAxes[2].Value = 0;

this.transportMatrix.HighestAllocationAxis.Value = 0;

while (!transportMatrix.Axes.AllAxesHasValues)
    transportMatrix.SolveForAxesValues();


foreach (Axis ax in transportMatrix.Axes)
{
    string text;
    if (ax.AxisType == AxisType.SupplyRow)
        text = "U";
    else
        text = "V";
    text += ax.Index.ToString() + "=" + ax.Value.ToString();
    this.listBox1.Items.Add(text);
}

transportMatrix.SolveForUnOccupiedCells();

foreach (TransportCell tc in transportMatrix.TransportCells.UnAllocatedCells)
{
    string text = "D(";
    text += tc.CellPosition.SupplyAxis.Index.ToString() + ", " + tc.CellPosition.DemandAxis.Index.ToString() + ") = ";
    text += tc.TmpCellAllocation.ToString();
    this.listBox2.Items.Add(text);
}
```

```
            oRoutes.Show();
        }
    }

    void Optimize()
    {
        foreach (Axis ax in this.transportMatrix.Axes)
            ax.Value = null;

        this.transportMatrix.PeformAddSubtractOperationOnPath(this.currPath);

        if (this.transportMatrix.HasDegeneracy)
            throw new Exception("Degeneracy Encountered");

        this.transportMatrix.HighestAllocationAxis.Value = 0;

        while (!transportMatrix.Axes.AllAxesHasValues)
            transportMatrix.SolveForAxesValues();

        transportMatrix.SolveForUnOccupiedCells();

        this.CheckForOptimality(true);
    }

    private void setupSupplyPointsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        transportCostSetUpForm tcsf = new transportCostSetUpForm();
        tcsf.OnSupplyOrDemandChanged += new
transportCostSetUpForm.SupplyDemandChangedEventHandler(tcsf_OnSupplyOrDemandChanged);
        tcsf.Show();
        tcsf.SetupTabControl.SelectedTab = tcsf.SupplyPoints;
    }

    private void setupDemandPointsToolStripMenuItem_Click(object sender, EventArgs e)
    {
        transportCostSetUpForm tcsf = new transportCostSetUpForm();
        tcsf.OnSupplyOrDemandChanged += new
transportCostSetUpForm.SupplyDemandChangedEventHandler(tcsf_OnSupplyOrDemandChanged);
        tcsf.Show();
        tcsf.SetupTabControl.SelectedTab = tcsf.DemandPoints;
    }

    private void solveButton_Click(object sender, EventArgs e)
    {
```

```
        this.CheckForOptimality(false);

    }

    public void Solve()
    {
        try
        {
            this.Reset();

            foreach (SupplyPoint sp in supplyPointsCheckedListBox.CheckedItems)
                transportMatrix.SupplyPoints.Add(sp);

            foreach (DemandPoint dp in demandPointsCheckedListBox.CheckedItems)
                transportMatrix.DemandPoints.Add(dp);

            transportMatrix.SetUpTransportMatrix();

            transportMatrix.GetInitialFeasibleSolution();

            if (this.transportMatrix.HasDegeneracy)
                throw new Exception("Degeneracy Encountered");

            this.transportMatrix.HighestAllocationAxis.Value = 0;

            while (!transportMatrix.Axes.AllAxesHasValues)
                transportMatrix.SolveForAxesValues();

            transportMatrix.SolveForUnOccupiedCells();
            this.CheckForOptimality(true);

            while (true)
                this.Optimize();

        }
        catch (NoTransportCostException ex)
        {
            MessageBox.Show(ex.Message, "Northwest Corner Method");
        }
        catch (Exception)
        {
            OptimumRoutes oRoutes = new OptimumRoutes(this.transportMatrix.TransportCells.AllocatedCells);
            oRoutes.ShowInTaskbar = false;
```

```
            this.Solve();
        }

    }
    public partial class transportCostSetUpForm : Form
    {

        public delegate void SupplyDemandChangedEventHandler(object sender, EventArgs e);
        public event SupplyDemandChangedEventHandler OnSupplyOrDemandChanged;

        public transportCostSetUpForm()
        {
            InitializeComponent();
        }

        void RaiseSupplyDemandChangedEvent()
        {
            if (this.OnSupplyOrDemandChanged != null)
                this.OnSupplyOrDemandChanged(this, new EventArgs());
        }

        private void transportCostSetUpForm_Load(object sender, EventArgs e)
        {
            this.RefreshData();
        }

        void RefreshData()
        {
            this.supplyPointSetupComboBox.Items.Clear();
            this.supplyPointComboBox.Items.Clear();
            this.demandPointSetupComboBox.Items.Clear();
            this.demandPointComboBox.Items.Clear();

            SupplyPointsCollection spc = new SupplyPointsDAL().GetSupplyPoints();
            foreach (SupplyPoint sp in spc)
            {
                this.supplyPointSetupComboBox.Items.Add(sp.Name);
                this.supplyPointComboBox.Items.Add(sp.Name);
            }

            this.supplyPointSetupComboBox.SelectedIndex = 0;
            this.supplyPointComboBox.SelectedIndex = 0;
```

```
        DemandPointsCollection dpc = new DemandPointsDAL().GetDemandPoints();
        foreach (DemandPoint dp in dpc)
        {
            this.demandPointSetupComboBox.Items.Add(dp.Name);
            this.demandPointComboBox.Items.Add(dp.Name);
        }
        this.demandPointSetupComboBox.SelectedIndex = 0;
        this.demandPointComboBox.SelectedIndex = 0;
    }

    private void GetDemand()
    {
        this.demandCapacityUpdateTextBox.Text = new
DemandPointsDAL().GetDemand(this.demandPointSetupComboBox.Text).ToString();
    }

    private void GetCapacity()
    {
        this.supplyCapacityUpdateTextBox.Text = new
SupplyPointsDAL().GetCapacity(this.supplyPointSetupComboBox.Text).ToString();
    }
    private void GetCost()
    {
        float cost = new TransportCostDAL().GetTransportCost(this.supplyPointComboBox.Text,
this.demandPointComboBox.Text);
        this.costTextBox.Text = cost.ToString();
    }

    private void supplyPointComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.GetCost();
    }

    private void demandPointComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.GetCost();
    }

    private void assignButton_Click(object sender, EventArgs e)
    {
        string supplyPoint = this.supplyPointComboBox.Text;
        string demandPoint = this.demandPointComboBox.Text;
        float cost = float.Parse(this.costTextBox.Text);
        new TransportCostDAL().AssignCost(supplyPoint, demandPoint, cost);
        MessageBox.Show("Cost assigned successfully");
```

```
}

public TabPage SupplyPoints
{
    get { return this.supplyPointsTabPage; }
}

public TabPage DemandPoints
{
    get { return this.demandPointsTabPage; }
}

public TabControl SetupTabControl
{
    get { return this.tabControl1; }
}
public TabPage TransportCosts
{
    get { return this.setupCostsTabPage; }
}

private void addSupplyButton_Click(object sender, EventArgs e)
{
    string supplyPoint = this.supplyPointTextBox.Text;
    int capacity = int.Parse(this.supplyCapacityNewTextBox.Text);
    new SupplyPointsDAL().AssignCapacity(supplyPoint, capacity);
    MessageBox.Show("Supply point added successfully");
    this.RefreshData();
    this.RaiseSupplyDemandChangedEvent();
}

private void updateSupplyButton_Click(object sender, EventArgs e)
{
    string supplyPoint = this.supplyPointSetupComboBox.Text;
    int capacity = int.Parse(this.supplyCapacityUpdateTextBox.Text);
    new SupplyPointsDAL().AssignCapacity(supplyPoint, capacity);
    MessageBox.Show("Supply point capacity updated successfully");
    this.RefreshData();
    this.RaiseSupplyDemandChangedEvent();
}

private void addDemandButton_Click(object sender, EventArgs e)
{
```

```
        string demandPoint = this.demandPointTextBox.Text;
        int demand = int.Parse(this.demandCapacityNewTextBox.Text);
        new DemandPointsDAL().AssignDemand(demandPoint, demand);
        MessageBox.Show("Demand point added successfully");
        this.RefreshData();
        this.RaiseSupplyDemandChangedEvent();
    }

    private void updateDemandButton_Click(object sender, EventArgs e)
    {
        string demandPoint = this.demandPointSetupComboBox.Text;
        int demand = int.Parse(this.demandCapacityUpdateTextBox.Text);
        new DemandPointsDAL().AssignDemand(demandPoint, demand);
        MessageBox.Show("Demand point updated successfully");
        this.RefreshData();
        this.RaiseSupplyDemandChangedEvent();
    }

    private void demandPointSetupComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.GetDemand();
    }

    private void supplyPointSetupComboBox_SelectedIndexChanged(object sender, EventArgs e)
    {
        this.GetCapacity();
    }

    private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (this.tabControl1.SelectedTab == this.SupplyPoints)
            this.GetCapacity();
        if (this.tabControl1.SelectedTab == this.DemandPoints)
            this.GetDemand();
        if (this.tabControl1.SelectedTab == this.TransportCosts)
            this.GetCost();
    }
}
public partial class OptimumRoutes : Form
{
    public OptimumRoutes(TransportCellsCollection allocatedCells)
    {
        InitializeComponent();
```

```
    ArrayList validCells = new ArrayList();
    //filter out the dummy cells
    foreach (TransportCell tc in allocatedCells)
    {
        if (tc.SupplySource.Name != "Dummy" && tc.DemandSource.Name != "Dummy")
            validCells.Add(tc);
    }
    this.TransportCellBindingSource.DataSource = validCells;
}

private void OptimumRoutes_Load(object sender, EventArgs e)
{
    this.reportViewer1.RefreshReport();
}
```

Questionnaire for CocaCola Kaduna Plc

Surname _____
Last Name _____
Phone _____
Job Title _____

What time do you start work? _____

What time do you close from work? _____

Please give a brief description of your job role

_____
_____
_____
_____
_____

How many different types of truck do you have?

_____
_____
_____
_____
_____
_____

Which areas do you deliver to?

_____
_____
_____
_____
_____
_____

Please give an estimate average of crates you deliver in a day

_____
_____
_____