Network Flow Module Documentation

Table Of Contents

A) Directory Structure

NetworkFlow_Module	
• bin	→ executable files
• build	→ generated .o files
• docs	→ documentation and image files
• input	→ input files for programs
• mod	→ generated .mod files
• output	→ generated output files
• src	→ module source files and test program files
• makefile	

B) Program Operation Guide

The module consists of 5 test files and 13 source files. The easiest way to use the module is through the Makefile. Typing “make” at the command line will result in the mod, build and bin folders being populated with the generated “.mod”, “.o” and executable files respectively. The output of the make command is as shown below.

```
File  Edit  View  Search  Terminal  Help
abhishek@abhishek-Inspiron-5559:~/CDAC/NetworkFlow_Module$ make
gfortran -o ./bin/graph ./build/testgraph.o ./build/GraphMod.o ./build/GraphUserInterface.o -J./mod -I./mod
gfortran -c -o ./build/testnetwork.o ./src/testnetwork.f90 -J./mod -I./mod
gfortran -o ./bin/network  ./build/testnetwork.o ./build/GraphMod.o ./build/GraphUserInterface.o ./build/NetworkFlowMod.o  ./build/NetworkUserInterfac
e.o ./build/NetworkAMPLInterface.o -J./mod -I./mod
gfortran -o ./bin/wvnetwork ./build/testwvnflow.o ./build/GraphMod.o ./build/GraphUserInterface.o  ./build/NetworkFlowMod.o  ./build/NetworkUserInterf
ace.o  ./build/NetworkAMPLInterface.o ./build/WeightedVertexNetworkFlowMod.o ./build/WeightedVertexNetworkUserInterface.o -J./mod -I./mod
gfortran -c -o ./build/map_module.o ./src/map_module.f90 -J ./mod -I ./mod
gfortran -c -o ./build/time_zone_module.o ./src/time_zone_module.f90 -J ./mod -I ./mod
gfortran -c -o ./build/activity_module.o ./src/activity_module.f90 -J ./mod -I ./mod
gfortran -c -o ./build/affinity_module.o ./src/affinity_module.f90 -J ./mod -I ./mod
gfortran -c -o ./build/NetworkTrafficModelInterface.o ./src/NetworkTrafficModelInterface.f90 -J./mod -I./mod
gfortran -c -o ./build/testtraffic.o ./src/testtraffic.f90 -J./mod -I./mod
gfortran -o ./bin/trafficnetwork ./build/testtraffic.o ./build/GraphMod.o ./build/NetworkFlowMod.o ./build/NetworkTrafficModelInterface.o ./build/Netw
orkAMPLInterface.o ./build/map_module.o ./build/time_zone_module.o ./build/activity_module.o ./build/affinity_module.o -J./mod -I./mod
abhishek@abhishek-Inspiron-5559:~/CDAC/NetworkFlow_Module$
```

The make file is printed below for the reference of the user.

```
#makefile

#FC = gfortran-8

FC = gfortran

SRC_DIR = ./src

BUILD_DIR = ./build

BIN_DIR = ./bin

MOD_DIR = ./mod

OUT_DIR = ./output


all: graph network wvnetwork trafficnetwork


$(BUILD_DIR)/GraphMod.o: $(SRC_DIR)/GraphMod.f90

	$(FC) -c -o $(BUILD_DIR)/GraphMod.o $(SRC_DIR)/GraphMod.f90 -J$(MOD_DIR) -I$(MOD_DIR)


$(BUILD_DIR)/GraphUserInterface.o: $(BUILD_DIR)/GraphMod.o

	$(FC) -c -o $(BUILD_DIR)/GraphUserInterface.o $(SRC_DIR)/GraphUserInterface.f90 -J$(MOD_DIR) -I$(MOD_DIR)


$(BUILD_DIR)/NetworkFlowMod.o: $(BUILD_DIR)/GraphMod.o

	$(FC) -c -o $(BUILD_DIR)/NetworkFlowMod.o $(SRC_DIR)/NetworkFlowMod.f90 -J$(MOD_DIR) -I$(MOD_DIR)


$(BUILD_DIR)/NetworkUserInterface.o: $(BUILD_DIR)/NetworkFlowMod.o

	$(FC) -c -o $(BUILD_DIR)/NetworkUserInterface.o $(SRC_DIR)/NetworkUserInterface.f90 -J$(MOD_DIR) -I$(MOD_DIR)


$(BUILD_DIR)/NetworkAMPLInterface.o: $(BUILD_DIR)/NetworkFlowMod.o
```

```
	$(FC) -c -o $(BUILD_DIR)/NetworkAMPLInterface.o $(SRC_DIR)/NetworkAMPLInterface.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/NetworkTrafficModelInterface.o : $(BUILD_DIR)/NetworkFlowMod.o math_model
	$(FC) -c -o $(BUILD_DIR)/NetworkTrafficModelInterface.o $(SRC_DIR)/NetworkTrafficModelInterface.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/WeightedVertexNetworkFlowMod.o: $(BUILD_DIR)/NetworkUserInterface.o
	$(FC) -c -o $(BUILD_DIR)/WeightedVertexNetworkFlowMod.o $(SRC_DIR)/WeightedVertexNetworkFlowMod.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/WeightedVertexNetworkUserInterface.o: $(BUILD_DIR)/WeightedVertexNetworkFlowMod.o
	$(FC) -c -o $(BUILD_DIR)/WeightedVertexNetworkUserInterface.o $(SRC_DIR)/WeightedVertexNetworkUserInterface.f90 -J$(MOD_DIR) -I$(MOD_DIR)

#math_model: $(SRC_DIR)/map_module.f90 $(SRC_DIR)/time_zone_module.f90 $(SRC_DIR)/activity_module.f90 $(SRC_DIR)/affinity_module.f90
#	$(FC) -c -o $(BUILD_DIR)/math_model.o $(SRC_DIR)/map_module.f90 $(SRC_DIR)/time_zone_module.f90 $(SRC_DIR)/activity_module.f90 $(SRC_DIR)/affinity_module.f90 -J$(MOD_DIR) -I$(MOD_DIR)

math_model: affinity_module

affinity_module: activity_module
	$(FC) -c -o $(BUILD_DIR)/affinity_module.o $(SRC_DIR)/affinity_module.f90 -J $(MOD_DIR) -I $(MOD_DIR)

activity_module: time_zone_module
```

```
	$(FC) -c -o $(BUILD_DIR)/activity_module.o $(SRC_DIR)/activity_module.f90 -J $(MOD_DIR) -I $(MOD_DIR)

time_zone_module: map_module
	$(FC) -c -o $(BUILD_DIR)/time_zone_module.o $(SRC_DIR)/time_zone_module.f90 -J $(MOD_DIR) -I $(MOD_DIR)

map_module:
	$(FC) -c -o $(BUILD_DIR)/map_module.o $(SRC_DIR)/map_module.f90 -J $(MOD_DIR) -I $(MOD_DIR)

$(BUILD_DIR)/testgraph.o: $(SRC_DIR)/testgraph.f90 $(BUILD_DIR)/GraphMod.o $(BUILD_DIR)/GraphUserInterface.o
	$(FC) -c -o $(BUILD_DIR)/testgraph.o $(SRC_DIR)/testgraph.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/testnetwork.o: $(SRC_DIR)/testnetwork.f90 $(BUILD_DIR)/NetworkFlowMod.o $(BUILD_DIR)/NetworkUserInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o
	$(FC) -c -o $(BUILD_DIR)/testnetwork.o $(SRC_DIR)/testnetwork.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/testtraffic.o: $(SRC_DIR)/testtraffic.f90 $(BUILD_DIR)/NetworkTrafficModelInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o $(BUILD_DIR)/NetworkFlowMod.o
	$(FC) -c -o $(BUILD_DIR)/testtraffic.o $(SRC_DIR)/testtraffic.f90 -J$(MOD_DIR) -I$(MOD_DIR)

$(BUILD_DIR)/testwvnflow.o: $(SRC_DIR)/testwvnflow.f90 $(BUILD_DIR)/WeightedVertexNetworkFlowMod.o $(BUILD_DIR)/WeightedVertexNetworkUserInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o
```

```
	$(FC) -c -o $(BUILD_DIR)/testwvnflow.o $(SRC_DIR)/testwvnflow.f90 -J$(MOD_DIR) -I$(MOD_DIR)


graph: $(BUILD_DIR)/testgraph.o
	$(FC) -o $(BIN_DIR)/graph $(BUILD_DIR)/testgraph.o $(BUILD_DIR)/GraphMod.o $(BUILD_DIR)/GraphUserInterface.o -J$(MOD_DIR) -I$(MOD_DIR)


network: $(BUILD_DIR)/testnetwork.o
	$(FC) -o $(BIN_DIR)/network $(BUILD_DIR)/testnetwork.o $(BUILD_DIR)/GraphMod.o $(BUILD_DIR)/GraphUserInterface.o $(BUILD_DIR)/NetworkFlowMod.o $(BUILD_DIR)/NetworkUserInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o -J$(MOD_DIR) -I$(MOD_DIR)


wvnetwork: $(BUILD_DIR)/testwvnflow.o
	$(FC) -o $(BIN_DIR)/wvnetwork $(BUILD_DIR)/testwvnflow.o $(BUILD_DIR)/GraphMod.o $(BUILD_DIR)/GraphUserInterface.o $(BUILD_DIR)/NetworkFlowMod.o $(BUILD_DIR)/NetworkUserInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o $(BUILD_DIR)/WeightedVertexNetworkFlowMod.o $(BUILD_DIR)/WeightedVertexNetworkUserInterface.o -J$(MOD_DIR) -I$(MOD_DIR)


trafficnetwork : $(BUILD_DIR)/testtraffic.o
	$(FC) -o $(BIN_DIR)/trafficnetwork $(BUILD_DIR)/testtraffic.o $(BUILD_DIR)/GraphMod.o $(BUILD_DIR)/NetworkFlowMod.o $(BUILD_DIR)/NetworkTrafficModelInterface.o $(BUILD_DIR)/NetworkAMPLInterface.o $(BUILD_DIR)/map_module.o $(BUILD_DIR)/time_zone_module.o $(BUILD_DIR)/activity_module.o $(BUILD_DIR)/affinity_module.o -J$(MOD_DIR) -I$(MOD_DIR)



clean:
```

$(RM) $(MOD_DIR)/*.mod

$(RM) $(BUILD_DIR)/*.o

$(RM) $(BIN_DIR)/graph

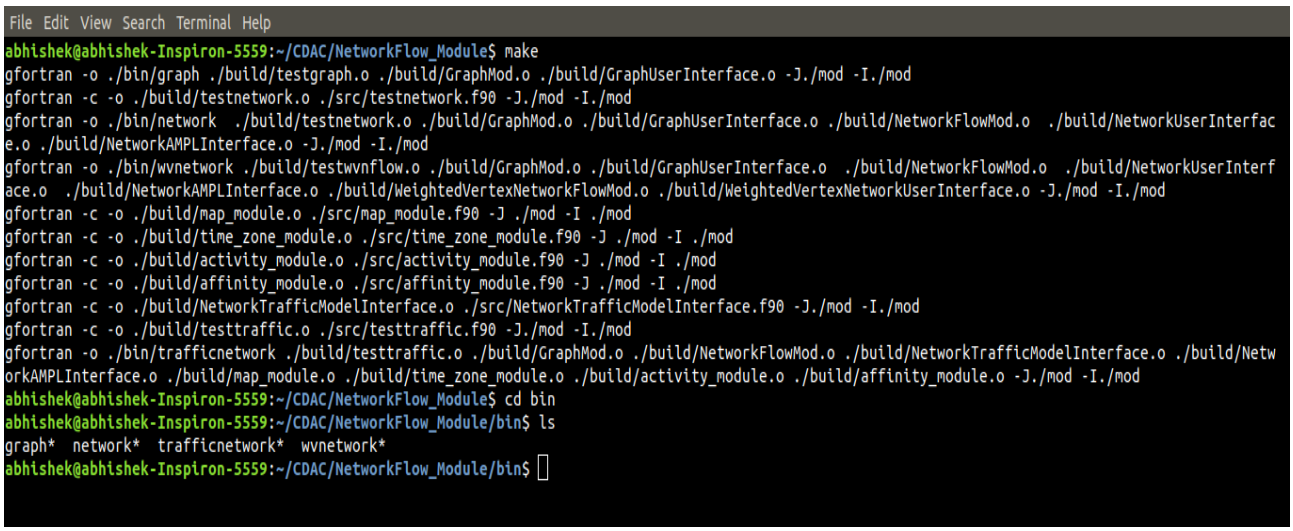
$(RM) $(BIN_DIR)/network

$(RM) $(BIN_DIR)/wvnetwork

$(RM) $(BIN_DIR)/trafficnetwork

$(RM) $(OUT_DIR)/AMPLInput.dat

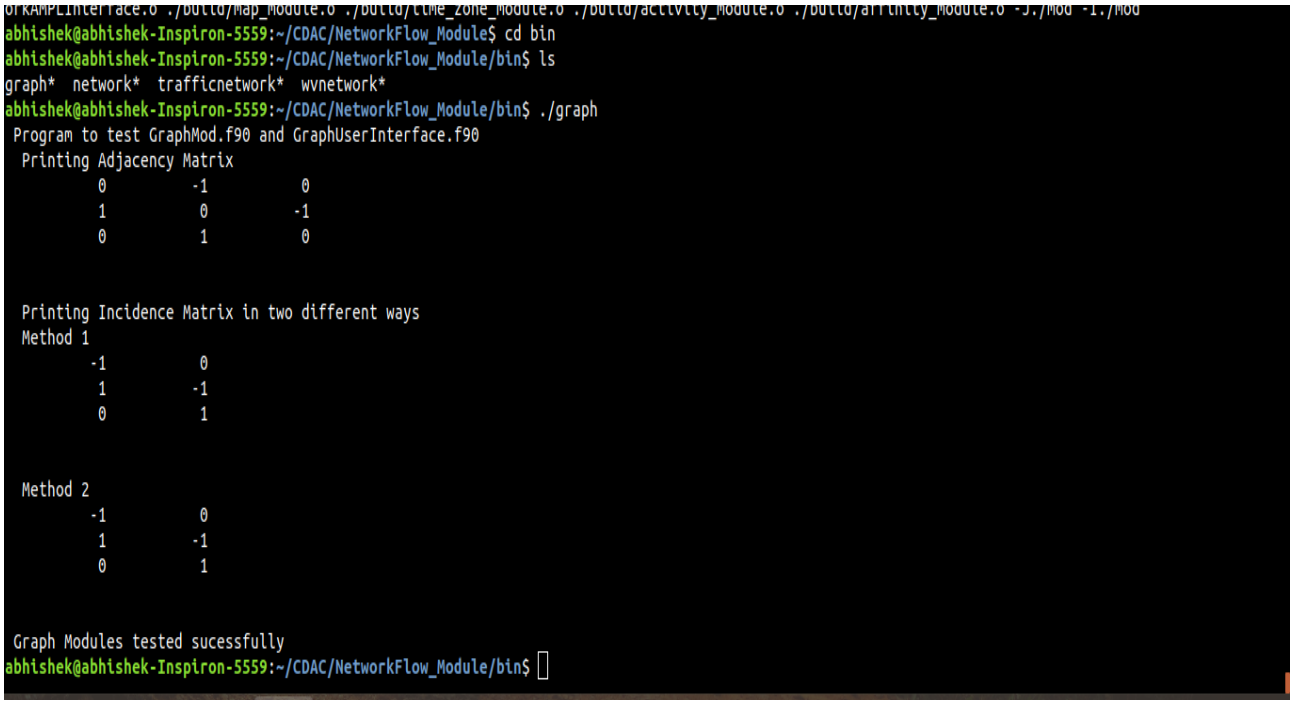
In order to run the generated executable files, navigate to the "bin" folder on the command line using the "**cd"** command.



There will exist 4 executable files in the bin directory. They are:

- graph ⟶ tests all subroutines belonging to the Graph Modules
- network ⟶ tests all subroutines belonging to Network Module and all its interfaces
- wvnetwork ⟶ tests all subroutines belonging to Weighted Vertex Network and its interfaces
- trafficnetwork ⟶ tests all subroutines belonging to the Traffic Model Interface

Execution of any one of these executables is carried out by typing "./exec-name" on the terminal and pressing Return.



C) Libary Description

The overall aim of this library is to serve as an interface between telecom data and Linear Programming based optimization software such as AMPL or NEOS. The input data to this library can be split into two types, data describing the underlying network and data describing the specific problem to be optimized. For example, data of the first type will initialize the edges of the Graph along with the cost and capacities associated with these edges while data of the second kind will initialize the amount of flow (telecom data in this example) between a pair of nodes.

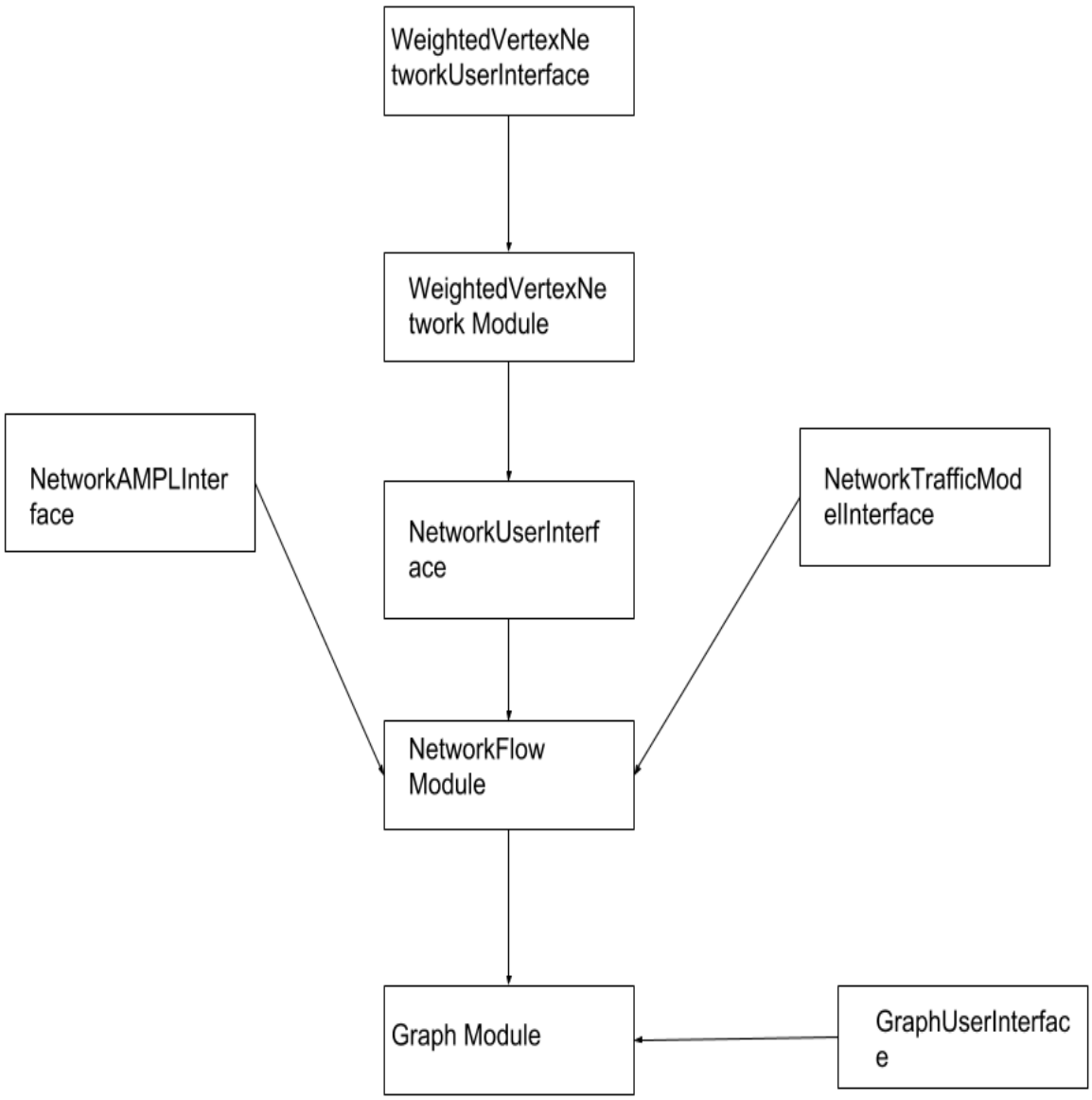
The central module in this library is the NetworkFlow Module. It contains the Network Derived type which serves as a template using which the user can model the Network-Flow problem. The NetworkAMPLInterface can be used to link the NetworkFlow Module with any solver that accepts AMPL files as input. (See Appendix A for a brief description of the AMPL language).

The NetworkTrafficModelInterface is an interface between the work done by the Math Model group and the Network Flow Module. It allows the user to convert input files prepared in accordance with the Math Model standards, to objects of the Network Derived type. These objects can then be converted into AMPL data files using the AMPL Interface.

The WeightedVertexNetworkModule is a simple extension of the Network Module in which each Network node has a weight associated with it. This Module also contains subroutines which allow an object of the WeightedVertexNetwork derived type to be converted to a regular Network object. (See Appendix C for a detailed expanation of how this is achieved.)

The Graph Module serves as a base to the modules mentioned above and was the starting point of this library. The various User Interface Modules contain subroutines to take input from and provide output to the user.

D) Module Dependency Diagram



E) Modular Description

1. Graph Module

This Module provides the underlying Graph Derived Type along with subroutines that allow access to Graph Properties. This module contains:

Type	**Name**	Description
Derived Type	Graph	Is the basic structure that holds a graph. It contains variables holding the number of nodes, edges and whether it is directed or not. It also contains a list of all the edges in the graph.
function	adjacency()	Returns the adjacency matrix of the given Graph
function	incidence()	Returns the incidence matrix of the given Graph
subroutine	adjacencyMatrix()	Performs the same function as adjacency()
subroutine	incidenceMatrix()	Performs the same function as incidence()
operator	“ = ”	The assignment operator has been overloaded with the incidenceMatrix() subroutine. Therefore assigning a Graph to a Matrix will cause the Matrix the be equal to the incidence matrix of the Graph
subroutine	confirmNoSelfLoop()	Given a graph this subroutine checks whether there are self loops present in the Graph. If presents it displays an error message and exits.

2. GraphUserInterface Module

The GraphUserInterface Module contains subroutines that allow the user to initialize a Graph Derived Type from an input file. It is an interface between the user and the Graph Module.

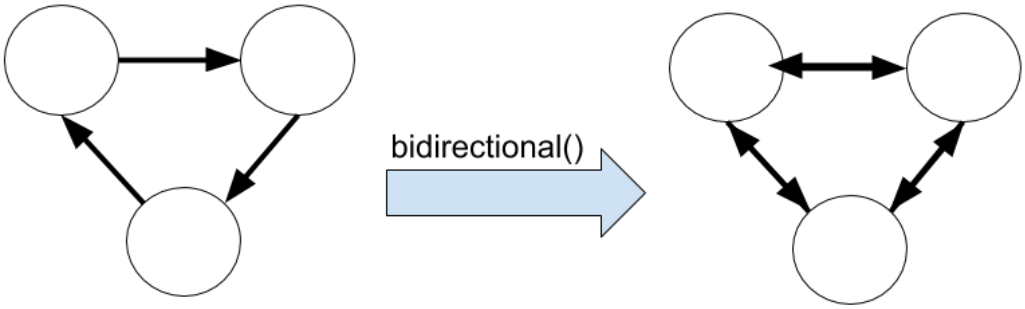
Type	**Name**	Description

subroutine	readGraphData()	This subroutine takes a filename as input and populates a Graph(also taken as input) using the values in the file. The structure of the file is described in the input file section below.
subroutine	printMatrix()	This subroutine accepts a 2-D array as input and prints it on the terminal.

3. NetworkFlow Module

The NetworkFlow Module contains the derived type Network which is the representation of a physical network. The setSourceDestinationFlow() subroutine in this module is to be used in conjunction with the readSourceDestinationFlow() subroutine in the NetworkUserInterface Module in order to initialize the VertexFlow array present in the Network derived type.

Type	Name	Description
Derived type	Network	Represents a real life network. Contains the following: 1. Graph Derived Type to hold the nodes and vertices. (Directed Graph by default) 2. CostVector array to hold the cost associated with each edge 3. CapacityVector array to hold the capacity associated with each edge 4. FlowVector array to hold the flow through each edge. (Currently unused) 5. VertexFlow array to hold the net flow associated with each node. (Non-zero only at sources and sinks)
subroutine	setSourceDestinationFlow()	This subroutine accepts a 2-D array as input and initializes the sources ,sinks and flow for each commodity in the Network.
subroutine	bidirectional()	This subroutine converts a regular directed Network into a bidirectional Network. It is necessary to do this for every Network before calling printAMPLDataFile(). This is because a bidirectional network allows the optimization problem to remain linear. See the diagram below for a better understanding of what a bidirectional network is.



4. NetworkUserInterface Module

This Module serves as an interface between the user and the Network derived type. This module contains subroutines in order to read data about the Network derived type. The readNetworkData() subroutine reads data from an input file and populates the edges, costVector and capacityVector arrays of the Network derived type. In essence it reads data pertaining to the structure of the Network from the input file. The readSourceDestinationFlowData() subroutine on the other hand reads data regarding sources, destinations and the flow between them from the input file. It then populates a 2-D array which can be used as input to setSourceDestinationFlow() in the previous module.

Type	**Name**	Description
subroutine	readNetworkData()	Reads the input file provided and initializes the Network.
subroutine	readSourceDestinationFlowData()	This subroutine reads a given input file and initializes a 2-D array with this information. This is to be used in conjunction with setSourceDestinationFlow()

5. NetworkAMPLInterface Module

This module serves as an interface between this library and any solver that accepts AMPL files as input. The printAMPLDataFile() subroutine produces a data file which can be fed as input to AMPL (Read Appendix A for more details)

Type	**Name**	Description
subroutine	printAMPLDataFile()	Given a Network this subroutine prepares a data file for AMPL which can be used in conjunction with

		ampl-networkflow.mod.

6. NetworkTrafficModelInterface Module

This module serves as an interface between the Math-Model library and this one. The createNetwork() subroutine allows the creation of a Network directly from an input file built in accordance with the standards of the Math-Model library. The procedure by which this is done is as follows:

1) The read_data() subroutine present in the map_module of the Math-Model is called in order to parse through the input file.
2) Thus the number of nodes in the Graph are obtained along with other metrics such as the population ,longitude ,latitude and language code.
3) The createEdges() subroutine is called. This subroutine updates the number of nodes in the Graph to the number of cities plus the number of intermediate nodes.
4) These intermediate nodes are assigned random latitudes and longitudes.
5) Edges are created within the Network. The criteria for an edge between two nodes is that the geographical distance between the two nodes must be less than the value of coverage_dist.
6) The distance_intermediate() function returns the geographical distance between any two cities
7) Next the fillVertexFlow() subroutine allocates and initializes the vertexFlow array of the Network derived type.
8) The flow between every pair of cities is treated as an unique commodity.
9) The amount of flow between a pair of cities is equal to the number of telephone calls between the two cities.
10) Lastly the fillCostCapacity() subroutine allocates and initializes the costVector and capacityVector arrays of the Network. Currently all the edges in the Network have been initialized with globally defined, constant values. (However this must change in the future).

Type	**Name**	Description
Parameter (Integer)	time_interval coverage_dist intermediate_nodes	Time interval of the day (Refer Math-Model docs). The radius within which two nodes have a direct link between them. Number of intermediate nodes introduced into the network.
Parameter (Real)	cost capacity	Cost associated with individual links Capacity associated with individual links (Both the above quantities are constant for all links)
Parameter (Real)	lat_upper_limit	The upper and lower limits of the latitudes and longitudes covered by India, approximated to the

	lon_upper_limit lat_lower_limit lon_lower_limit	nearest integer.
subroutine	createNetwork()	Given an input file in the format prescribed by the Math-Model, this subroutine initializes a Network with the data in the file. This subroutine calls three other subroutines, createEdges(), fillVertexFlow() and fillCostCapacity().
subroutine	createEdges()	This subroutine initializes an edge between all pairs of cities which lie within the coverage distance. (coverage_dist)
subroutine	fillVertexFlow()	This subroutine initializes the VertexFlow array present in the Network Derived Type. The flow between any two pairs of cities is equal to the number of phone calls between these two cities.
subroutine	fillCostCapacity()	This subroutine initializes the cost and capacity vectors of the Network with the globally defined cost and capacity values.
function	distance_intermediate()	This function returns the distance between two cities (or intermediate nodes) . It internally calls the distance() function present in Math Model.

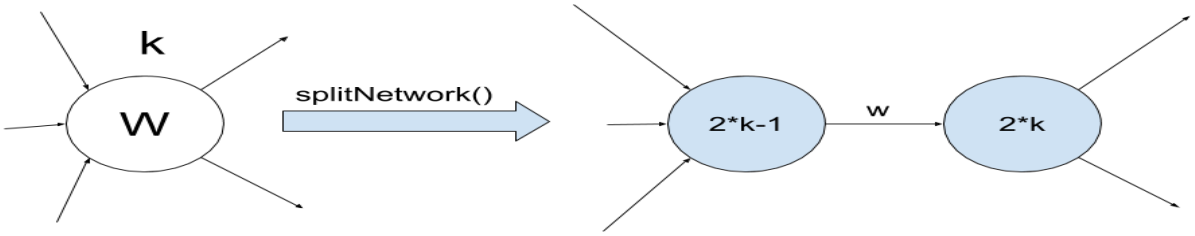
7. WeightedVertexNetworkFlow Module

This module accomodates Networks with weighted vertices. It contains of a derived data type called WeightedVertexNetwork. This Module also contains subroutines which allow an object of the WeightedVertexNetwork derived type to be converted to a regular Network object. (See Appendix C for a detailed expanation of how this is achieved.)

Type	**Name**	Description
Derived type	WeightedVertexNetwork	Consists of a Network along with an extra array to hold the vertex weights.
operator	“ = “	The assignment operator is overloaded. It internally calls splitNetwork() to convert a weighted Network to a regular Network.

subroutine	readWeighte dVertexNetw orkData()	This subroutine reads data from an input file and initializes a WeightedVertexNetwork.
subroutine	splitNetwork ()	This subroutine takes a WeightedVertexNetwork as input and returns an instance of the Network Derived Type. (This conversion is explained in Appendix A)
subroutine	setWeighted NetworkSour ceDestination Flow()	This subroutine reads a given input 2-D array and initializes the sources ,sinks and flow for each commodity in the WeightedVertexNetwork.

The diagram below illustrates the function of the splitNetwork() subroutine.



8. WeightedVertexNetworkUserInterface Module

This module allows the user to conveniently initialize a Weighted Vertex Network using an input file. This module is similar in structure to the NetworkUserInterface Module and contains the same subroutines. The only significant difference being that these operate on Weighted Vertex Networks.

Type	**Name**	Description
subroutine	readWeighted VertexNetwor kData()	Reads the input file provided and initializes the Weighted Vertex Network.
subroutine	readWeighted NetworkSourc	This subroutine reads a given input file and initializes a 2-D array with this information. This is to be used

	eDestinationFlow()	in conjunction with setSourceDestinationFlow()

F) Library Usage

This library is to be coupled with a linear programming based solver such as the AMPL software or the NEOS web resource. A linear programming problem can be fed as input into the library in the following two ways.

- Using the input files NetworkInput.dat, WeightedVertexNetworkInput.dat and FlowInput.dat in the input directory of this folder. The user can replace the data in these files in accordance with the input formats.
- Using the input.txt file used by the Math-Model library and the NetworkTrafficModelInterface module. In this case the user may wish to modify the intermediate_nodes parameter present in the module to control the number of intermediate nodes added.

Once the data has been read by the library, the user can use the NetworkAMPL Interface in order to create a data file for input to AMPL. This file is by default created in the output directory of the library. (For more details regarding how to use this file with AMPL, refer Appendix A).

G) Source Code and Input Files

Please refer source_code.pdf document present in the docs folder of the Library.

H) Appendix A – AMPL

- AMPL (A Mathematical Programming Language) is an algebric modelling language to describe and solve large scale mathematical computing problems (1). The AMPL software is a set of tools that can be used to both describe and solve optimization problems.

- The limited version of the AMPL software is freely available for download at https://ampl.com/products/ampl/ampl-for-students/

- The AMPL book is the official guide for AMPL and can be accessed online at https://ampl.com/resources/the-ampl-book/

- Using AMPL requires the user to create three files which are given as input to the software. These files are the model file, data file and the command file.

- The model file describes your linear programming model using the AMPL language. It consists of parameters, sets, variables, constraints and objective functions.

    1. Parameter : similar to a parameter in Fortran i.e. a constant.
    2. Sets : a group of related entities that occur together. For example a set of cities.
    3. Variables : The entities that can be changed by the solver in order to optimize the objective function. In a Network-Flow problem this is the flow itself.
    4. Objective Function : The quantity that is to be optimized.
    5. Constraints : Conditions that the solver must satisfy while optimizing the objective function.

Please refer ampl-networkflow.mod for an example of how to write model files for AMPL.
The AMPLInterface for NetworkFlow creates an output file that is by default named "AMPLInput.dat" . This file is to be used as the data file (.dat) for AMPL. The command file is a file which contains instructions to AMPL. This file tells AMPL which model to run, from where to pick up the data and which variables to be displayed as output. See exec.run in the source directory for an example.

I) Appendix B – NEOS Web Resource

The NEOS Server is a free internet-based service for solving numerical optimization problems.
There are multiple ways to submit a job to NEOS. For a list of all the possible methods please visit : https://neos-guide.org/content/users-guide

The method described in this document is the Web Interface method.

Steps to submit a job to NEOS:

1) Create the AMPL model (.mod) and data (.dat) file.

2) Create a AMPL command (.run) file. (This file should not contain the model modelname.mod and data dataname.dat commands. These are executed automatically)

3) Go to https://neos-server.org/neos/solvers/index.html .(This page contains a list of solvers that NEOS provides. The solver that we want to use for our problem is called MINOS. It is useful for nonlinearly constrained optimization problems.)

4) Go to the "Solver" tab and click on the letter M. Select the AMPL input format for the MINOS solver from the drop-down menu.

5) Scroll down to the bottom of the page. There should be a web submission form. Using the browse buttons present, select your model, data and command file.

6) Add comments (if you wish), enter a valid email-id and click the Submit to NEOS button.

7) After your job runs, the page should get reloaded automatically to show the results. If the results of your program are too huge to display on the screen, NEOS will send you an E-mail with the results.

J) Appendix C – Weighted Vertex Network to Network

The WeightedVertexNetwork is converted to a regular Network in the following manner:
Each Node in the Network is split into two nodes which are joined by an edge with infinite capacity and cost equal to the cost associated with the node. The new nodes generated have no cost associated with them.
All edges that are incident onto the original node are mapped to one of the newly generated nodes and all edges originating from the original node now originate from the other newly generated node.
This operation causes the number of nodes in the original network to double. A node which originally had rank k is mapped to nodes with ranks 2*k-1 and 2*k.
The number of edges in the Network increase by an amount equal to the number of nodes. One extra edge is created for every node that is split.