ABSTRACT

XIANG, LING. Performance Model for a Public Logistics Network. (Under the direction of Michael G. Kay.)

A public logistics network (PLN) has been proposed as an alternative to private networks for the ground transport of parcels. In this dissertation, a heuristic approach to approximate the package average waiting time in a PLN is presented; and then based on this waiting time approximation, a PLN design procedure is developed.

A PLN can be viewed as a priority queuing network with bulk arrivals and bulk service. It is difficult to obtain a closed-form solution for package average waiting time in a PLN. The problem is reformulated so that trucks transport loads instead of individual packages, thereby relaxing the bulk arrivals and bulk service feature. The package average waiting time is approximated fairly accurately by Kingman's equation when the server utilization is high.

A simulation model is created to determine the parameters needed in Kingman's equation (the coefficient of variation for package interarrival times and for truck interarrival times). A regression analysis of the results shows that the headway ratio is around 5.5. The package average waiting time is approximated by the product of the truck headway (truck average interarrival time) and the headway ratio.

The PLN simulation with protocols and the package bidding process is discussed as an extension to the basic simulation model. Packages bid for their trips along the way. The highest bidder gets the highest priority for truck transport services. Results from a simulation model incorporating a series of protocols developed by Kay show that a PLN with these protocols performs better than a FIFO system.

For the PLN design problem, the goal is to design a PLN that results in the minimal package average waiting time for the entire network. Potential locations (search space) for the distribution centers (DCs) are found using the U.S. network of interstate and highways. A genetic algorithm (GA) was applied to search for the optimal number of DCs, their locations, and direct arc connections between each pair of DCs.

Performance Model for a Public Logistics Network

by
Ling Xiang

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Industrial and Systems Engineering

Raleigh, North Carolina

2009

APPROVED BY:

DEDICATION

This dissertation is dedicated to my mother and father. I am greatly indebted to them for their love and support in all I am pursuing.

BIOGRAPHY

Ling Xiang was born in December, 1978 in Yunyang, Chongqing, China. He completed his schooling at Wuhan, China. He earned a bachelor's degree in automation science and engineering from Huazhong University of Science and Technology, Wuhan, China.

He came to North Carolina State University for graduate school in the fall of 2004. He studied industrial and systems engineering, focusing on logistics networks and production systems engineering under the supervision of Dr. Michael Kay.

ACKNOWLEDGMENTS

I would like to thank the following people for their generous and continued support for this work:

- Dr. Michael Kay served as my advisor and committee chair. He spent an enormous amount of time and effort on this project. He guided and encouraged me through hard times along the way. Without him, I could not have finished this work. He has been very supportive of my extracurricular activities as well.

- Dr. Russell King, Dr. James Wilson, and Dr. Tao Pang served as my committee members, and I thank them for their strong encouragement and feedback along the way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Introduction

A public logistics network (PLN) has been proposed as a fast and low-cost alternative to private logistics networks for the ground transport of parcels [1, 2]. In contrast to UPS or FedEx, the resources in a PLN are owned by multiple companies instead of a single company. Each of these companies functions totally on its own. For example, some companies may own some trucks, and other companies may own one distribution center (DC) or even part of a DC. Unlike private logistics networks such as UPS or FedEx, which require a huge capital investment to operate, this structure allows even small companies with limited capital to do business in a PLN, and to compete against each other [3, 4].

There is a similarity between the packages transported in the PLN and the data packets transmitted through the Internet. In a PLN, a package is sent from a store or a warehouse and then hops through a sequence of public DCs which are mostly located in metropolitan areas; and finally it is delivered to a home in a matter of hours [1]. The DCs, functioning like routers in the Internet, can also be located at major highway interchanges to avoid packages being transported through local city areas that are not their final destinations. Currently, private logistics firms like UPS and FedEx transport a package throughout their logistics network by utilizing centralized control. In a PLN, different units of the network would be separated in terms of function. In such a network, decentralized control is adapted so that coordination from a single firm is not needed [1, 2].

Due to the decentralized nature of a PLN, a coordination mechanism is needed to facilitate communication and cooperation between different units and functions. Kay [1] proposed some protocols for trucks and packages. Distribution centers play a central role in the operation of a PLN by providing DC services to make packages, trucks, and other related information available to every participating unit in the PLN.

However, a PLN has some drawbacks, such as huge initial building costs for new DCs and other facilities. Sophisticated protocols, terms, and conditions need to be developed to make the PLN function efficiently, and to also prevent some units from gaming the system.

Figure 1.1 shows a hypothetical public logistics network with 36 public DCs covering the southeastern portion of the U.S., connected via interstate highways [1, 2]. Besides connecting each other to form a backbone structure for a PLN, each of the DCs in the figure would also function as a transportation hub for the local area surrounding the DC [2].

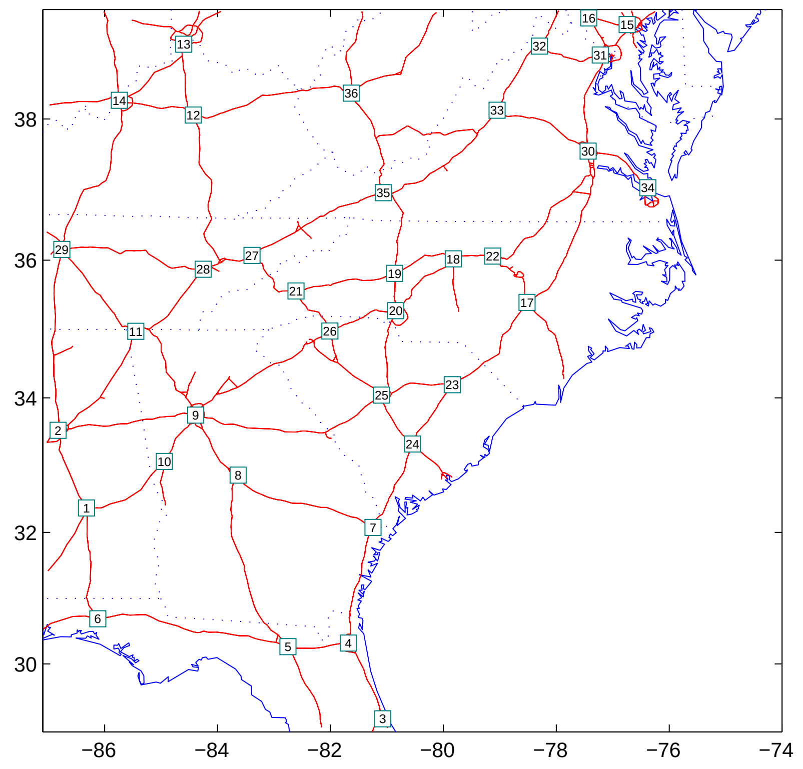


Figure 1.1. Hypothetical PLN Covering the Southeastern USA

1.2 Operations of PLN

In a PLN, each DC covers a specific area. Wherever a package enters the network, it is transported to the closest DC. The package then starts its trip (usually involving multiple DC hops), ultimately reaching its final destination DC, where it is then transported to its destination address. Figure 1.2 shows a package that enters at DC 3, travels to destination DC 5, and then is delivered to its destination address [1].

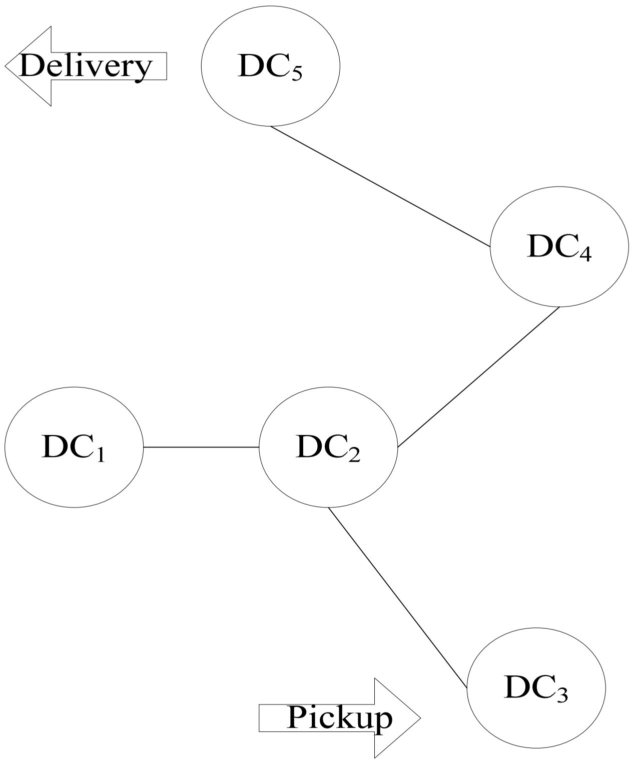


Figure 1.2. A Package Originating at DC 3 Travels to DC 5

Table 1.1 shows a comparison for functions of components in a PLN and the Internet [3]. DCs function like routers in the Internet. Packages transmitted by trucks are like data packets transmitted by cables in the Internet.

Table 1.1. Comparison between Components of a PLN and the Internet

PLN	Internet
Distribution Centers (DC)	Routers
Packages (Parcels)	Data Packets
Trucks	Wires/Cables

1.3 Literature Review

Kay and Parlikad [2] analyzed material flow features in a PLN and compared three different networks in terms of package average travel time. Bansal [5] studied PLN configurations and proposed three mechanism to design a PLN. Kay [1] and Jain [3] proposed protocols for the entities in the PLN to comply with. Those protocols make the PLN operate efficiently. In their research, the average waiting time for a package is estimated to be the product of the truck headway, which is the average interarrival time for trucks, and the headway ratio, which is the assumed to have the value 0.5. As explained in the rest of this dissertation, assuming a headway ratio of 0.5 is incorrect and yields inaccurate estimates of the performance of a PLN.

1.4 Research Objectives

The main objectives of this research are to do the following:

1. Develop a simulation model that builds on previous work of John Telford [6] to study PLN operations. A PLN can be viewed as a priority queuing network with bulk arrivals and bulk service. Due to the complexity of this problem, it is very difficult to obtain a closed-form solution for package average waiting time. Therefore, a simulation model is created in order to gain some insight and obtain some statistics on the PLN operation, such as package average waiting time and the truck load factor (that is, the space utilization for each truck trip, which is assumed to be the same for all trucks), to be used in further analysis.

2. Use analytical and simulation models to find an accurate and practical procedure to estimate package average waiting time. Kingman's equation [7] is applied to approximate the package average waiting time. The latter quantity can be represented as the product of the truck headway and the headway ratio (refer to Section 2.2 for the definition of this term). Statistics from the simulation and analytical models are then used to estimate the headway ratio.

3. Design the optimal PLN that yields minimal package average waiting times for all packages' origins and destinations. Potential locations (search space) for the DCs are found using the U.S. network of interstate and highways. A genetic algorithm (GA) is applied to the search space to determine the optimal number of DCs, their locations, and the direct arc connections between adjacent DCs.

In addition to the main research objectives, we also discuss using the PLN simulation to implement the protocols proposed in Kay [1] and Jain [3]. As an extension to the basic simulation for the PLN, those advanced features focus on protocols, package bidding procedures, truck agent behaviors, etc. A series of protocols have been developed by Kay [1] for entities in the PLN to comply with. Packages bid for their transportation along the way. Higher bidders get higher priority for truck transport services. Trucks obey a greedy rule, in that they will transport the load of packages with the highest total bid. The bidding process and the load acceptance process are controlled by agents (intelligent software).

1.5 Layout of the Dissertation

Chapter 2 introduces the analytical and simulation models. We illustrate how the truck headway and the headway ratio are used in the package average waiting time calculation. Validation of the simulation model is presented as well.

Chapter 3 focuses on obtaining the package average waiting time approximation by Kingman’s equation. Some statistics, such as the coefficient of variation for package interarrival time, and the coefficient of variation for truck interarrival time, etc., are collected from analytical and simulation models (with a simple package bidding scheme) and are used in a regression-based method to estimate the headway ratio.

Chapter 4 presents the PLN simulation with some advanced features, including the package bidding and truck acceptance of package loads. In addition, some of the simulation architecture design issues are discussed.

Using results from Chapter 3, we present in Chapter 5 a procedure to design the PLN that yields the minimal package average waiting time. A genetic algorithm is applied to search for the number of DCs and their locations based on the U.S. network of interstate and highways.

Finally, Chapter 6 wraps up with conclusions and recommendations for future work.

2 PLN Analytical Model and Simulation Model

2.1 Literature Review

A computer simulation is a computer program that attempts to simulate an abstract model of a particular system. Computer simulations have become a useful part of mathematical modeling of many natural systems to gain insight into the operation of those systems, or to observe their behavior [8, 9, 10]. Sadoun [11] provides a comprehensive review for applied system simulation.

2.2 Headway and Headway Ratio for Waiting Time Calculation

Queuing theory has been widely used in practice to study average queue length, average waiting time for a unit to receive service, etc. In the waiting time approximation, the service time plays a big role. Actually, from the simple M/M/1 queue to the fairly complicated G/G/1 queue, expected waiting time is calculated as:

$$W_q = \theta * \mu, \tag{2.1}$$

where μ is the average service time and θ is a number determined by server utilization, job arrival rate, and machine service rate. Different types of queuing systems have different representations for θ. For example, for an M/M/1 queue, $\theta = \rho/(1-\rho)$ (where ρ is server utilization); and for a G/G/1 queue, $\theta = \left(\frac{\rho}{1-\rho}\right)\left(\frac{C_A^2 + C_B^2}{2}\right)$, where ρ is the server utilization, C_A^2 is the squared coefficient of variation for the job interarrival time, and C_B^2 is the squared coefficient of variation of the service time. In Equation (2.1), the average waiting time for a unit to receive service is expressed as the product of average service time (μ) and a number (θ). In a PLN, the service time for package transportation is the truck interarrival time. This suggests that, in a PLN, the package average waiting time can be approximated as the product of truck headway (truck average interarrival time, which represents average service time in a PLN) and the truck headway ratio, which is a term similar to θ in Equation (2.1), consolidating all parameters except truck headway in the waiting time approximation equation (for detailed queuing abstraction and model specification, refer to Section 3.2.1).

A simple example of this is as follows. If three trucks visit a station in 24 hours, then the average truck headway is 8 hours. If the headway ratio is 0.5, then packages wait on average four hours for a truck. Again, for different types of queues, the headway ratio has different values.

The queuing principle that Poisson Arrivals See Time Averages (PASTA) states when the arrival process to a system is Poisson, the long-run fraction of arrivals that see the process in a particular state is equal to the long-run fraction of the time the process is in that state [12]. It is well known that, compared with the single-arrival, and single-service queuing model, bulk features (bulk arrivals or/and bulk services) introduce more uncertainty and make the average waiting time longer [15]. Without further research on package average waiting time in a PLN (with a Poisson arrival process), we initially surmised that the headway ratio would be less than one (if a random variable is exponentially distributed, its coefficient of variation is 1). But it turns out that in a PLN, the headway ratio is much larger than 1, due to the uncertainty induced by the bulk arrivals and bulk service as described above.

2.3 Analytical Model

Simulation is a good tool to study real-world problems. However, it is not always possible to get insight into the system's behavior solely by analyzing the output of a simulation of the target system. Moreover, simulation is computationally intensive. Therefore, an approximate analytical model has been developed and coded in MATLAB. This analytical model calculates the number of trucks needed to handle a specific package demand, which can be used as an input parameter in a simulation model to target a desired truck load factor.

In a PLN, the transport time for a package traveling from DC_i to DC_j is composed of three parts: the time waiting for a truck (WT_{ij}), the loading and unloading time at each DC (TimeLU, it is assumed to be the same in all DCs), and the actual travel time onboard a truck (TT_{ij}):

$$t_{ij} = \mathrm{WT}_{ij} + \mathrm{TT}_{ij} + \mathrm{TimeLU}. \tag{2.2}$$

Since the loading and unloading time and the travel time are fixed (assuming trucks travel at a constant speed and packages are routed along the shortest paths), the only part left to determine the total transport time is the time waiting for a truck.

2.3.1 Analytical Model Assumptions

1. Trucks operate around the clock and travel at constant speed.
2. The truck load factor (which is defined to be the fraction of the truck's capacity to carry packages that is used on each trip) is 80%.
3. The headway and headway ratio are used to approximate package average waiting time as the product of the truck headway and the headway ratio.

2.3.2 Analytical Model Input Parameters

TimeLU: Loading and unloading time within DC (unit: minutes);

ProxFac: Proximity factor [2] (unit: none);

Nopkg: Total number of packages entering the PLN over a 24-hour period (unit: packages);

TrCap: Truck capacity (unit: packages);

TrSpeed: Truck traveling speed (unit: miles per hour);

LdFacA: Load factor for trucks (unit: none);

HWRatio: Headway ratio (unit: none).

The Proximity factor (ProxFac) represents the degree to which a DC is more likely to transport packages to nearby DCs as opposed to DCs located further away. More detailed information about the proximity factor can be found in Kay [2]. The quantity Nopkg is assumed to be constant in the analytical model. The symbol arc$_{i,j}$ represents the direct connection from DC$_i$ to DC$_j$. Similarly, the symbol arc$_{j,i}$ represents the direct connection from DC$_j$ to DC$_i$. We assume the direct connections between DCs are symmetric, that means if arc$_{i,j}$ exists, then arc$_{j,i}$ exists as well, and vice versa. Note that if there is no direct connection between DC$_i$ to DC$_j$, then arc$_{i,j}$ and arc$_{j,i}$ do not exist.

2.3.3 Analytical Model Detailed Calculation

In the analytical model, analysis is focused on each individual arc (the direct connection between two DCs). On each arc we calculate truck headway, package average waiting time, and the number of trucks needed to meet the package demand.

In the analytical model, a number of packages (Nopkg) are transported by trucks in 24 hours. The number of packages originating in a DC is determined by the population weight for the area covered by that DC (as calculated in Equation (2.3)). The number of packages traveling from an origin DC to a destination DC is proportional to the product of the population weights for the origin and destination DCs (as explained in Section 2.3.3.1). For each specific arc, the analytical model calculates how many truck trips are needed to meet the package demand. Then the truck headway for that arc is the ratio of 24 to the number of truck trips on that arc. The package average waiting time is calculated as the product of the truck headway and the headway ratio. Since each truck travels at a constant speed, the package's travel time on each arc along the way can be computed. Then the total time that the package spends in the PLN can be calculated by Equation (2.2). Finally, the number of trucks needed is the ratio of the total truck hours (summing up the total travel time for each truck) to 24.

2.3.3.1 Number of Packages Transported between Two DCs

In the analytical model, the DC population weights are represented by a 36-element vector. (We use Figure 1.1 as the PLN network where there are 36 DCs. Their locations and the population each DC covers are known). The population weight of DC$_i$, w_i , is calculated in Equation (2.3),

$$w_i = \frac{\text{Pop}_i}{\sum_{j=1}^{36} \text{Pop}_j} \qquad i = 1, 2, ..., 36, \tag{2.3}$$

where Pop$_i$ is the population covered by DC$_i$.

The number of packages originating at each DC is proportional to its population weight,

$$\text{nopkg\_DC}_i = \text{nopkg} * w_i, \tag{2.4}$$

The number of packages originating at DC$_i$ is the product of Nopkg and w_i.

The total number of packages ($\text{nopkg\_OD}_{i,j}$) that travel from DC$_i$ to DC$_j$ is calculated as:

$$\text{nopkg\_OD}_{i,j} = \text{Nopkg} * w_{ij}. \tag{2.5}$$

For a proximity factor equal to zero (ProxFac = 0),

$$w_{ij} = w_i * w_j. \tag{2.6}$$

For none-zero proximity factor (ProxFac≠0), refer to Kay [2] for details on how to calculate w_{ij}.

Similarly, the total number of packages ($\text{nopkg\_OD}_{j,i}$) that travel from DC$_j$ to DC$_i$ is calculated as:

$$\text{nopkg\_OD}_{j,i} = \text{nopkg} * w_{ji}. \tag{2.7}$$

Since $w_{ij} = w_{ji} = w_i * w_j$, the quantity $\text{nopkg\_OD}_{i,j}$ is the same with the quantity $\text{nopkg\_OD}_{j,i}$.

2.3.3.2 Number of Packages Transported on a Single Arc

The number of packages traveling over the arc from DC$_i$ to DC$_j$ is represented by $\text{nopkg\_arc}_{i,j}$. Similarly, the number of packages traveling over the arc from DC$_j$ to DC$_i$ is represented by $\text{nopkg\_arc}_{j,i}$. Dijkstra's algorithm [13] is used to find out the shortest path from any DC to another.

The following example explains the difference between $\text{nopkg\_OD}_{i,j}$ and $\text{nopkg\_arc}_{i,j}$. In the PLN shown in Figure 2.1, the solid lines represent arcs between two DCs, and the dotted lines represent packages transported from one DC to another. The quantity $\text{nopkg\_OD}_{3,2}$ represents the number of packages traveling from DC$_3$ to DC$_2$ and the shortest path for those packages requires travelling through arc$_{3,4}$, and arc$_{4,2}$. The quantity $\text{nopkg\_OD}_{4,2}$ represents the number of packages traveling from DC$_4$ to DC$_2$, and the shortest path for those packages requires travelling over $arc_{4,2}$. The quantity $nopkg\_OD_{5,2}$ represents the number of packages traveling from DC_5 to DC_2, and the shortest path for those packages requirs travelling through $arc_{5,4}$, and $arc_{4,2}$.

The quantity $nopkg\_arc_{4,2}$ represents the number of packages going through $arc_{4,2}$. In the example given above, the quantity $nopkg\_arc_{4,2}$ is the summation of the quantities $nopkg\_OD_{3,2}$, $nopkg\_OD_{4,2}$ and $nopkg\_OD_{5,2}$ since all of those packages go through $arc_{4,2}$.

Note that, as explained in Section 2.3.2, the symbol $arc_{i,j}$ represents the direct connection from DC_i to DC_j. But if there is no direct connection between DC_i to DC_j, then $arc_{i,j}$ does not exist and the quantity $nopkg\_arc_{i,j}$ is not defined. For example in Figure 2.1, the symbol $arc_{3,2}$ does not exist since there is no direct connection between DC_3 and DC_2. Therefore, the quantity $nopkg\_arc_{3,2}$ is not defined. However, $nopkg\_OD_{3,2}$ is defined as the number of packages traveling from DC_3 to DC_2. This shows the difference between $nopkg\_OD_{i,j}$ and $nopkg\_arc_{i,j}$.

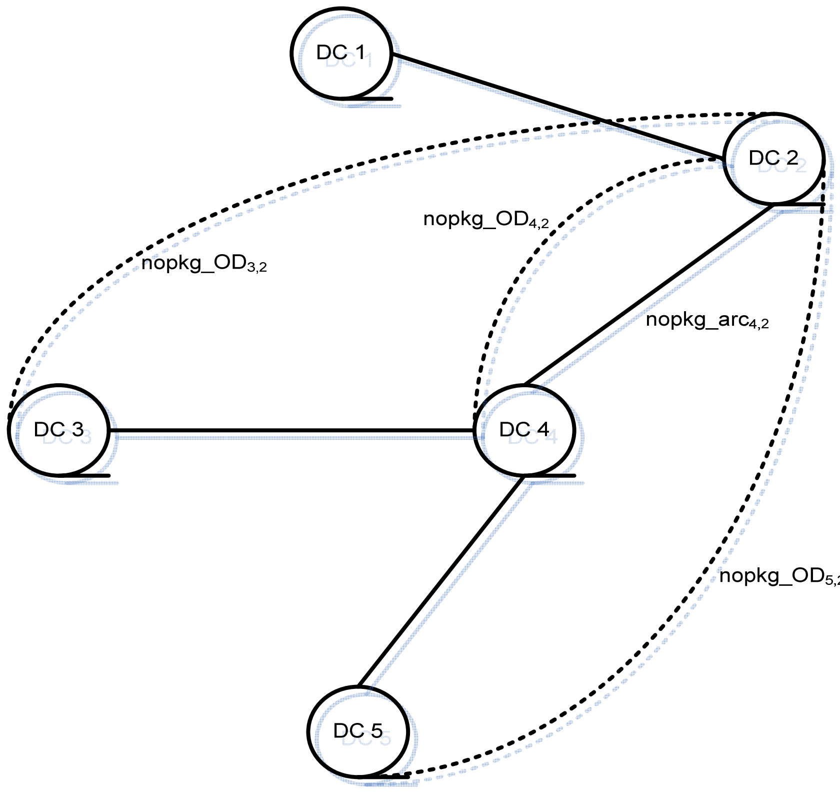


Figure 2.1. Packages Transported between DCs and Packages Transported on a Single Arc

Thus, the quantity $\text{nopkg\_arc}_{i,j}$ is the total number of packages whose shortest path goes through $\text{arc}_{i,j}$,

$$\text{nopkg\_arc}_{i,j} = \sum_{k=1}^{\text{nopkg}} I(\text{pkg } k \text{ goes through the arc from DC}_i \text{ to DC}_j), \tag{2.8}$$

where $I(\cdot)$ is an indication function,

$$I(\text{condition}) = \begin{cases} 1, & \text{if condition is true,} \\ 0, & \text{if condition is false.} \end{cases} \tag{2.9}$$

2.3.3.3 Truck Headway

Since we assume trucks operate around the clock, the truck headway is calculated as follows:

$$\text{Headway}_{i,j} = \frac{24}{\text{nopkg\_arc}_{i,j} / (\text{TrCap} * \text{LdFacA})}, \tag{2.10}$$

where $\text{TrCap} * \text{LdFacA}$ represents how many packages each truck trip transports, and $\text{nopkg\_arc}_{i,j} / (\text{TrCap} * \text{LdFacA})$ represents how many truck trips are needed on $\text{arc}_{i,j}$ to meet the demand.

2.3.3.4 Number of Trucks in the PLN

The number of trucks needed is calculated as follows:

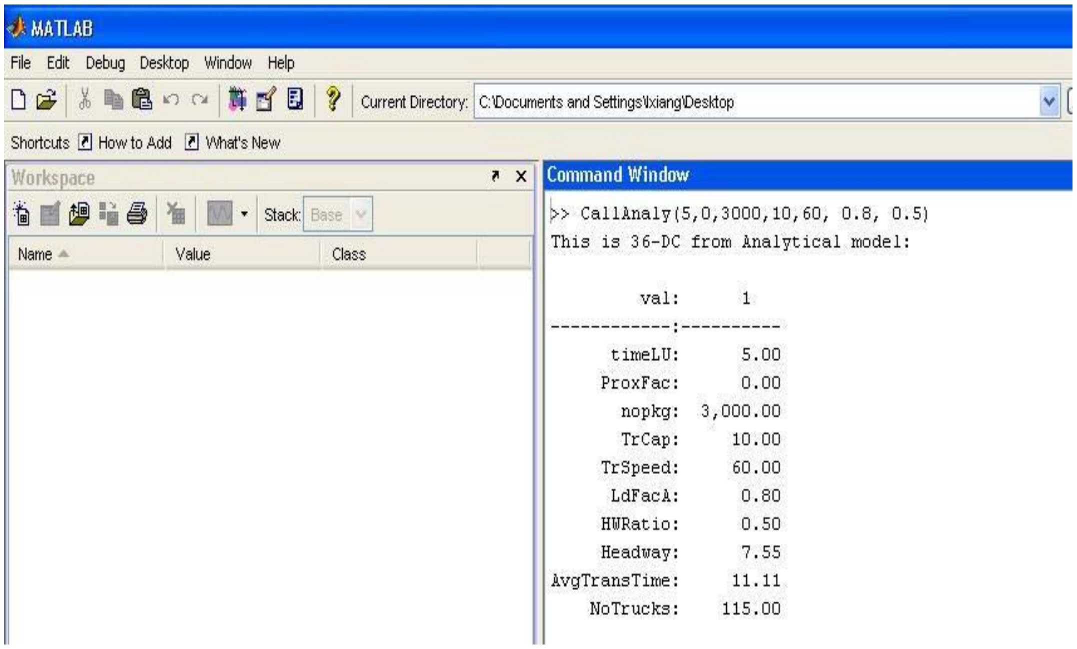
$$\text{notruck} = \left\lceil \frac{\sum_{k=1}^{\text{nopkg}} t_k / (\text{TrCap} * \text{LdFacA})}{24} \right\rceil, \tag{2.11}$$

where t_k is the total truck transport time of *k*th package. Notice that this does not include package waiting for truck time and loading and unloading time at DCs.

2.3.4 The Outputs from the Analytical Model

To give an example of output from the analytical model, we run the analytical model with following parameter values, TimeLU=5, ProxFac=0, Nopkg=3000, TrCap=10, TrSpeed=60, LdFacA=0.8. Figure 2.2 shows outputs from the analytical model. In order to reach an 80%

load factor, 115 trucks are needed in the simulation. This number of trucks serves as one input parameter into the simulation model to achieve the same load factor level.


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\Documents and Settings\xiang\Desktop
Shortcuts How to Add What's New
Workspace
Stack: Base
Name Value Class
Command Window

```
>> CallAnaly(5,0,3000,10,60, 0.8, 0.5)
This is 36-DC from Analytical model:

         val:      1
------------:----------
      timeLU:     5.00
     ProxFac:     0.00
       nopkg: 3,000.00
       TrCap:    10.00
     TrSpeed:    60.00
      LdFacA:     0.80
     HWRatio:     0.50
     Headway:     7.55
AvgTransTime:    11.11
    NoTrucks:   115.00
```

Figure 2.2. Output Snapshot from the Analytical Model

2.4 Simulation Model

The simulation model is developed using Microsoft .NET framework to take advantage of its advanced data structures and efficient program control procedures [14]. Under the .NET platform, every entity is implemented as an object [14]. This simulation model is based on Telford’s logistics simulation library [6]. In Figure 2.3, each object is enclosed in a rectangle along with its properties, operations, etc. For example, each package is an object and has properties StartDC and EndDC, as well as several associated functions (e.g., *Package*). Directed arcs denote the interactions, such as function calling, between objects.

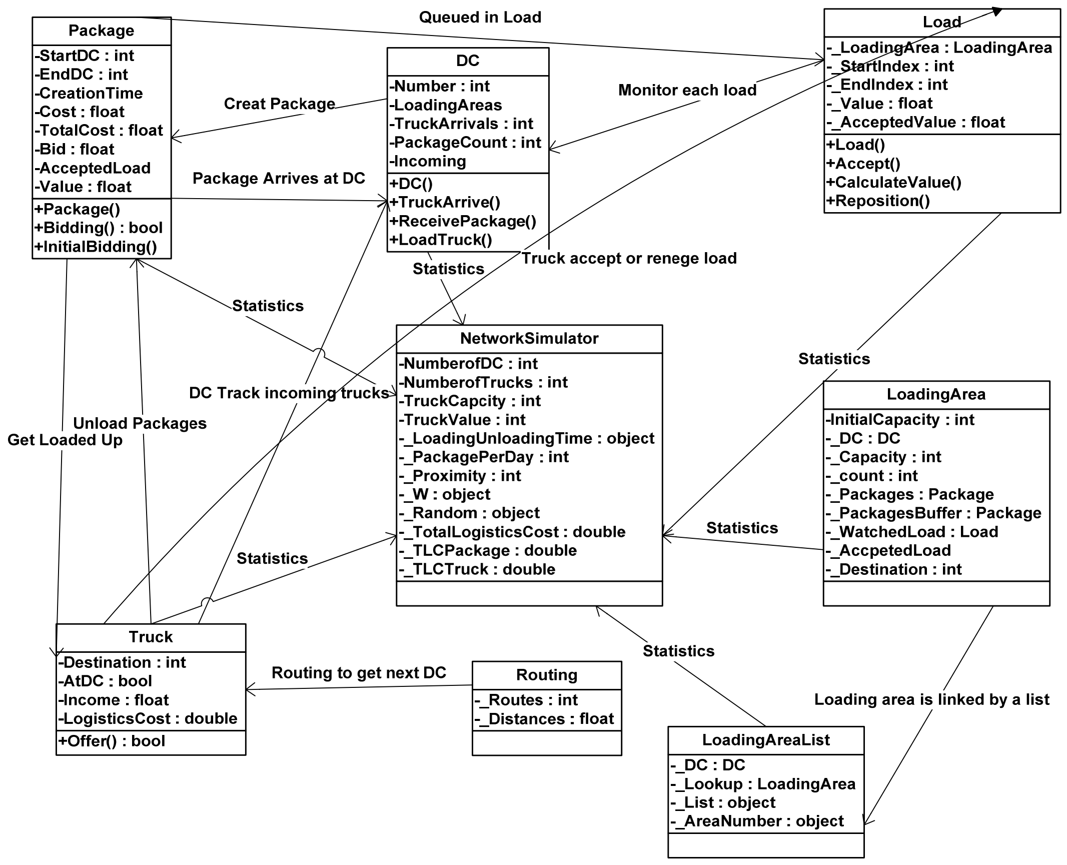


Figure 2.3. Object Design Configuration

2.4.1 Simulation Model Detailed Description

2.4.1.1 Simulation Initialization

The simulation program runs 25 replications, and 3,600 days for each replication. The initialization function initializes the simulation program. The initialization function generates the event calendar, loads the input data sets, and calls functions to create DCs, loading areas, packages, trucks, etc. It also initializes objects to keep track of statistics, such as, package waiting times, truck load factors, etc. Finally, it schedules the simulation termination events onto the event calendar to finish the simulation when the termination condition is met.

2.4.1.2 Load Input Data Sets

The analytical model generates text files to hold the following information: distance between any two DCs (*Distances*), a matrix representing the direct arc between two DCs (*ArcDistance*), and the package routing scheme (*Routing*), which is determined using Dijkstra's algorithm [13]. The LoadData function loads this information to be used in simulation.

2.4.1.3 Package Generation

The total number of packages generated in the simulation is 1.8 million per day, which is approximately the same number of packages handled daily by UPS in the southeastern U.S.A. [2]. Packages are generated according to a Poisson process. Like the analytical model, the number of packages originating in a particular DC is proportional to its population weight. When a package is generated, its origin DC and destination DC are randomly generated based on the DCs' population weights. At the same time, the count for total number of packages generated in the simulation (*PackageID*) is incremented by one.

2.4.1.4 Package Routing

Whenever a package reaches a DC, the *Routing* function is called to determine its next DC according to the shortest path determined by Dijkstra's algorithm. If the current DC is this package's destination DC, then the current package is eliminated from the simulation and some statistics, such as PackagesDelivered, PackageWaitTime, etc. are updated. If the current DC is not this package's destination, then the package is put in the package queue (loading area) to wait for a truck to transport the package to its next destination.

2.4.1.5 DC Services

DCs play a central role in the PLN operation. A DC collects packages originating from the local areas it covers, maintains package queues, provides related information for incoming trucks and packages, etc. Each DC has a loading area (package queue), for each immediately adjacent DC, i.e., each DC that shares a direct arc connection. For example, in Figure 2.4, DC_1 is adjacent to DC_2, DC_3, and DC_4. Then, within DC_1, there are three loading areas designated for packages heading to DC_2, DC_3, and DC_4. Whenever a package reaches a DC, if this is not its destination DC, then the package is placed in the loading area that corresponds to its next DC.

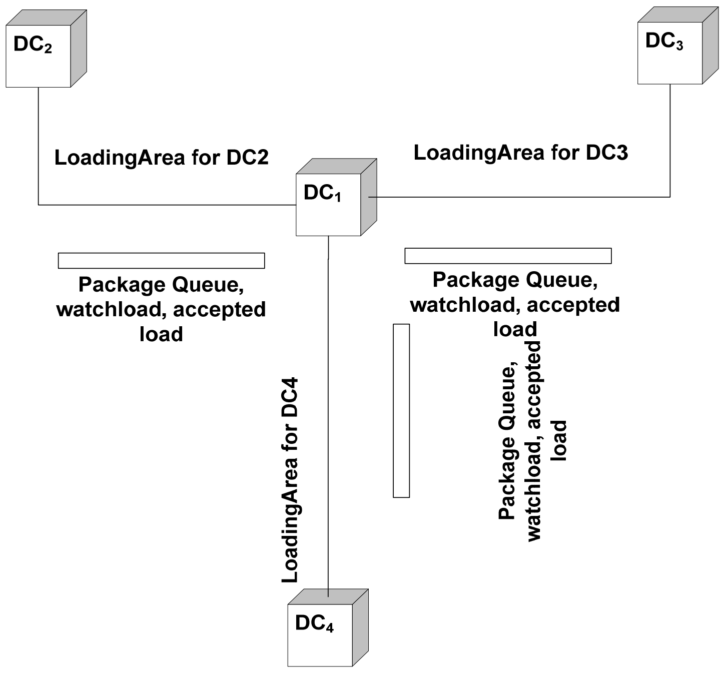

DC₂
DC₃
LoadingArea for DC2
LoadingArea for DC3
DC₁
Package Queue, watchload, accepted load
Package Queue, watchload, accepted load
LoadingArea for DC4
Package Queue, watchload, accepted load
DC₄

Figure 2.4. Example DC Configuration

When a truck arrives at a DC, the *ReceivePackage* function is called to perform the following operations: (i) empty the arriving truck; (ii) call function *Routing* to route each package on that truck to its next DC by putting the package in the corresponding loading area (if this is the final destination DC for a package, this package is eliminated from the simulation); and (iii) update all the related statistics.

In order to attract trucks to come to a DC and transport packages, only trucks currently at this DC and trucks heading to this DC from other DCs directly connected with this one are made eligible to accept a load of packages (that is, eligible for the load offering process). For example, in Figure 2.5, DC_2 has a load to offer to trucks. Truck 1 is heading to DC_2, truck 2 is heading to DC_4, which is adjacent to DC_2, and truck 3 is currently at DC_2. In this scenario, only truck 1 and truck 3 are eligible for this load offering. Since truck 3 is currently at DC_2, truck 3 has higher priority than truck 1; and thus truck 3 will be the first truck to have the chance to consider accepting this load. Note truck 2 is heading to DC_4 not DC_2, so truck 2 is not eligible for the load offering at DC_2.

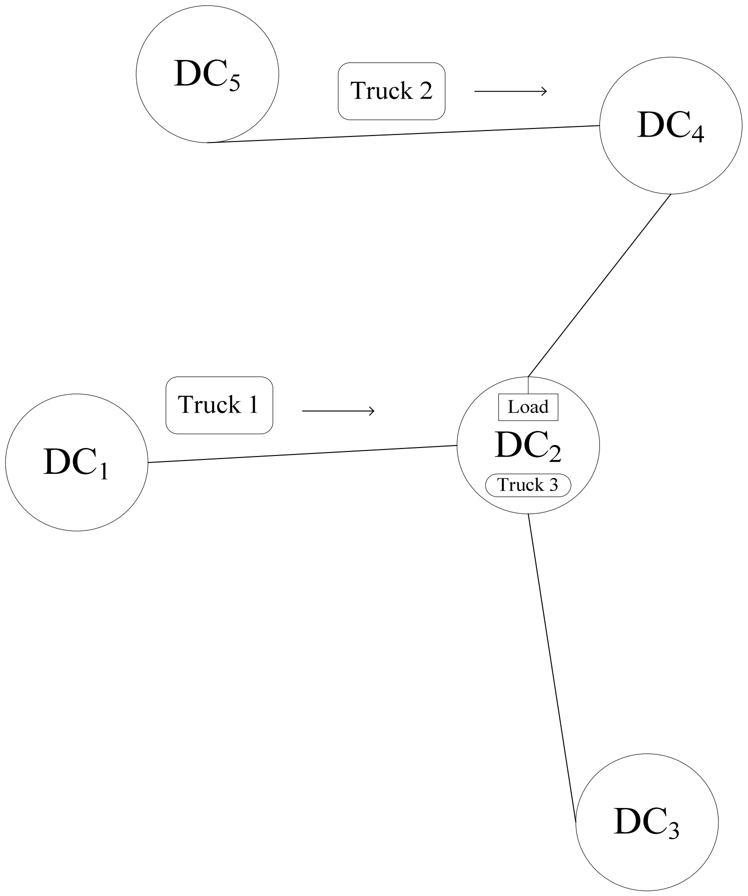


Figure 2.5. Illustration of Trucks Eligible for Load Offering

2.4.1.6 Trucks

The analytical model calculates the number of trucks that are needed to achieve the target of an 80% truck load factor on each arc on the average. In the initialization process, the initial locations of all trucks are randomly generated. That means, unlike packages, all trucks enter the simulation at one time. Once packages enter the simulation, trucks start to accept packages, load the accepted packages, and travel to their next DC.

2.4.1.7 Loading Areas

Within a given DC, each loading area is a queue to hold packages waiting for transport to a specific DC. When packages come into a DC and are routed to the next DC, a function named *Add* is called to add those packages to the corresponding loading area. The first L packages (where $0< L <=$TrCap*LdFacA) comprise a load (named *watchload*). The loading areas in a DC take turns to offer their *watchload* to trucks. If a truck accepts that *watchload*, then a function named *LoadTruck* is called to perform the following operations: (i) move this load of packages onto the truck; and (ii) arrange for the truck to leave for the next DC immediately by making a call to function *TruckAccepted*.

2.4.1.8 Statistics Objects

A number of statistics objects, such as *ArcPackageIntArrMean*, *ArcTrucksLastArr*, *PackagesInSystem*, etc., are created to keep track of what is happening in the simulation. Those statistics are output to a text file for later analysis.

2.4.1.9 Simulation Termination

When the simulation time reaches the preset limit, a function named *Terminate* is called to terminate the simulation program. The function *Terminate* destroys all objects, releases computer memory, and creates text files with designated statistical outputs.

2.4.2 Simulation Model Input Parameters

TimeLU: Loading and unloading time within a DC (unit: minutes);

ProxFac: Proximity factor (unit: note);

Nopkg: Total number of packages entering the PLN over a 24-hour period (unit: packages);

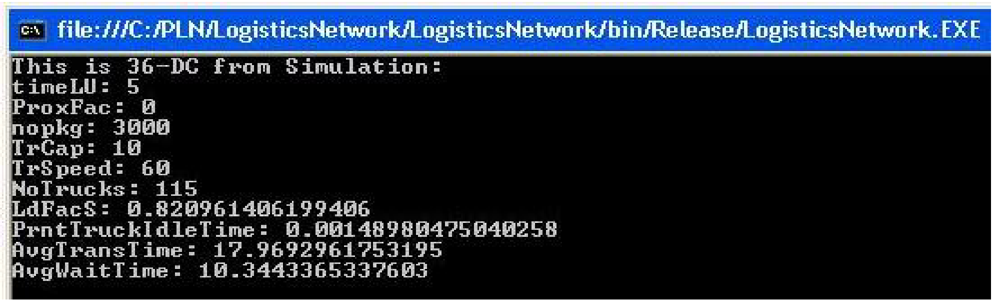
TrCap: Ttruck capacity (unit: packages per truck);

TrSpeed: Truck traveling speed (units: miles per hour);

NoTrucks: Number of trucks in the PLN (unit: trucks).

Note the parameter ProxFac is explained in Section 2.3.2.

2.4.3 Simulation Outputs



```
file:///C:/PLN/LogisticsNetwork/LogisticsNetwork/bin/Release/LogisticsNetwork.EXE
This is 36-DC from Simulation:
timeLU: 5
ProxFac: 0
nopkg: 3000
TrCap: 10
TrSpeed: 60
NoTrucks: 115
LdFacS: 0.820961406199406
PrntTruckIdleTime: 0.00148980475040258
AvgTransTime: 17.9692961753195
AvgWaitTime: 10.3443365337603
```

Figure 2.6. Output Snapshot from the Simulation Model

Figure 2.6 shows that with the current set of parameters (TimeLU = 5, ProxFac = 0, Nopkg = 3000, TrCap = 10, TrSpeed = 60, LdFacA = 0.8), the load factor is 82%. The average transit time for packages is 17.96 hours, and the package average wait time is 10.34 hours.

2.5 Simulation Model Validation

Before further research is carried out with the simulation model, it is necessary to validate that it is working correctly. Little's law [15] calculation and M/G/1 queuing calculation are used to validate the simulation model.

Little's Law: The long-run average number of customers in a stable system (over some time interval), WIP, is equal to the long-run average arrival rate of customers, λ, multiplied by the long-run average cycle time in the system, CT [15]:

$$\text{WIP} = \lambda\ \text{CT}. \tag{2.12}$$

In the simulation, WIP is the average number of total packages waiting in queues and the number of packages in transit on trucks (average number of packages in the PLN), λ is the average arrival rate of packages, and CT is the average length of time a package stays in the PLN. The simulation model keeps track of the total number of packages created and each package's cycle time. The long-run average arrival rate of customers is the ratio of the total number of packages created to the simulation length. The long-run average cycle time is the average of recorded package's cycle time in the simulation. The value of WIP (denoted by WIP_Calc) is calculated as the the product of the long-run average arrival rate of customers and the long-run average cycle time. The simulation model aslo keeps track of package queue length. The estimated WIP value (denoted by WIP_Simu) is the average number of packages in the PLN. The validation is to see how much difference is between WIP_Simu and WIP_Calc.

The simulation program runs 25 replications, and 3,600 days for each replication. The simulation wamp-up period is 90 days. Based on a 36-DC PLN (shown in Figure 1.1), eighteen different parameter sets were used in the simulation model. TimeLU (loading and unloading time of 5 minutes), and TrSpeed (truck speed of 60 mph), were kept the same across those 18 parameter sets.

Table 2.1. Validation of the Simulation Model against Little's Law

Experiment #	Nopkg/day	TrCap	ProxFac	WIP_Calc	WIP_Simu	Diff
1	15000	45	0	1774.5	1776.3	-0.10%
2	15000	50	0	1488.4	1489.9	-0.11%
3	15000	55	0	1452.4	1455.9	-0.25%
4	16000	45	0	2183.7	2348.7	-7.55%
5	16000	50	0	1580.8	1581.9	-0.07%
6	16000	55	0	1475.1	1475.7	-0.04%
7	17000	45	0	2241.1	2324.3	-3.71%
8	17000	50	0	1653.6	1654.1	-0.03%
9	17000	55	0	1583.7	1584.2	-0.03%
10	15000	45	2	1547.9	1550	-0.13%
11	15000	50	2	1367.6	1368.2	-0.04%
12	15000	55	2	1336.6	1337.2	-0.05%
13	16000	45	2	1708.3	1714.9	-0.39%
14	16000	50	2	1451.1	1451.7	-0.04%
15	16000	55	2	1389.1	1389.7	-0.04%
16	17000	45	2	1790.5	1793.7	-0.18%
17	17000	50	2	1570.4	1570.9	-0.03%
18	17000	55	2	1441.5	1442.1	-0.04%
Average						-0.71%
Std. Deviation						0.04%

As shown in Table 2.1, the average difference between the WIP value recorded in the simulation and the theoretical WIP value is –0.71% (with a standard deviation of 0.04%, and a 95% confidence interval of (–1.24% , –0.64%)).

In the PLN, if the truck capacity is constrained to be one and the packages arrival process is Poisson distribution, then the PLN network reduces to a M/G/1 queuing model. The Pollaczek-Khinchin formula [16] is used to calculate the long-run package average waiting time.

The package average waiting time ($\hat{W}_q$) is computed as,

$$\hat{W}_q = \left(\frac{\rho_{\text{eff}}}{1-\rho_{\text{eff}}}\right)\left(\frac{1+C_B^2}{2}\right)\mu_B, \qquad (2.13)$$

where ρ_{eff} is the effective server utilization, C_B^2 is the squared coeffient of variation for the truck headway (time between the (*k*-1)st truck and *k*th truck to arrive at DC$_i$ with DC$_j$ as the next destination) and μ_B is the mean value for the truck headway. Refer to Section 3.2.2.1 for detailed notation and definition. Note that since the package arrival process has a Poisson distribution, the squared coeffient of variation for the package interarrival time is one.

The simulation program runs 25 replications, and 3,600 days for each replication. The simulation wamp-up period is 90 days. Based on a 3-DC PLN, in the simulation model, packages arrive according to a Poisson process (different numbers of packages for the PLN are used for multiple experiments), the truck capacity is one, and the loading and unloading time is 5 minutes. In the simulation, we record the squared coeffient of variation for the truck headway (C_B^2), the mean value for the truck headway (μ_B), and the package average waiting time (W_q). The quantity ρ_{eff} is 80% (refer to Section 3.2.2.4 for a detailed discussion for the effective server utilization) and then $\hat{W}_q$ is estimated by Equation (2.13). The comparison between the package average waiting time in the simulation (W_q) and the estimated package average waiting time ($\hat{W}_q$) is shown in Table 2.2.

Table 2.2 Validation of the Simulation Model against the M/G/1 Queuing Model

Experiment #	Nopkg/day	WT_Simu (W_q)	WT_Calc ($\hat{W}_q$)	Difference
1	150	9.1	8.2	10.3%
2	160	9.1	8.3	8.5%
3	170	9.6	9.1	5.4%
4	180	9.7	9.5	1.8%
5	190	9.9	10.1	–1.6%
6	200	10.5	10.9	–3.8%
Average				3.4%
Std. Deviation				5.6%

On average, the difference between the package average waiting time recorded in simulation (WT_Simu) and those calculated by M/G/1 queuing model (WT_Calc) is 3.4% (with a standard deviation of 5.6%).

3 Package Average Waiting Time Approximation

3.1 Literature Review and Research Issues

Kay and Parlikad [2] analyzed material flow features in a PLN. Bansal [5] studied PLN configuration and proposed some mechanisms to design a PLN. Kay [1] and Jain [3,3] proposed some protocols (for package bidding, truck agent behavior, etc) for the PLN, and they tried to make the PLN work more efficiently. In their research, the average waiting time for a package is based on truck headway, which is the interarrival time for a truck at a DC. Average waiting time is assumed to be half of the headway (headway ratio is 0.5).

In the PLN network design problem, the package average waiting time is critical since it is a large component of the measurements to evaluate each proposed network configuration. Therefore, a better approximation for package average waiting time is needed to revise and refine a model of the performance of a PLN network.

A simulation model [8] has been built to observe the values of performance metrics that are needed in the package average waiting time approximation. Those values cannot be obtained by exact queuing-theoretic analysis due to the complexity of the PLN (a priority queuing network with bulk arrivals and bulk service).

Bulk service queuing problems in transportation have attracted many researchers and scholars since the mid-twentieth century. The original paper in this field was published by Bailey [17], who studied the problem that each vehicle departure is independent of the number of waiting customers. Since then, a large number of contributions have been made to the field, including Downton [18] Miller [19], Neuts [20], Cohen [21], Chaudhry [22] and Powell [23, 24, 25]. Downton formally analyzed the average waiting time problem with bulk arrivals and bulk service, which laid a solid groundwork for future research on this problem. Neuts researched bulk queues with different vehicle-control strategies, but he only considered a single Poisson arrival stream. Powell studied this research issue from the perspective of load planning. He also approximated the average waiting time with iterative numerical computing. However, his work is heavily dependent on his specific model and vehicle-control policies. Tegiilm [26] and Borthakur [27] considered the general bulk service rule for bulk arrivals (with variable sizes for arrivals).

Whitt [28, 29] studied limiting theorems in the condition of heavy traffic. He also studied an interpolation approximation for the mean workload in a GI/G/ 1 queue. However, this does not apply to the bulk-arrival and bulk-service scenario. Marchal [30] developed some simple waiting-time approximations for the GI/G/1 and GI/G/c queues. Kingman [7] developed an approximation for the average waiting time for the GI/G/1 queue. Kingman’s approximation has been used by Hopp [15], who showed that when server utilization is high, the approximation is fairly accurate.

3.2 Package Average Waiting Time Approximation in a PLN

3.2.1 The PLN Queuing Model Abstraction

Figure 3.1 shows the abstraction of the PLN to a standard queuing model. In a PLN, any arc (direct connection) between two DCs contains a package queue. The arc between DC_1 and DC_2 contains two package queues: one queue with packages going from DC_1 to DC_2, and another queue with packages going from DC_2 to $DC_{1.}$ A truck transports a load of packages on each trip between two DCs, and when the truck reaches a DC, its load of packages are disassembled, and the associated packages are moved to their assigned package queues. Clearly, the PLN operation involves bulk arrivals and bulk service. A PLN is an open interconnected network of such package queues. So a PLN can be abstracted as an open queuing network with bulk arrivals and bulk service.

The package waiting time for truck transport is composed of two parts. In part 1 of the waiting process, packages form a load in the assembly queue. In part 2 of the waiting process, loads waiting in the transport queue for shipment to their next destination DC. When packages arrive at a DC, they are put in the appropriate assembly queue. Whenever a number of packages (L packages, where L is the load size) form a load, this load is immediately moved to the transport queue to wait for truck transport. On each trip, a truck only transports one load to the destination DC. If customers in the system are loads, then the system can be modeled as a network of interconnected G/G/1 queues. The customers' arrival process is defined as the arrival of loads at a DC for transport to another DC, and the interarrival time is the time between the formation of consecutive loads. The service is defined as truck transport, and the service time is the interarrival time between consecutive trucks destined for the same DC. Detailed definition can be found in Section 3.2.2.1.

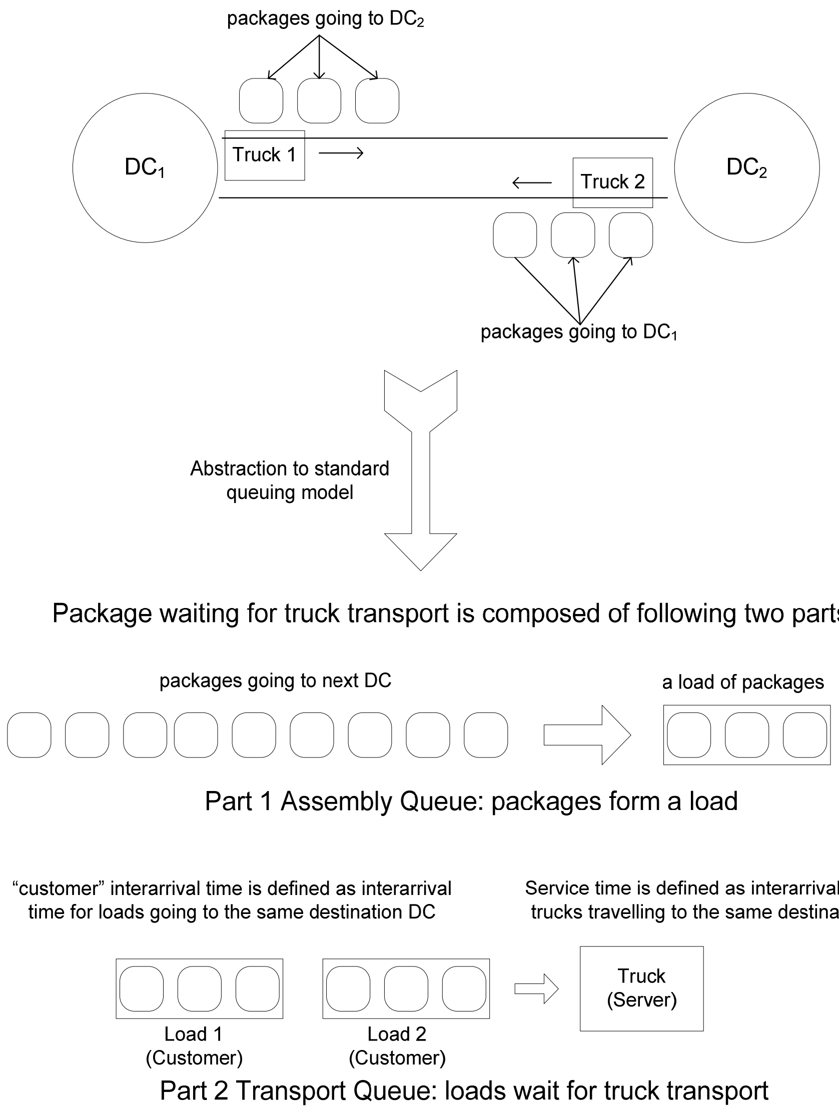

packages going to DC2
DC1
Truck 1
Truck 2
DC2
packages going to DC1
Abstraction to standard queuing model
Package waiting for truck transport is composed of following two parts
packages going to next DC
a load of packages
Part 1 Assembly Queue: packages form a load
"customer" interarrival time is defined as interarrival time for loads going to the same destination DC
Service time is defined as interarrival time for trucks travelling to the same destination DC
Load 1 (Customer)
Load 2 (Customer)
Truck (Server)
Part 2 Transport Queue: loads wait for truck transport

Figure 3.1. The PLN Abstraction to a Queuing Model

3.2.2 Mathematical Formulation of Target Problem

3.2.2.1 Definitions

The following definitions and terms apply to the package waiting time approximation.

Service: truck trip from one DC to another

Headway: interarrival time between consecutive trucks arriving at DC$_i$ that are destined for DC$_j$

Average load size: L packages

U_k : Time between arrival of package k–1 and package k at the assembly queue of packages at DC$_i$ waiting for transport to DC$_j$.

μ_u : Mean value of U_k

σ_u : Standard deviation of U_k

C_u : CV (coefficient of variation) of U_k , $C_u = \frac{\sigma_u}{\mu_u}$

A_k: Time between the formation of load k–1 and load k in the queue of loads at DC$_i$ waiting for transport to DC$_j$.

μ_A : Mean value of A_k

σ_A : Standard deviation of A_k

C_A : CV of A_k, $C_A = \frac{\sigma_A}{\mu_A}$

B_k: Headway, time between the arrival of the (*k* – 1)st truck and *k*th trucks at one DC with the same next destination.

μ_B : Mean value of B_k

σ_B : Standard deviation of B_k

C_B : CV of B_k, $C_B = \dfrac{\sigma_B}{\mu_B}$

r_A : Arrival rate of loads at one DC waiting for transport to the same next DC: $r_A = 1/\mu_A$

r_B : Rate at which trucks arrive to pick up a load from the queue of loads waiting at DC$_i$ for transport to DC$_j$: $r_B = 1/\mu_B$

ρ : Truck (server) utilization: $\rho = \dfrac{r_A}{r_B} = \dfrac{\mu_B}{\mu_A}$

Since a load is composed of L packages ($A_k = \sum_{j=(k-1)L+1}^{kL} U_j$), we have $\mu_A = L\mu_u$, and $\sigma_A^2 = L\sigma_u^2$. Substituting in these values, we can see that $C_A^2 = \dfrac{\sigma_A^2}{\mu_A^2} = \dfrac{L\sigma_u^2}{(L\mu_u)^2} = \dfrac{\sigma_u^2}{L\mu_u^2} = \dfrac{C_u^2}{L}$.

3.2.2.2 Average Waiting Time for a Load

Consider a load at DC$_i$ waiting to be transported to DC$_j$. Based on the average waiting time for a G/G/1 queue, the long-run average waiting time for the load is as follows [7]:

$$\hat{W}_q = \left(\frac{\rho}{1-\rho}\right)\left(\frac{C_A^2 + C_B^2}{2}\right)\mu_B \tag{3.1}$$

where ρ is truck utilization, C_A^2 is the squared coefficient of variation for load interarrival time, C_B^2 is the squared coefficient of variation for trucks interarrival time (headway), and μ_B is the headway mean value.

Within a PLN, since the package arrival process is not a Poisson process due to the batch arrival of packages on trucks to a DC and the service time (truck headway) is not exponentially distributed, values for ρ, C_A^2 and C_B^2 can only be observed from the simulation. The headway mean value μ_B can be obtained by the analytical model (described in Section 2.3), given the demand rates at each DC and the truck capacity.

3.2.2.3 Average Waiting Time for a Package

In a PLN, when a package arrives, it is put in the assembly queue to wait for a load to be formed. After a load is formed, it is put into a transport queue, where it waits for truck transportation.

Suppose a package is currently in DC$_i$, and it is going to DC$_j$. The average waiting time for this package consists of two parts:

(i) The time spent waiting in an "assembly queue" until a "complete" truckload of L packages has been assembled, at which point the truckload immediately exits the assembly queue and joins the "transport queue," which is a queue of truckloads waiting for a truck that will transport the first waiting truckload from DC$_i$ to DC$_j$;

(ii) The time spent in the "transport queue" until a truck destined for DC$_j$ finally arrives at DC$_i$ so that the first waiting truckload can exit the transport queue and start the trip to DC$_j$.

The "service time" in the transport queue is taken to be the headway (delay) between the arrival of successive trucks that are destined to deliver a truckload to DC$_j$, and where it is assumed that the transport queue is rarely empty.

Let X denote the waiting time of the target package in the assembly queue. If the target package is the kth package to be added to the truckload currently being assembled for $k = 1, \ldots, L-1$, then for $l = k+1, \ldots, L$, let U_l denote the interarrival time between the lth package and the $(l-1)$st package in the same truckload so that the total waiting time in the assembly queue for the target package is

$$X_k = \begin{cases} 0, & \text{if } k = L, \\ \sum_{l=k+1}^{L} U_l, & \text{if } k = 1, \ldots, L-1. \end{cases} \tag{3.2}$$

Thus, we have

$$E[X_k] = \left\{ \begin{array}{ll} 0, & \text{if } k = L \\ \sum_{l=k+1}^{L} E[U_l], & \text{if } k = 1,\ldots,L-1 \end{array} \right\} = (L-k)\mu_U \quad \text{for } k = 1,\ldots,L. \tag{3.3}$$

The operation of assembling each truckload is defined as a renewal-reward process in which a renewal epoch occurs each time the assembly of another truckload is finally completed, therefore a standard renewal-reward argument can be used to prove the intuitively obvious result that the long-run average time spent by the target package in the assembly queue is

$$W_{\text{assem}} = \frac{1}{L}\sum_{k=1}^{L} E[X_k]. \tag{3.4}$$

From Equations (3.3) and (3.4), the long-run average waiting time for a package in the assembly queue is

$$W_{\text{assem}} = \frac{1}{L}\sum_{k=1}^{L}(L-k)\mu_U \tag{3.5}$$

$$= \frac{\mu_U}{L}\sum_{v=1}^{L-1} v \tag{3.6}$$

$$= \frac{\mu_U}{L}\left[\frac{(L-1)L}{2}\right] \tag{3.7}$$

$$= \frac{\mu_U(L-1)}{2}. \tag{3.8}$$

Since we have

$$\mu_U = \frac{\mu_A}{L} \tag{3.9}$$

Substituting this into Equation (3.8) yields

$$W_{\text{assem}} = \frac{L-1}{2L}\mu_A \tag{3.10}$$

The average waiting time for a package in a queue in PLN is

$$W_q = W_{\text{assem}} + W_{\text{trans}} = \left(\frac{L-1}{2L}\right)\mu_A + \left(\frac{\rho}{1-\rho}\right)\left(\frac{C_A^2 + C_B^2}{2}\right)\mu_B \tag{3.11}$$

For simplicity, here we are neglecting the time it takes to place a load onto the truck and also the time it takes to remove a load from the truck.

3.2.2.4 Explanation for Effective Server Utilization

We used an effective server utilization in Kingman's approximation to approximate the package average waiting time. The reasons are as follows.

1. A PLN represents an open queuing network with batch arrivals and batch service. At each DC, the service time is the interarrival time between two trucks destined for the given DC. Therefore, the service time at one DC is not independent of that at another DC since trucks are running from one DC to another. In addition, the number of servers (trucks) at one DC is a random variable (depending on how many trucks are available at that DC). These two features distinguish a PLN queuing network from

other well-studied normal queuing networks [31, 32, 33, 34]. For the PLN queuing network, the server utilization is hard to define in this case. For example, in the PLN (Figure 3.2), trucks are running around each DC to transport packages. For each individual arc, no truck is designated to serve that arc exclusively. Each time a truck loads up and hits the road, that truck virtually belongs to the destination DC. This situation is not like a manufacturing shop, where a fixed number of workstations always serve a specific production line.

2. Server utilization is the percentage of the server's available capacity that is actually used in providing service to the "customers" (loads). On the other hand, the truck load factor represents the percentage of room that is available onboard. Since in a PLN, service is defined as package being transported from the origin DC to the destination DC, the truck load factor represents the same idea with server utilization.

Our use of Kingman's equation is based on the assumption that the "server" is effectively being used to provide "service" almost all the time so that the theoretical server utilization is

$$\rho_{\text{theor}} \approx 1.0; \tag{3.12}$$

However, if each "customer" only gets 80% of the "server's" attention while the "customer" has engaged (seized) the "server," then the effective server utilization is

$$\rho_{\text{eff}} = 0.8 * \rho_{\text{theor}} \approx 0.8\ . \tag{3.13}$$

In this situation, the correct way to estimate W_q on the arc DC$_i$ to DC$_j$ is the following:

$$\widehat{W_q} = \left(\frac{L-1}{2L}\right)\widehat{\mu_A} + \left(\frac{\rho_{\text{eff}}}{1-\rho_{\text{eff}}}\right)\left(\frac{\widehat{C_A^2}+\widehat{C_B^2}}{2}\right)\widehat{\mu_B}. \tag{3.14}$$

The expression

$$\left(\frac{\rho_{\text{eff}}}{1-\rho_{\text{eff}}}\right)\left(\frac{C_A^2+C_B^2}{2}\right)\mu_B \tag{3.15}$$

is an extension of Kingman’s equation that properly accounts for the concept of effective server utilization.

3.2.3 Verification of Package Average Waiting Time Approximation

3.2.3.1 The Simulation Model Experiments with Different Parameter Sets

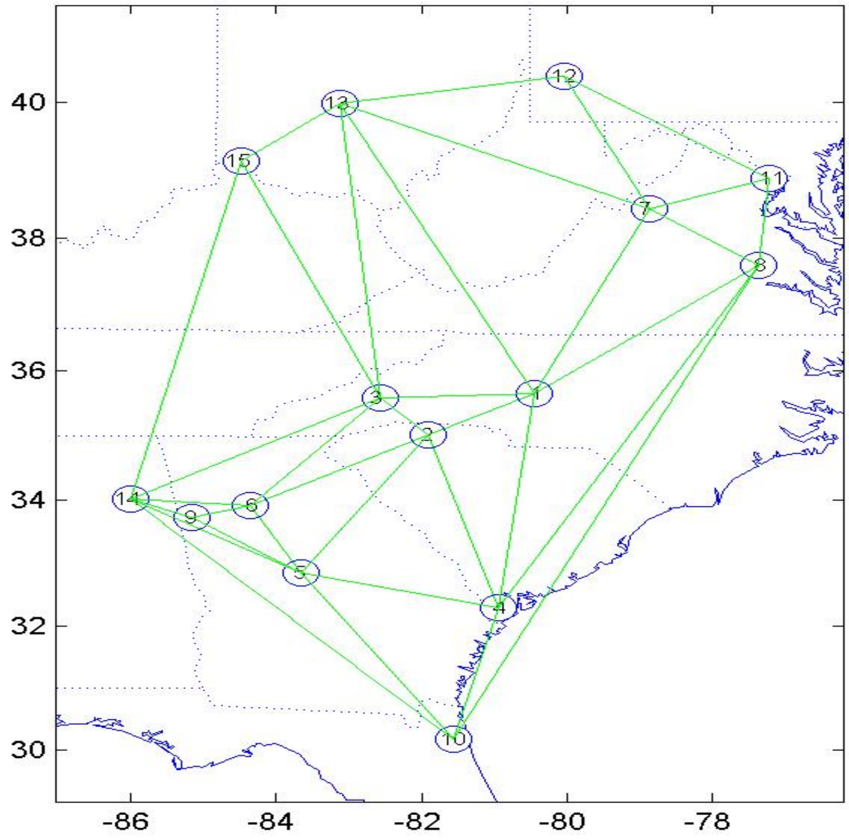


Figure 3.2. 15-DC PLN Configuration Plot

Based on a 15-DC PLN shown in Figure 3.2, the simulation runs 25 replications, and 3,600 days for each replication. Eighteen different parameter sets are used in the simulation model. The total number of packages varies over three levels (250 packages/day, 450 packages/day, 650 packages/day), the loading and unloading time varies over three levels (0 minutes, 5 minutes, 10 minutes), and the truck capacity varies over two levels (5 packages max, 10 package max). The ProxFac (refer to Section 2.3.2 for detail) is 2, and the truck speed is 60 mph across all parameter sets. For detailed information about how the simulation model works, refer to Section 2.4.1. The detailed experimental design with the parameter values and results (on the whole network level not on the individual arc level) are shown in Table 3.1.

Table 3.1. Comparison between Approximated and Recorded Values in Simulation on the Network Level

Experiment No.	TimeLU	Nopkg/day	TrCap	PkgWT	Approx(3.14)	Diff
1	0	250	5	1843.9	1782.9	3.31%
2	5	250	5	2204.9	2326.9	–5.53%
3	10	250	5	1810.9	1711.5	5.49%
4	0	450	5	3397.5	3160.0	6.99%
5	5	450	5	2138.8	2289.6	–7.05%
6	10	450	5	2228.2	2361.4	–5.98%
7	0	650	5	1115.5	1222.4	–9.58%
8	5	650	5	495.2	526.6	–6.33%
9	10	650	5	136.4	130.8	4.14%
10	0	250	10	2572.2	2787.6	–8.38%
11	5	250	10	1568.8	1523.0	2.92%
12	10	250	10	1358.1	1289.2	5.07%
13	0	450	10	128.1	139.3	–8.74%
14	5	450	10	6.6	6.1	7.35%
15	10	450	10	4.3	4.7	–9.69%
16	0	650	10	2115.3	2217.4	–4.83%
17	5	650	10	900.0	836.4	7.06%
18	10	650	10	956.0	978.3	–2.34%
Average						–1.5%
Std. Deviation						6.5%

In Table 3.1, the PkgWT column represents the package average waiting time (not the average value over an arc, but the average value over all packages in the simulation) recorded in the simulation. The column named Approx(3.14) is the value calculated by Kingman's equation with the effective server utilization (Equation (3.14)). For package average waiting time, the average difference between the approximated values and recorded values in the simulation is –1.5% with a standard deviation 6.5%.

3.2.3.2 Results Comparison on the Arc Level for Experiment One

Table 3.2 shows a comparison between the package average waiting time recorded in the simulation and the approximated values on each arc for experiment 1. The approximated values on each arc are calculated by Equation (3.14). In Table 3.2, the arc weight column represents the arc weight, which is the ratio of the number of packages going through that arc to the total number of packages in the PLN,

$$\text{W\_arc}_{i,j} = \frac{\text{nopkg\_arc}_{i,j}}{\text{nopkg}}. \tag{3.16}$$

where $\text{W\_arc}_{i,j}$ is the weight for the $\text{arc}_{i,j}$, and $\text{nopkg\_arc}_{i,j}$ is the number of packages going through $arc_{i,j}$, as calculated in Equation (2.8). The column ArcDelay is the package average waiting time on that arc (recorded in the simulation). The column Approx is the approximated value, and the column WeightedDifference is the weighted difference between those two values.

Table 3.2. Comparison between True Waiting Time (Hrs.) and Approximated Value on the Arc Level

Arc	Arc Weight	Arc Delay	Approx	Weighted Difference	Arc	Arc Weight	Arc Delay	Approx	Weighted Difference
2,1	1.68%	6072.9	17.1	–1.7%	12,7	3.93%	6306.6	74.6	–3.9%
3,1	1.03%	1675.0	27.9	–1.0%	13,7	1.46%	1686.0	12.6	–1.5%
4,1	0.18%	37.9	13.7	–0.1%	1,8	0.57%	18.8	10.5	–0.3%
7,1	1.08%	3558.6	10.6	–1.1%	4,8	0.10%	9.4	23.7	0.2%
8,1	0.57%	2039.9	56.1	–0.6%	7,8	1.39%	1688.8	32.4	–1.4%
13,1	0.39%	8.3	8.6	0.0%	10,8	0.19%	155.1	1146.9	1.2%
1,2	1.68%	5044.6	11.6	–1.7%	11,8	1.31%	1574.1	849.8	–0.6%
3,2	2.21%	2909.6	31.4	–2.2%	5,9	0.30%	61.1	2168.8	10.5%
4,2	0.29%	18.9	12.5	–0.1%	6,9	1.52%	2457.7	33.8	–1.5%
5,2	0.54%	62.3	13.0	–0.4%	14,9	0.85%	98.0	22.8	–0.6%
6,2	2.69%	6056.4	35.5	–2.7%	4,10	0.26%	11.3	14.7	0.1%
1,3	1.03%	1485.8	14.6	–1.0%	5,10	0.17%	299.8	53.4	–0.1%
2,3	2.21%	3855.3	22.4	–2.2%	8,10	0.19%	586.6	45.5	–0.2%
6,3	1.90%	5142.3	212.8	–1.8%	14,10	0.42%	14.3	23.7	0.3%
13,3	0.69%	1484.5	23.2	–0.7%	7,11	1.85%	691.1	21.1	–1.8%
14,3	0.69%	1491.0	31.2	–0.7%	8,11	1.31%	346.1	29.8	–1.2%
15,3	1.80%	5547.5	358.7	–1.7%	12,11	3.20%	879.6	49.7	–3.0%
1,4	0.18%	442.8	16.4	–0.2%	7,12	3.93%	2787.1	11.2	–3.9%
2,4	0.29%	138.8	493.7	0.8%	11,12	3.20%	2525.4	70.0	–3.1%
5,4	0.12%	19.2	25.5	0.0%	13,12	5.17%	2698.3	13.0	–5.1%
8,4	0.10%	14.5	34.8	0.1%	1,13	0.39%	155.8	13.0	–0.4%
10,4	0.26%	281.4	17.1	–0.2%	3,13	0.69%	1642.6	13.4	–0.7%
4,5	0.12%	14.8	16.5	0.0%	7,13	1.46%	3244.0	23.3	–1.5%
6,5	0.94%	1830.2	77.0	–0.9%	12,13	5.17%	4023.0	274.3	–4.8%
9,5	0.30%	2968.2	3376.2	0.0%	15,13	6.74%	6419.0	10.8	–6.7%
10,5	0.17%	536.2	77.2	–0.1%	3,14	0.69%	6.7	8.0	0.1%
14,5	0.45%	16.0	20.2	0.1%	5,14	0.45%	29.4	28.7	0.0%
2,6	2.69%	5435.5	19.6	–2.7%	6,14	1.97%	536.2	26.4	–1.9%
3,6	1.90%	1525.3	39.6	–1.9%	9,14	0.85%	3539.8	670.5	–0.7%
5,6	0.94%	34.6	27.3	–0.2%	10,14	0.42%	59.2	47.7	–0.1%
9,6	1.52%	1308.4	19.0	–1.5%	15,14	2.15%	1502.3	32.8	–2.1%
14,6	1.97%	459.2	30.6	–1.8%	3,15	1.80%	1682.4	19.6	–1.8%

Table 3.2. Continued

Arc	Arc Weight	Arc Delay	Approx	Weighted Difference	Arc	Arc Weight	Arc Delay	Approx	Weighted Difference
1,7	1.08%	4381.8	8.0	–1.1%	13,15	6.74%	4416.4	13.5	–6.7%
8,7	1.39%	684.3	15.9	–1.4%	14,15	2.15%	1061.9	17.6	–2.1%
11,7	1.85%	1652.1	14.5	–1.8%					
Average Weighted Difference									–1.2%

From Table 3.2, the average weighted difference between the package average waiting time recorded in the simulation and the approximated value is –1.2% at the network level.

Since we use the package average waiting time as a criterion to define the optimal PLN, we only need a network level approximation, not an arc level approximation. The small difference on the network level also testifies to the effective server utilization.

3.2.4 Headway Ratio

For our current research issues, we considered a set of network instances composed of 15, 20, 25, 30, 35, and 40 DCs, respectively, covering the southeast region of U.S as our test set. These six scenarios compose our universe of target PLN networks.

In the G/G/1 approximation Equation (3.14), μ_B is the headway for trucks, and

$\left(\frac{\rho_{\text{eff}}}{1-\rho_{\text{eff}}}\right)\left(\frac{C_A^2+C_B^2}{2}\right)$ is called the headway ratio.

Table 3.3. C_A^2 and C_A^2 Values across Different PLNs

Number of DCs	C_A^2	C_B^2
15	1.02	2.47
20	0.89	2.19
25	1.19	2.64
30	1.07	2.14
35	0.88	2.37
40	1.02	2.93
Average	1.01	2.46

From the simulation across different PLNs (15DC to 40DC), we found out that, as shown in Table 3.3, C_A^2 is around 0.9~1.0, and C_B^2 is around 2.4~2.5. Then, $\frac{C_A^2 + C_B^2}{2}$ turns out to be around 1.5~1.6. The effective utilization ρ_{eff} is around 80%, so the headway ratio turns out to be roughly around 6.

3.2.4.1 Headway Ratio Regression Analysis

As stated above, six scenarios (15DC, 20DC, 25DC, 30DC, 35DC, 40DC) compose our universe of target PLN networks. For each of these scenarios, three PLN parameters generate 18 different parameter sets (shown in Table 3.4). Each one of six DC scenarios runs with these 18 parameter sets. We collect the package average waiting time (ArcDelay) in the simulation model and the truck headway (ArcHeadway), and we have 6 * 18=108 pairs of ArcDelay and ArcHeadway values (shown in Table 3.5).

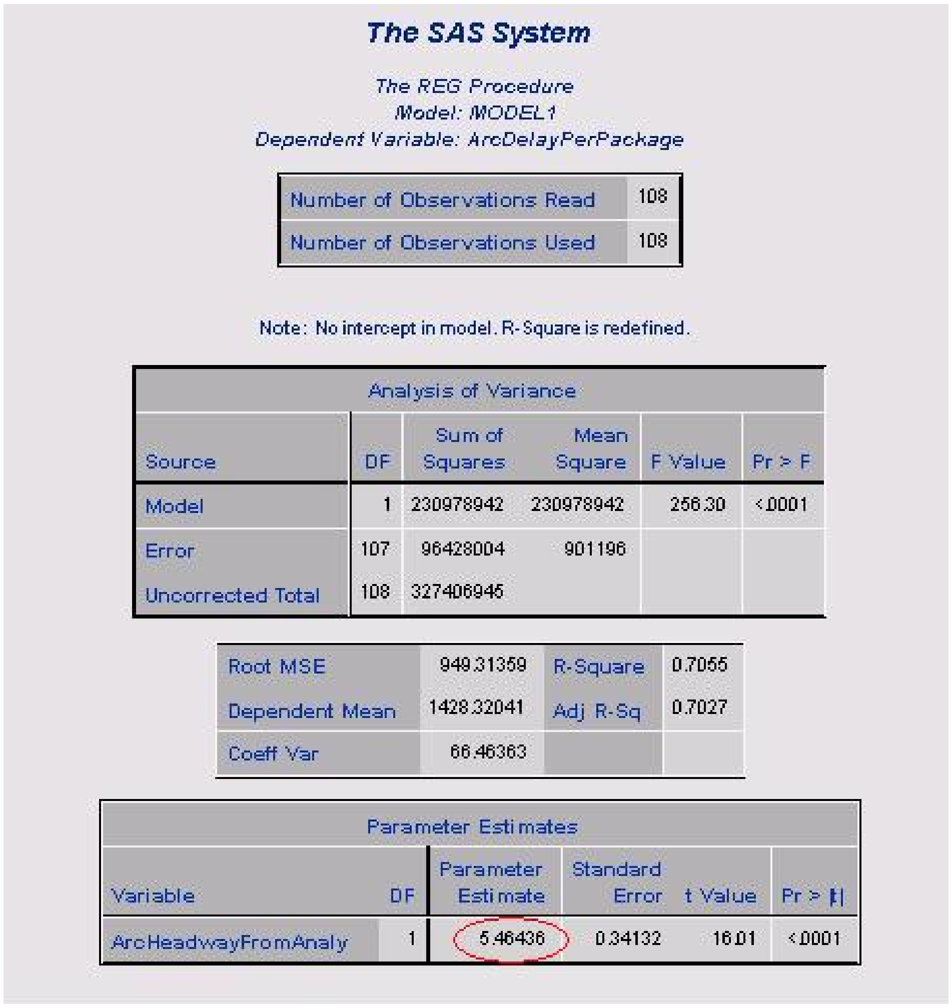
Table 3.4 Headway Ratio Regression Experiment Design

Experiment No.	Nopkg/day	TrCap	TimeLU
1	250	5	0
2	450	5	0
3	650	5	0
4	250	10	0
5	450	10	0
6	650	10	0
7	250	5	5
8	450	5	5
9	650	5	5
10	250	10	5
11	450	10	5
12	650	10	5
13	250	5	10
14	450	5	10
15	650	5	10
16	250	10	10
17	450	10	10
18	650	10	10

Regression of ArcDelay (the dependent variable) on ArcHeadway (the explanatory variable) gives us the headway ratio estimate. The regression shows the headway ratio is 5.5 (circled in Figure 3.3) and is statistically significant. The headway ratio regression plot is shown in Figure 3.4.

Table 3.5. Package Waiting Time (Hrs.) and Analytical Headway form Simulation

ArcDelay	ArcHeadway	ArcDelay	ArcHeadway	ArcDelay	ArcHeadway
2388.66	322.47	1843.89	161.21	1067.32	181.27
1195.96	179.15	2204.95	89.56	1347.31	100.71
2253.10	124.03	1810.87	62.00	998.29	69.72
2532.66	644.93	3397.48	322.42	3848.07	362.54
2501.79	358.30	2138.80	179.12	1691.76	201.41
2541.15	248.05	2228.19	124.01	2684.26	139.44
2082.00	322.47	1115.51	161.21	465.42	181.27
895.12	179.15	495.19	89.56	617.65	100.71
751.41	124.03	136.41	62.00	120.26	69.72
2353.02	644.93	2572.18	322.42	1911.47	362.54
2664.23	358.30	1568.82	179.12	1171.66	201.41
1366.69	248.05	1358.06	124.01	2251.91	139.44
717.70	322.47	128.08	161.21	216.26	181.27
266.62	179.15	6.58	89.56	8.43	100.71
151.37	124.03	4.28	62.00	10.44	69.72
1640.57	644.93	2115.31	322.42	1765.79	362.54
2478.16	358.30	899.98	179.12	459.27	201.41
1269.92	248.05	955.95	124.01	1233.78	139.44
1198.65	257.51	1977.23	145.94	1253.53	321.89
1490.79	143.06	1148.81	81.08	1071.66	178.83
2282.21	99.04	834.50	56.13	1179.21	123.80
2365.64	515.02	3327.41	291.88	3469.43	643.78
2184.49	286.12	2258.54	162.15	1765.28	357.65
3678.90	198.09	1610.11	112.26	3849.10	247.61
2423.79	257.51	1214.41	145.94	632.46	321.89
112.83	143.06	274.30	81.08	795.59	178.83
409.42	99.04	233.96	56.13	616.19	123.80
2128.66	515.02	3375.66	291.88	1967.24	643.78
2664.95	286.12	1531.39	162.15	1247.28	357.65
1476.54	198.09	2383.88	112.26	2356.85	247.61
663.16	257.51	338.09	145.94	636.50	321.89
23.83	143.06	21.41	81.08	16.51	178.83
7.03	99.04	5.48	56.13	93.80	123.80
1201.76	515.02	2084.43	291.88	1916.40	643.78
3149.16	286.12	1004.50	162.15	797.45	357.65
862.84	198.09	789.96	112.26	955.44	247.61



The SAS System

The REG Procedure
Model: MODEL1
Dependent Variable: ArcDelayPerPackage

Number of Observations Read	108
Number of Observations Used	108

Note: No intercept in model. R-Square is redefined.

Analysis of Variance					
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	1	230978942	230978942	256.30	<.0001
Error	107	96428004	901196		
Uncorrected Total	108	327406945			

Root MSE	949.31359	R-Square	0.7055
Dependent Mean	1428.32041	Adj R-Sq	0.7027
Coeff Var	66.46363		

Parameter Estimates					
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
ArcHeadwayFromAnaly	1	5.46436	0.34132	16.01	<.0001

Figure 3.3. Output from Regression of Package Waiting Time on Truck Headway

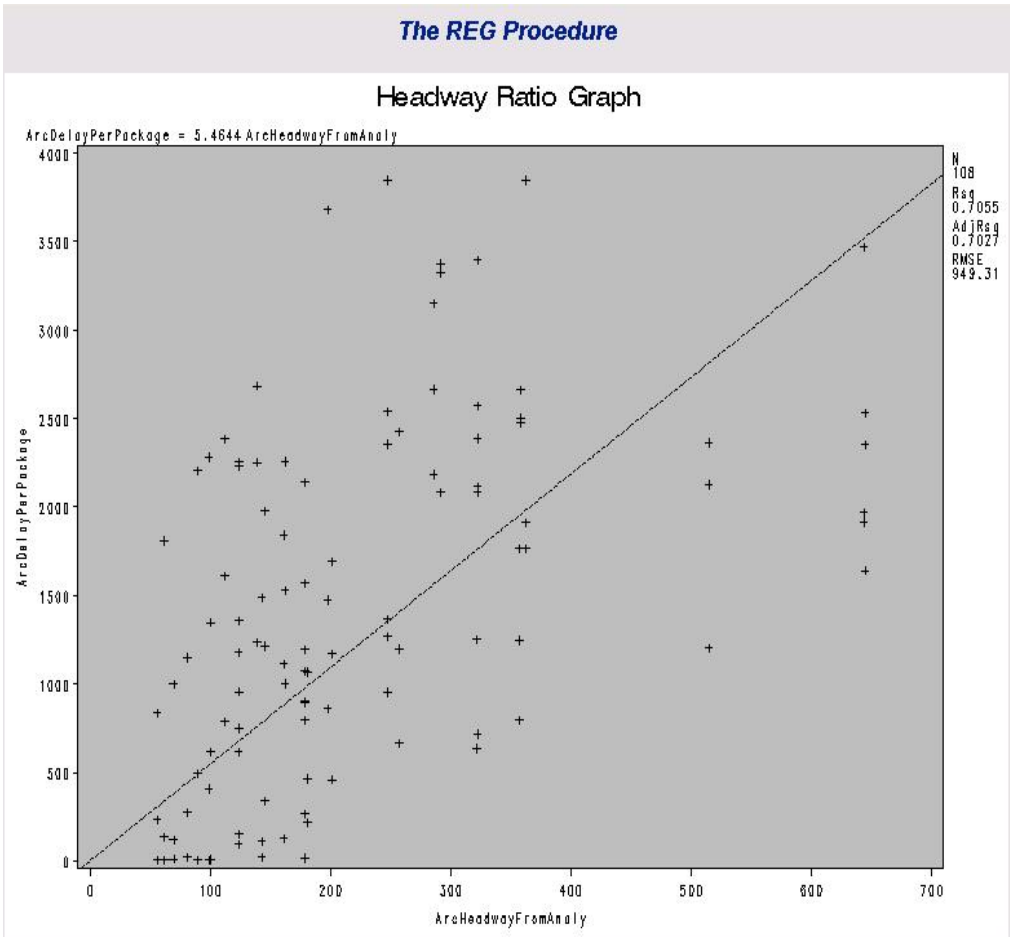


Figure 3.4. Headway Ratio Regression Plot

3.3 Conclusion

The package average waiting time is used as a criterion to design the optimal PLN, which requires a fast and simple way to approximate package average waiting time in a PLN. Kingman's equation can approximate the package average waiting time in a G/G/1 queue. The analytical and simulation models have been developed to verify that this approximation is fairly accurate for a PLN.

Kingman’s approximation suggests that in a PLN, the headway ratio should be around 6. A regression analysis of the package average waiting time on truck headways across different PLNs with different parameter sets showed the headway ratio is 5.5, which is close to the value 6 suggested by Kingman’s equation. Therefore, a headway ratio of 5.5 and truck headway can be used to quickly and accurately approximate the package average waiting time in order to use it as a criterion to obtain the optimal PLN.

3.4 Future Work

Future work can be focused on designing the optimal PLN in terms of package average transport time. There are two types of PLN design problems. The first one is to design a PLN with a fixed number of DCs and fixed locations for those DCs. The second one is to design a PLN in an area (in the U.S.A.) without knowing the optimal number of DCs and their optimal locations. The second problem is a lot harder than first one. The binary conventional genetic algorithm is a good heuristic optimization method, and we will attempt to apply it to solve those complicated problems.

4 Simulation with Bidding Scheme and Agent Behavior

The simulation with a bidding scheme and agent behaviors is an extension to the basic simulation model described in Section 2.4.

In a PLN, each package and each truck can be equipped with an agent, which is an onboard microchip with intelligent software. This agent determines the behavior of packages and trucks.

When a package is created, it is assigned a random self value and time value. Self value (unit: $) represents how much this package is worth, and time value (unit: $/hour, which means how much it would cost this package by spending one hour in the PLN) characterizes how urgently this package needs to get to its destination. For example, a package may contain a computer chip worth $1,000, but its time value is $0.1/hour since it is not in a hurry to get to its destination. On the other hand, a package may contain a surgery knife that is needed for surgery soon and may have a self value of just $15, but its time value could be $5/hour since it needs to be rushed to its destination.

Each package bids for each trip it takes. In the assembly queue (refer to Section 3.2.2.3 for details about this queue), packages are ranked by their bids. A higher bid gives that package higher priority in queues. A load is a group of packages currently in the package queue. The number of packages in the load can range from 1 to the truck capacity. The bid from a load is simply the summation of bids from each package in that load. The truck agent decides which load this truck will transport. Package and truck agents collect information provided by DCs to make those decisions. For example, in Figure 1.1, one truck is currently in DC 1, and its agent collects information, such as how many packages are currently in DC 2 and DC 6 (DCs that connect directly to DC 1) and what those load bids are. This agent may decide to take a lower bid load at DC 1 to go to a DC with a high volume of packages in order to make more profit at that DC. A series protocols have been developed by Kay for entities in a PLN to abide by and work with each other logically. For detailed PLN protocols information, refer to Kay [1].

DCs play a vital role in a PLN. They provide DC services, such as bidding information, truck offering support, truck estimated arrival time, etc., to packages and trucks. One thing to notice is that if every package bids the same value, a PLN defaults to the first-in-first-out (FIFO) system.

4.1 Package Bidding and Truck Offering Design in the Simulation Model

In the basic mode, each package bids the same value, and the PLN queues default to FIFO queues. In the advanced mode, each package bids according to its self value (how much this package is worth). Figure 4.1 reflects the truck offering and package bidding procedure that occur during a period of time from a truck's arrival at a DC until its departure. The procedure depicted in Figure 4.1 involves all the protocols, and it is the core procedure in the implementation.

The general flow is as follows. When a truck reaches a DC, if it is travelling empty, then the truck is sent directly to the truck-waiting-for-packages area. If this truck has some packages onboard when it arrives, a function named *ReceivePackage* is called to remove the packages from the truck, route those packages, and then put them in the corresponding loading areas. At the same time, some statistics, such as the truck load factor, etc., are updated. Then the packages, including new packages and packages already in the loading areas, start the bidding process (this process is detailed in Figure 4.2 and explained later). After the package bidding process finishes, the load is offered to trucks to see if any truck accepts it. If any truck accepts this load, a function named *LoadTruck* is called to move this load onto the truck, and the truck then leaves immediately. At the same time, some statistics like truck income, the total distance this truck travels, etc., are updated. If this load is not accepted by any trucks, then it is held until all other loads are offered to trucks.

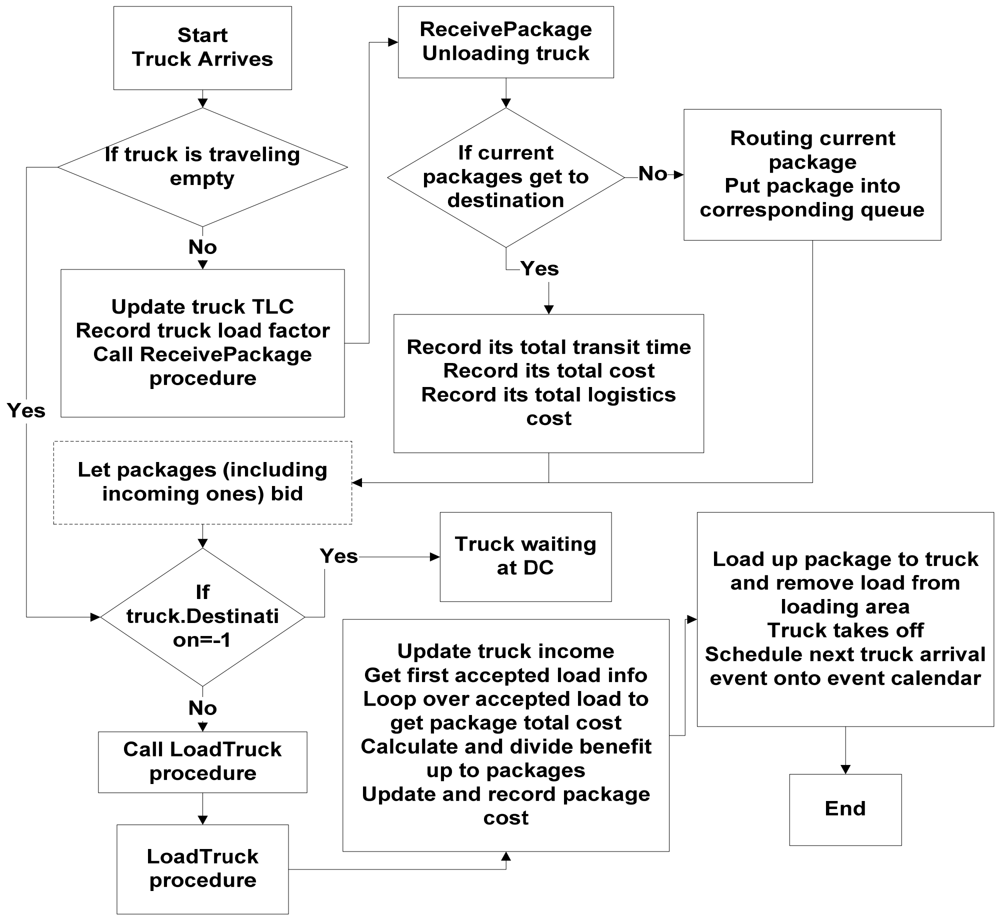

Start
Truck Arrives
If truck is traveling empty
No
Yes
Update truck TLC
Record truck load factor
Call ReceivePackage procedure
ReceivePackage
Unloading truck
If current packages get to destination
No
Routing current package
Put package into corresponding queue
Yes
Record its total transit time
Record its total cost
Record its total logistics cost
Let packages (including incoming ones) bid
If truck.Destination=-1
Yes
Truck waiting at DC
No
Call LoadTruck procedure
LoadTruck procedure
Update truck income
Get first accepted load info
Loop over accepted load to get package total cost
Calculate and divide benefit up to packages
Update and record package cost
Load up package to truck and remove load from loading area
Truck takes off
Schedule next truck arrival event onto event calendar
End

Figure 4.1. Program Flow Chart

Figure 4.2 represents the dashed box in Figure 4.1. It details the package bidding and truck offering process. The key idea for package bidding is, whenever a new package joins a queue, it gets a chance to bid, and also the packages that are already in that queue get a chance to rebid (the package's rebid can be less, equal, or more that its initial bid). After each package bids, the whole package assembly queue will be sorted in descending order by each package's bid.

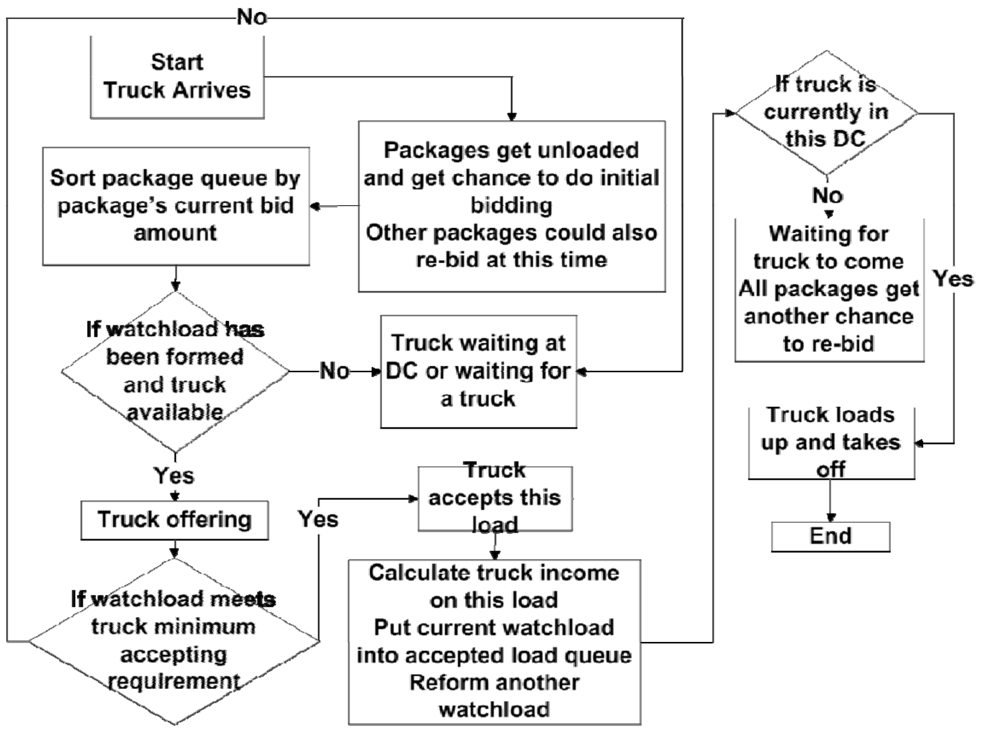

No
Start
Truck Arrives
Packages get unloaded and get chance to do initial bidding
Other packages could also re-bid at this time
Sort package queue by package's current bid amount
If watchload has been formed and truck available
No
Truck waiting at DC or waiting for a truck
Yes
Truck offering
If watchload meets truck minimum accepting requirement
Yes
Truck accepts this load
Calculate truck income on this load
Put current watchload into accepted load queue
Reform another watchload
If truck is currently in this DC
No
Waiting for truck to come
All packages get another chance to re-bid
Yes
Truck loads up and takes off
End

Figure 4.2. Package Bidding & Truck Offering Procedure

4.2 Optimization for the Simulation Model

Approximately 1.8 million packages are handled per day by UPS in the southeastern U.S. [2]. To compare PLN performance with the UPS network, a simulation and computation at the same package level are desired. Advanced data structures are incorporated to make this model run efficiently.

For example, for the package ordering procedure in each loading area, simple array copy and paste functions are used in the beginning. However, even with a small number of packages per day, this is inefficient and makes the program take an excessive amount of time to run.

Incorporating the .NET framework's advanced data structures, such as linked lists and heaps, allows a more efficient algorithm to be developed.

In Figure 4.3, the package queue is pointed to by linked list A, which originally holds all the packages in that loading area. Each time a package rebids with a higher value, it is moved to a heap. After the bidding process is finished, a comparison between the linked list A and the heap takes place. A heap is an efficient data structure to order data since the maximum value is always at the top. Whichever bid is bigger between linked list A and the heap will be moved to linked list B. Then, instead of copying the entire list B to list A, the package pointer is updated to point to linked list B. In that way, linked lists A and B serve alternatively as the original and swap space.

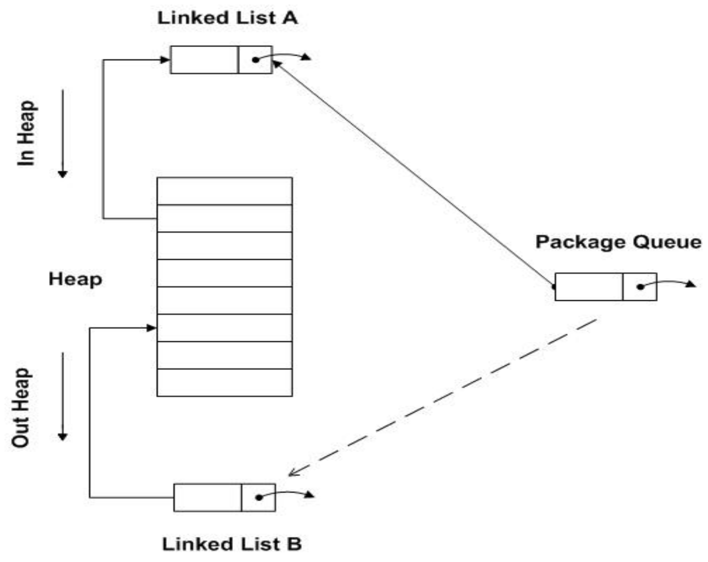


Figure 4.3. Efficient Design for Package Re-Ordering Process

By optimizing the simulation model, as shown in Table 4.1, the simulation model can simulate a large number of packages in a reasonably short time. Results in Table 4.1 are obtained by running the simulation on a computer with 1.5G Hz Intel(R) CPU, 512M RAM and 40GB hard drive.

Table 4.1. Simulation Running Speed

No. of Packages per Day	Simulation Length (day)	Running Speed (s)
9000	10	1.45
18,880	10	3.726
18,880*2	10	23.71
18,880*3	10	136.12
18,880*4	10	294.18

4.3 PLN Performance under Natural Disasters

PLN performance under natural disasters, such as hurricanes, tornadoes, etc., is evaluated by comparing it against a normal situation [35]. It is well known that UPS and FedEx use centralized control to schedule truck transport. In those networks, when under natural disasters, trucks are instructed to travel to impacted areas since demand in those areas is likely to increase. It would be interesting to see in a PLN, which utilizes decentralized control, if it can achieve this similar result.

Assume, for example, that a disaster strikes DC 5, and it happens on the fifth day and lasts for one day. We make the demand at DC 5 to increase tenfold and have the packages going to DC 5 bid three times as much as in a normal situation (since those packages are in a rush to reach DC 5).

Table 4.2. Comparison of PLN Performance between Normal and Disaster Mode

	Normal Mode	Disaster Mode	Comparison
DC Number	Truck Arrivals	Truck Arrivals	Number of trucks difference (Percentage Difference)
0	1132	1471	339 (29.9%)
1	703	803	100 (14.2%)
2	385	416	31 (8.1%)
3	1933	2185	252 (13.0%)
4	1525	1814	289 (19.0%)
5	698	973	275 (39.4%)
6	2004	2014	10 (0.5%)
7	2239	2026	–213 (–9.5%)
8	3710	3488	–222 (–6.0%)

Table 4.2 is only a portion (DCs 0 to 8) of the entire table that includes DCs 0 to 35. Compared to the normal condition, when the PLN is in the disaster condition, trucks that are going through DC 5 have increased from 698 to 973 (39.4% increase), and also the DCs around DC 5, such as DC 4 and DC 3 received many more trucks. This shows that with decentralized control, more trucks are traveling to the impacted areas under natural disaster, and that it achieves a similar result as a centralized control network.

4.4 Performance Evaluation with Protocols

In existing UPS or FedEx logistics networks, transit service is distinguished by different service class categories [35]. For simplification, in the same service class, we assume packages are served on a first-come-first-served basis, which is also called first-in-first-out

(FIFO). In a PLN, a series of protocols [1] have been implemented which encourage and favor packages with a higher bid value. Packages compete against each other by their bid amount for a faster transit service. A comparison has been carried out between these two systems (FIFO and a PLN with protocols).

4.4.1 Experiment Design for Comparison of FIFO and PLN with Protocols

The same simulation program is used for FIFO and for PLN with protocols. The only difference is the way packages bid for their trips. FIFO is implemented by simply making the bids of all packages the same. The PLN with protocols is implemented by making a package submit a bid that is proportional to its self value, which represents how much this package is worth. We run 25 replications for simulation, and each replication runs 3,600 days. The number of packages spread out over 36 DCs is 1.8 million packages per day. A total of 200 trucks (each can hold 10 packages maximum) are used. Package self values are of truncated normal distribution with a mean of \$50 and a standard deviation of \$5 (whenever a negative value is generated, it is discarded. Resample until obtain a positive value). Package time values are of truncated normal distribution with a mean of \$1.22/hour and a standard deviation of \$1.23/hour. For a PLN with protocols, 3 types of bidding styles are used, with package bids of 1/5, 1/10, and 1/50 of the self value all the time. Truck operating cost is assumed to be the same for all trucks (\$0.5/mile).

The Total Logistics Cost (TLC), which is the sum of the cost associated with the distance traveled by each truck, and the cost associated with the time each package spends in the PLN, is a criterion used to evaluate the performance of the PLN protocol:

$$\mathrm{TLC} = \sum_{i=1}^{\mathrm{nopkg}} v_i t_i + \sum_{j=1}^{n} c_{\mathrm{truck}} d_j, \tag{4.1}$$

where nopkg is the number of packages which have been delivered by the PLN, n is number of trucks, v_i is the time value (\$/hour) for package *i*, t_i is the time (hours) package *i* spends in the PLN, c_{truck} is the truck operating cost (\$/mile), and d_j is the distance (miles) truck *j* traveled.

4.4.2 Results Comparison

Table 4.3 shows the results comparing the FIFO system and a PLN with protocols. If package bids of 1/5 of the self value, TLC decreases from \$98.1 to \$89.8 million (a 8.4% decrease with standard deviation 0.5%, which is statistically significant). If package bids of 1/10 of the self value, TLC decreases from \$98.1 to \$88.5 million (a 9.8% decrease with standard deviation 0.7%). If package bids of 1/50 of the self value, TLC decreases from \$91.3 to \$85.7 million (a 6.1% decrease with standard deviation 0.2). These results show that a PLN with the bidding protocols performs better than a FIFO system in terms of total logistics cost.

Table 4.3. Comparison Result between FIFO and PLN with Intelligent Protocols (Standard Deviation is in the parenthesis)

	FIFO			PLN		
Bidding Pattern	$10 bid	$5 bid	$1 bid	1/5 of its value bid	1/10 of its value bid	1/50 of its value bid
PLN TLC ($ millions)	98.1(10.1)	98.1(10.5)	91.3(9.2)	89.8(12.5)	88.5(12.1)	85.7(11.7)
TLC Decrease				8.4% (0.5%)	9.8% (0.7%)	6.1% (0.2%)

5 PLN Network Design

5.1 Introduction

A public logistics network (PLN) has been proposed as a fast and low-cost alternative to private logistics networks for the ground transport of parcels [2, 1, 4, 3]. There is much similarity between the packages transported in the PLN and the data packets transmitted through the Internet. In a PLN, a package would be sent from a store or a house, then hop through a sequence of public distribution centers (DCs) mostly located in metropolitan areas, and finally be delivered to a home in a matter of hours [1]. The DCs, functioning like routers in the Internet, can also be located at major highway interchanges to avoid packages being transported through local city areas that are not their final destinations [2, 1].

Figure 1.1 shows a hypothetical public logistics network with 36 public DCs (this is a sketch design, in which DCs are primarily located at intersections of interstates) covering the southeastern portion of the U.S., and connected via interstate highways [2, 1].

5.1.1 Research Significance and Open Issues in the PLN Design

A PLN is an alternative logistics network with some potentially attractive features. However, since it does not exist now, a PLN designed to achieve the best performance is a key issue. The PLN design problem includes determining the number of DCs, selecting locations of those DCs, and determining the connections between the DCs in the network.

For example, in Figure 1.1, in the southeastern region of the U.S., the number of DCs needed to cover this area, where those DCs should be located, and which two DCs should have direct arc connections are key issues we are trying to solve. As shown in the graph, most of the DCs are located at the intersection of interstate highways to take advantage of the travel convenience.

In order to evaluate if a PLN, as a parcel transportation network, has some advantages over existing centralized-control networks, such as UPS, FedEx, etc., package average travel time is used as the criterion for comparison. Therefore, the goal of the PLN design problem is to design a network that results in the minimum average travel time for the packages. Certainly, other measures, such as minimal package travel time (minimize the worst case package delivery times), minimal total logistics cost, etc., can be used as criteria as well. See more discussion on this in Section 5.5.

5.1.2 Related Research

Network design techniques are used in many areas: computer network design, wireless network design, supply chain network design, transport network design, etc. The network's design and layout play a vital role in its future performance and service. Lots of effort has been contributed toward the network design problem for supply chain networks and transportation networks (see Beamon [36] and Magnanti et al. [37] for a review of research in each respective area).

The PLN design problem has some similarity to the location-allocation problem [38] (one of the important problems in supply chain network design), such as finding the optimal number of new facilities, determining the locations for new facilities, etc. But unlike the location-allocation design problem where no material flows occur between new facilities (NFs), the PLN design problem involves material flows between nearby DCs (as shown in Figure 5.1). Compared to the location-allocation problem, the PLN design problem has the extra issue of designating the direct arcs between DCs.

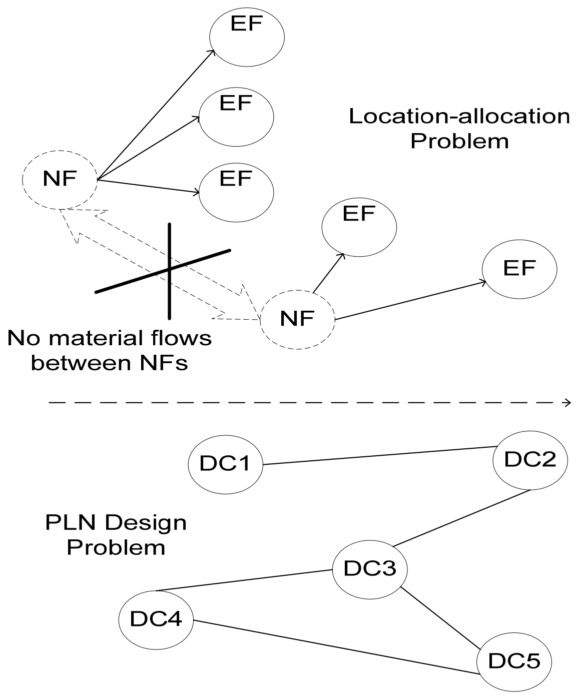


Figure 5.1. Difference between Location-Allocation Problem and PLN Design Problem

The major difference between the transportation-network design problem and the PLN-design problem is their objective functions. Transportation-network design problems typically have total operating cost as the objective function, which increases as traffic becomes denser. On the contrary, PLN-design problems have package average travel time as the objective function, which decreases as traffic becomes denser before it reaches a minimum point; more traffic on an arc results in more trucks traveling on this arc to satisfy demand, resulting in lower package average waiting times. On the other hand, less traffic results in higher waiting times. This in turn results in less traffic and eventually packages that get abandoned due to high waiting time. Wardrop's equilibrium [39] states that, under equilibrium conditions traffic arranges itself in congested networks such that all used routes between an origin-destination (O-D) pair have equal and minimum cost while all unused routes have greater or equal costs. We use Delaunay triangulation to designate the direct arc between DCs.

Less-than-truckload (LTL) shipping is the transportation of relatively small freight. Less than truckload carriers collect freight from various shippers and consolidate that freight onto enclosed trailers for line haul to the delivering terminal or to a hub terminal where the freight will be further sorted and consolidated for additional line hauls. The LTL network design problem is close to the PLN design problem in terms of designating arcs between terminals. Lamar and Sheffi [40] designed an LTL network by applying a lower bound iteratively with a link-inclusion heuristic in an implicit enumeration framework. However, the procedure of Lamar and Sheffi only applies to a small sized problem and requires intensive computation. Zhang, Wu, and Liu [41] investigated the routing problem and proposed a new mathematical model in a hybrid hub-and-spoke LTL network. They claimed that, compared to the traditional hub-and-spoke transportation network, the hybrid hub-and-spoke transportation network can perform better in terms of transportation cost with all other conditions being the same. But their LTL network differs from a PLN network in that LTL is mostly a hub-and-spoke network and the loading and unloading times are much larger components of each item's overall transport time when compared with the overall transport time of items in a PLN with a DC structure, where loading and unloading is performed by a machine automatically and takes a much shorter time [2].

Kay and Parlikad [2] analyzed material flow features in a PLN. They compared different networks (HUB, PLN, P2P) by using average package transport time. Bansal [5] proposed three mechanisms to design a PLN road network. He used a genetic algorithm to do a heuristic search for the optimal PLN design. In his work, the average package waiting time is calculated from headway with a 0.5 headway ratio. In this dissertation, a 5.5 headway ratio is used (refer to Chapter 3 for details) to estimate package average waiting time; and we extend Bansal's work by using a more comprehensive network design mechanism, investigating the PLN layout sensitivity to the PLN parameters, and designing the network using additional criteria.

5.2 PLN Underlying Road Network Design

5.2.1 Underlying Road Network Overall Process

As previously discussed, DCs are primarily located at the intersection of interstates and/or interstates and U.S. highways. This is beneficial especially for long-haul package transport since they just remain on the highway, bypassing the local streets along their way.

Before the PLN design can be carried out, a basic U.S. underlying road network is needed to start with [5]. One natural idea is to use the interstate highway network (shown as Figure 5.2, generated by using the MATLOG toolbox [42]).

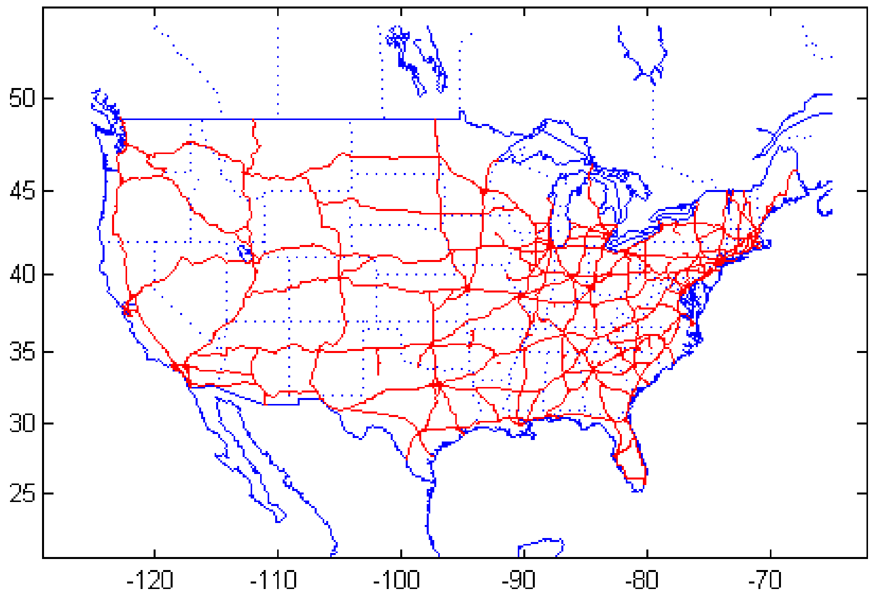


Figure 5.2. U.S. Interstate Highway Network (Excluding State AK, HI and PR)

However, although the U.S. interstate highway network appears to be reasonably dense in the east, it is too sparse in the west. Therefore, the U.S. interstate highway network is extended to include not only interstates but also U.S. highways to try to increase its density in the west (shown as Figure 5.3).

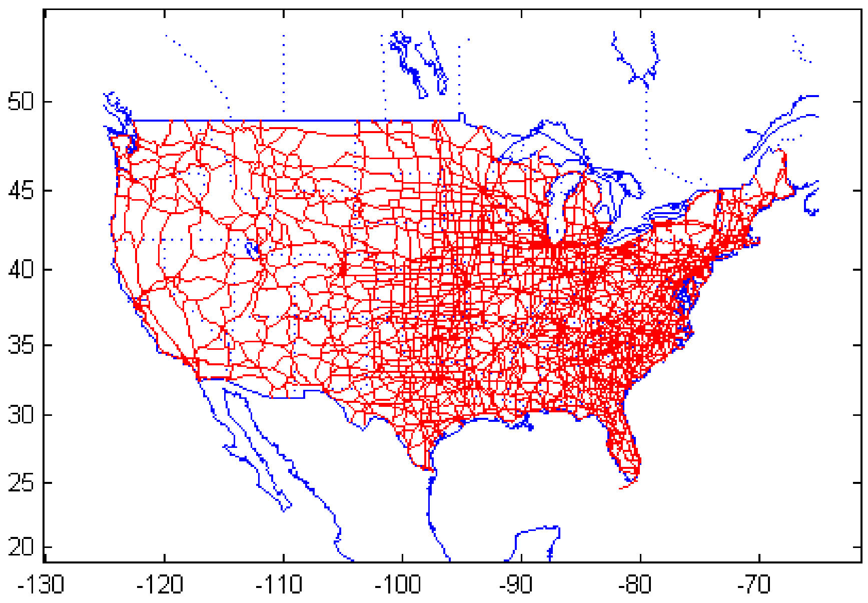


Figure 5.3. U.S. Interstate and U.S. Highway Network (Excluding State AK, HI and PR)

In Figure 5.3, there are 30,978 interstate and U.S. highway intersections (nodes). It is not possible and not economical to set up a DC at each of these nodes. The more population an area has, the more package transportation traffic it generates; and so the candidate DC location should be closer to areas with higher population when other conditions are the same. Population-weighted 3-digit ZIP code centroids (903 Zip-3 Areas) were used as references for nodes. Nodes closest to those areas were picked out as candidate nodes (termed CandiNode) to start with. The detailed design procedure flow is shown in Figure 5.4.

We start out with the interstate and U.S. highway network. *Thin procedure* is applied to that network to remove degree-two nodes, and *RemoveUnconnNodes procedure* is used to eliminate isolated nodes from this network. As stated above, we choose some intersections from 30,978 interstate and U.S. highway intersections as candidate DC locations. A 3-digit ZIP code centroid is used to determine which one to choose (intersections that are closest to any of those centroids are picked out and called CandiNodes). Delaunay triangulation is used to create an IJD array (an IJD array characterizes the direct connection between two points in a graph and their corresponding distance. Each row of the IJD array has three elements, the start point, the end point of the direct connection and the corresponding distance for that direct connection) for those CandiNodes. Dijkstra’s algorithm is applied to find out nodes along the path from one CandiNode to another CandiNode, and those nodes are added to the CandiNode group. *Addconnector procedure* is used to connect the isolated CandiNodes to the network. *Subgraph procedure* is applied on CandiNode groups to create a final underlying road network for the PLN design.

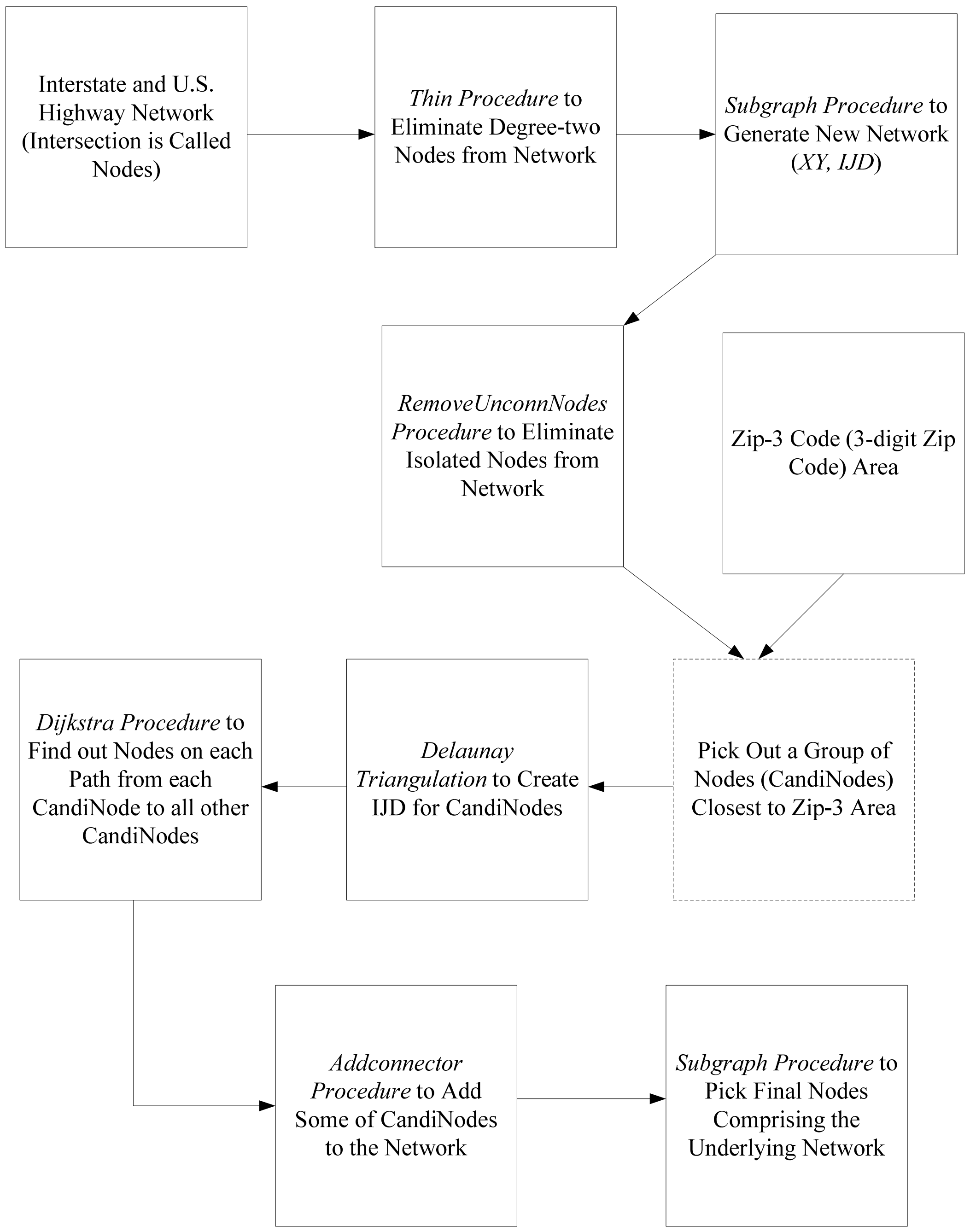

Interstate and U.S. Highway Network (Intersection is Called Nodes)
Thin Procedure to Eliminate Degree-two Nodes from Network
Subgraph Procedure to Generate New Network (*XY, IJD*)
RemoveUnconnNodes Procedure to Eliminate Isolated Nodes from Network
Zip-3 Code (3-digit Zip Code) Area
Pick Out a Group of Nodes (CandiNodes) Closest to Zip-3 Area
Delaunay Triangulation to Create IJD for CandiNodes
Dijkstra Procedure to Find out Nodes on each Path from each CandiNode to all other CandiNodes
Addconnector Procedure to Add Some of CandiNodes to the Network
Subgraph Procedure to Pick Final Nodes Comprising the Underlying Network

Figure 5.4. PLN Underlying Network Design Flowchart

5.2.2 Underlying Road Network Detailed Design

5.2.2.1 Degree-two Thinning

Adding and removing arcs results in some nodes connected by two direct arcs, which can be eliminated to reduce the number of nodes and improve average package travel time. As shown in Figure 5.5, node 2 and node 3 are classified as degree-two nodes since there are only two arcs connecting them to other nodes. By applying degree-two thinning, node 2 and node 3 are eliminated to reduce the number of nodes.

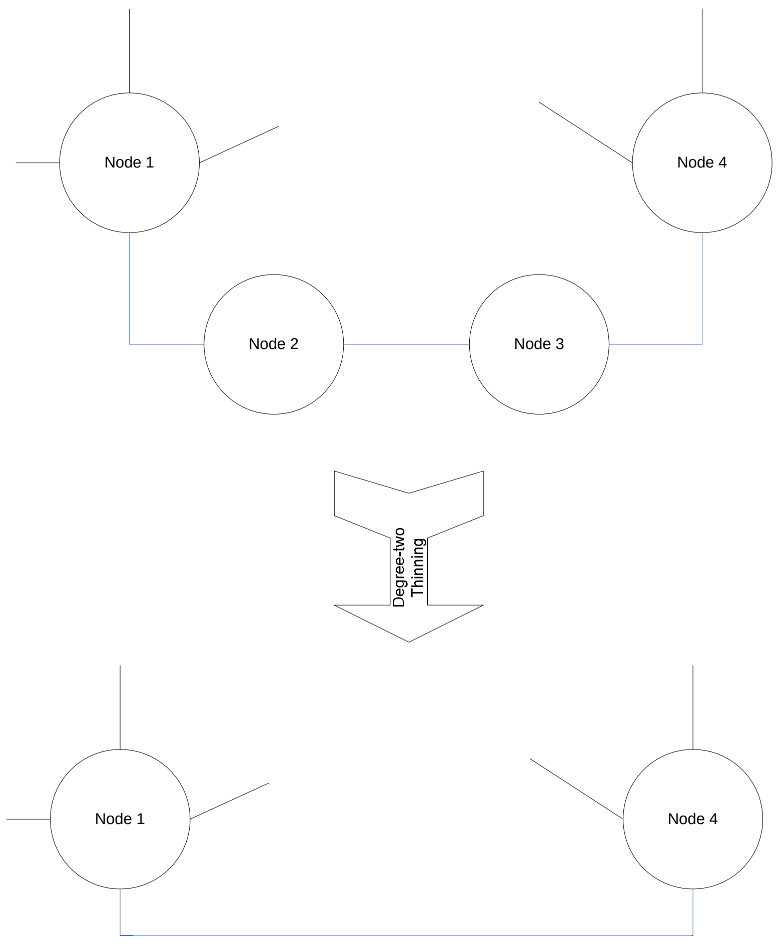


Figure 5.5. Degree-two *Thinning Procedure*

5.2.2.2 Remove Isolated Nodes from Network

In degree-two thinning, removing arcs results in some nodes that are not connected to any part of the main network (isolated). *RemoveUnconnNode* procedure applies Dijkstra's algorithm to compute the shortest distance between any two nodes. If any of the distances turns out to be infinity, then this situation involves an isolated node that must be removed from the network.

5.2.2.3 Selection of CandiNodes

This section explains the design logic for the dashed line box in Figure 5.4.

For each zip-3 area, the closest node is located by the great circle distance to the centroid of that area. In Figure 5.6, since D2 is smallest among D1, D2, and D3, node 2 is selected as the CandiNode. The number of CandiNodes in the resulting set of CandiNodes has the same number with the number of zip-3 centroid.

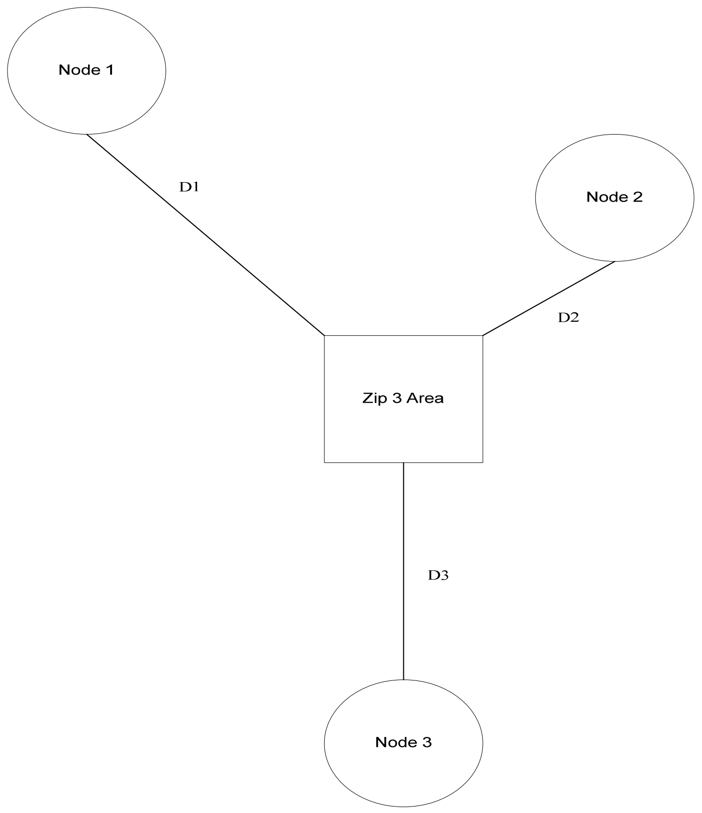


Figure 5.6. Selection of CandiNode

After selecting the CandiNodes, Dijkstra's algorithm is applied to each pair of CandiNodes in the current set of CandiNodes to find out the shortest path between each pair of CandiNodes.

Any nodes on these shortest paths are selected for inclusion in the current set of CandiNodes. As shown in Figure 5.7, node 1 and node 2 are on the shortest path from CandiNode 1 to CandiNode 2; therefore, node 1 and node 2 are selected as CandiNodes as well. For any node, if it is not on any path between two CandiNodes, it is eliminated (e.g., node 3 and node 4 are eliminated).

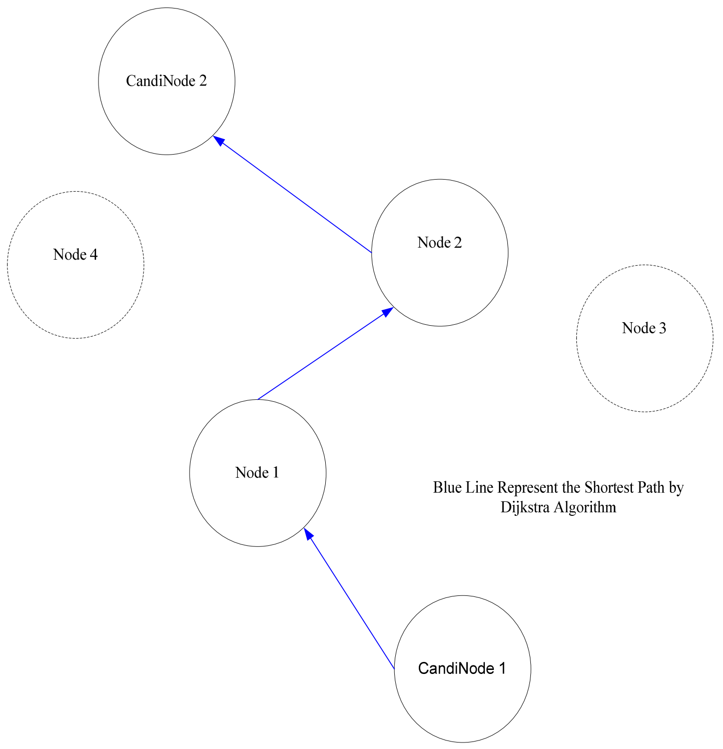


Figure 5.7. Nodes along the Shortest Paths Are Selected as CandiNode

5.2.2.4 Delaunay Triangulation

In mathematics, a Delaunay triangulation DT(*P*) for a set *P* of points in the plane is a triangulation such that no point in *P* is inside the circumcircle of any triangle in DT(*P*).

Delaunay triangulations are used to maximize the minimum angle of all the angles of the triangles in the triangulation. Delaunay triangulation tends to avoid skinny triangles [43].

After determining the CandiNodes, Delaunay triangulation is used to designate the direct arc between any two nodes. As shown in Figure 5.8, Delaunay triangulation connects any two nodes to maximize the minimum angle of all the angles of the triangles.

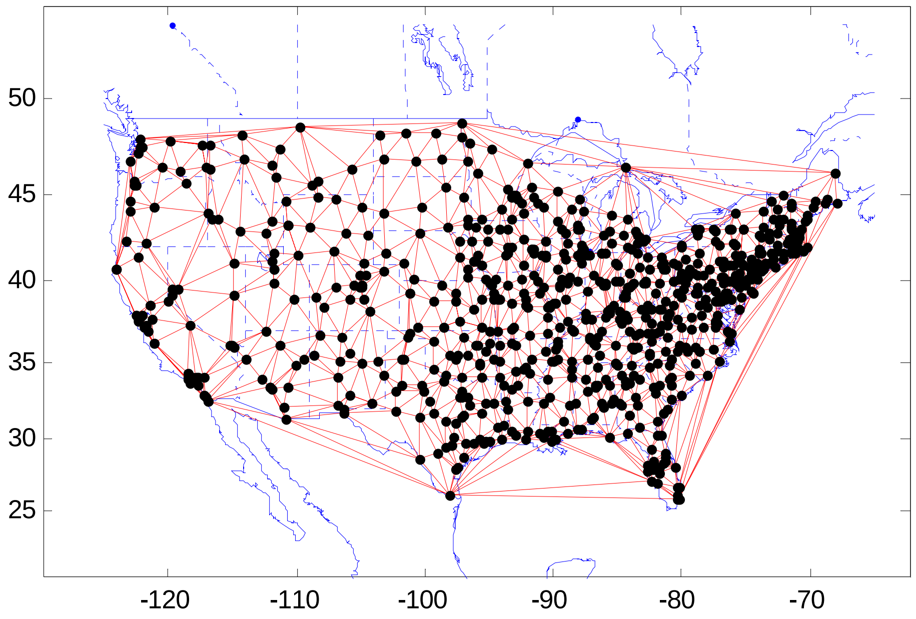


Figure 5.8. Delaunay Triangulation for CandiNode in a PLN Underlying Network

5.2.2.5 Remove Nodes That Are Too Close to Each Other

Another issue is that some of the nodes are very close to each other (namely, the distance between them is less than 25 miles, which is chosen by experimentation). It is not economical to set up two DCs for two locations that are so close together; therefore, a trimming procedure is used to eliminate these “too close” nodes. The resulting network has nodes that are at least 25 miles away from each other.

5.2.2.6 Add Some CandiNodes to the Network

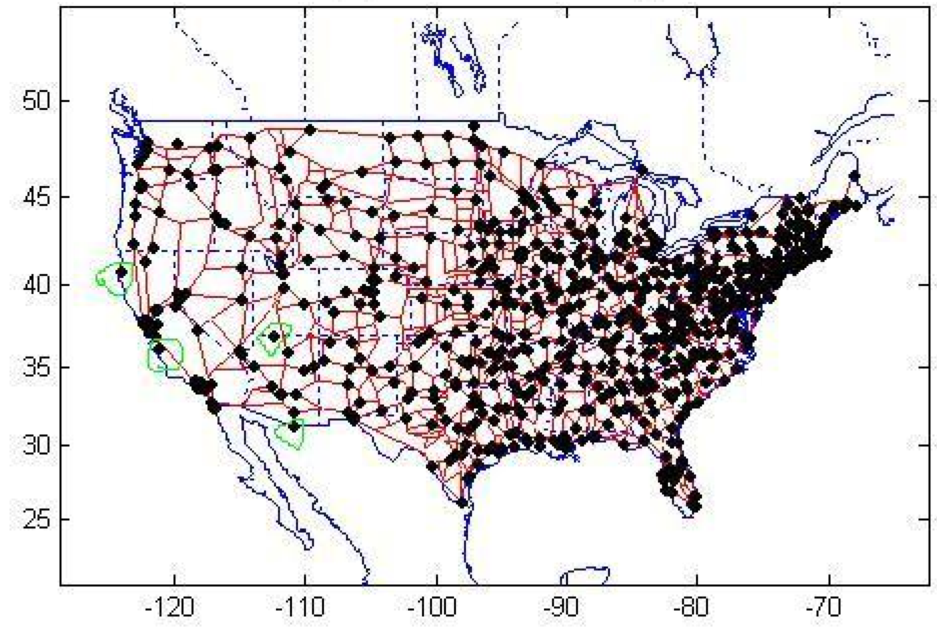


Figure 5.9. Network without Applying *Addconnector Procedure*

The *Addconnector procedure* is used to add connectors from new locations to the nodes of the transportation network. An arc between a new location and the node in the original network is added to the combined network. Whichever node is closest to the new location will be directly connected to the new location [42]. As shown in Figure 5.10, node 5 originally is isolated from the rest of the network. After the *Addconnector procedure* is applied, node 5 now is connected to the whole network (since node 4 is closest to node 5, node 5 is directly connected to node 4).

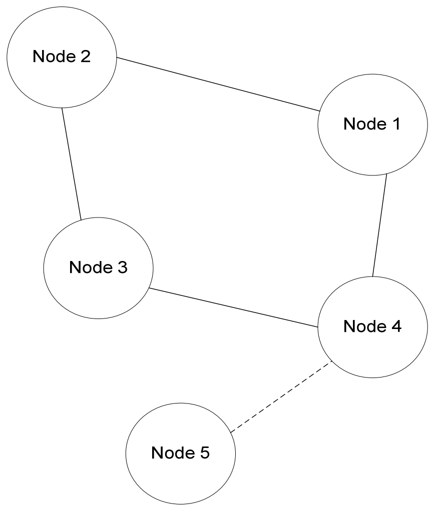


Figure 5.10. *Addconnector Procedure* Illustration

The *Addconnector procedure* was applied to add those nodes that are not included in the network generated by the Delaunay Triangulation (shown as Figure 5.9, circled in red).

5.2.2.7 Creating the PLN Underlying Road Network

The PLN underlying road network is composed of CandiNodes that can potentially be selected as DCs. Due to the characteristics of a PLN, those DCs should mainly be located at the intersection of interstates and/or interstates and U.S. highways, not just at cities with large population. After using the procedure described above, a PLN underlying road network is created (shown in Figure 5.11) and is ready to be applied to design the optimal PLN.

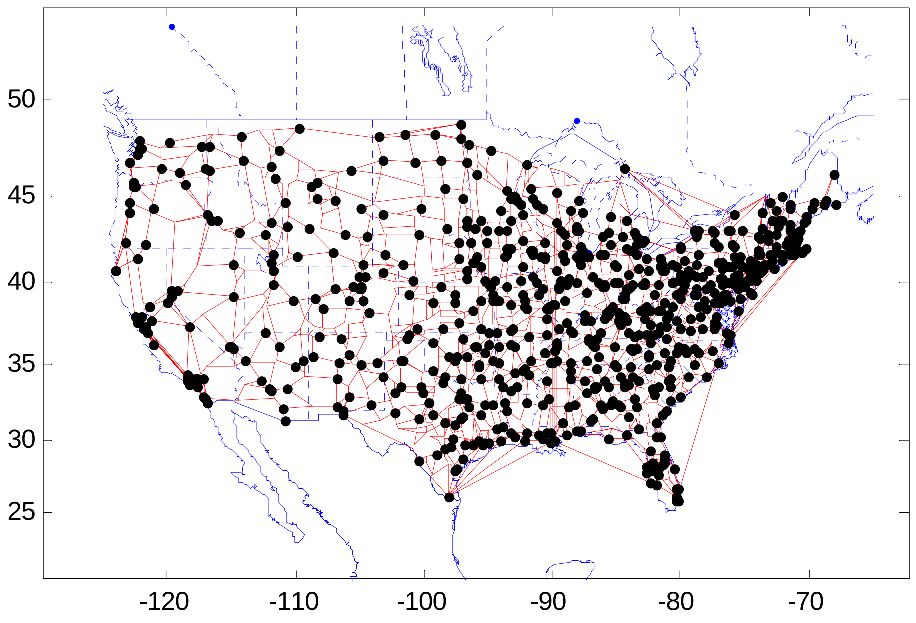


Figure 5.11. Underlying Road Network for PLN Design

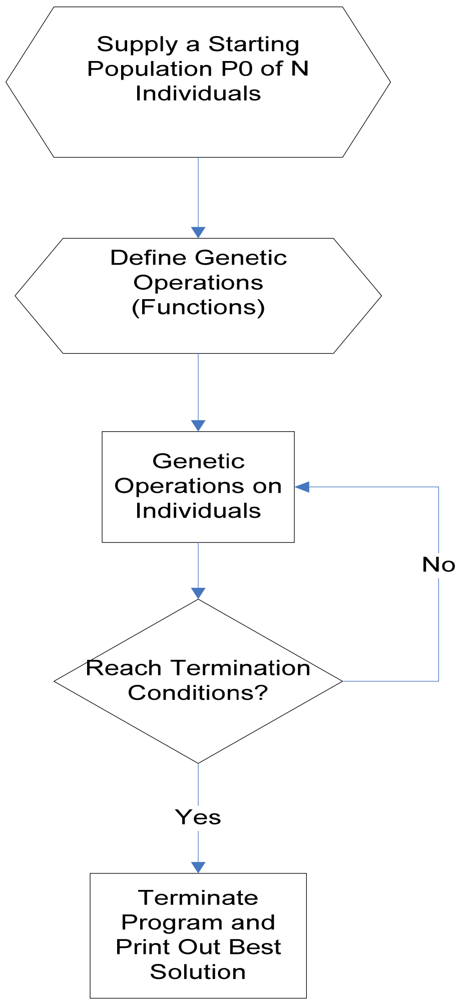
5.3 Genetic Algorithm

5.3.1 Introduction

A genetic algorithm (GA) is a widely used search technique to find exact or approximate solutions to optimization and heuristic search problems. Genetic algorithms are a particular type of evolutionary algorithm that uses procedures inspired by natural evolutionary biology in the real world, such as inheritance, selection, crossover, and mutation [44, 45, 46].

Genetic algorithms search the solution space of a function by simulating evolution, such as the survival of the fittest individuals through generations. It is widely believed that the fittest individuals in any population tend to reproduce and survive to the next generation, thus improving their offspring. However, by performing selection, crossover, and mutation, some of those inferior individuals can survive and reproduce. Genetic algorithms have been shown to efficiently and quickly solve linear and nonlinear problems by exploring all regions of the solution space [47,48,49,50,51].

A general genetic algorithm procedure flow chart is given in Figure 5.12.



Supply a Starting Population P0 of N Individuals
Define Genetic Operations (Functions)
Genetic Operations on Individuals
No
Reach Termination Conditions?
Yes
Terminate Program and Print Out Best Solution

Figure 5.12. Genetic Algorithm Procedure Flow

5.3.2 GA’s Suitability for PLN Design Problem

Genetic algorithms were chosen to solve the PLN-design problem because of their popular applications for network design problems. Compared with other heuristic search techniques, genetic algorithms have superior global search capabilities, and a high degree of flexibility for objective functions and constraints [44, 49].

Traditionally, network design problems were solved by linear programming, integer programming, and mixed linear/integer programming techniques [52, 53, 54]. However, as a problem’s scale and complexity increase dramatically, these techniques cannot handle them effectively. Heuristic search methods, especially genetic algorithms, gained more and more popularity in tackling these complex problems [55]. Gen et al. [55, 56] provides a comprehensive review on research of network design using genetic algorithms. It covers computer communication network design, transportation network design, network topological design, etc. Vitetta [57] used a genetic algorithm to tackle a road network design problem. He incorporated multiple criteria (total travel time, road traffic flow, and total length of modified links belonging to primary networks) into a complex objective function. He found out that in effect, a small reduction of travel time is accompanied by high values for other criteria. Cunha and Silva [58] proposed a genetic algorithm to design a LTL network that consists of determining the number of hubs, their locations, and the assignment of the spokes to the hubs, aiming to minimize the total cost.

For the PLN design problem, there are no constraints on the solutions, and it has great decomposability (one sub-graph structure has little effect on other distant sub-graph structures). Although the PLN design and the LTL network design have some significant differences, they share some similarities, such as large problem scales and nonlinear objective functions.

5.3.3 Genetic Operations

5.3.3.1 Selection

During each generation, the GA selects a portion of the existing population to create a new generation. Fitter solutions that are measured by a fitness function are typically more likely to be selected as the parents to generate offspring [44]. The selection of existing individuals to generate successive generations plays a vital role in a genetic algorithm. An individual in the population can be selected more than once during this process and can be put in the next generation directly. There are several popular schemes for the selection process: roulette wheel selection, scaling techniques, ranking methods, etc. [48, 50, 51]. After some individuals are selected, the next step is to create a second generation population through the genetic operators of crossover and mutation.

5.3.3.2 Crossover

Crossover is a genetic operator used to vary the form of a chromosome or chromosomes from one generation to the next. Normally, two parents' chromosomes cross over at a pre-determined point to generate two new chromosomes. It is similar to the chromosomes reproduction and biological crossover upon which genetic algorithms are based. Some popular crossover operations are one-point, two-point, and “cut and splice” crossovers [44, 47, 48].

5.3.3.3 Mutation

Mutation changes some randomly selected part of chromosomes, which is analogous to biological mutation. Mutation is a genetic operator used to maintain genetic diversity among a population from one generation to the next. Whether an arbitrary bit in a genetic sequence is changed from its original state depends on the mutation probability. A common method of implementing the mutation operation is to generate a random variable (from some desired distribution) for each bit in a chromosome, which determines whether that particular bit will be changed [44, 47, 48].

5.4 GA Application on PLN Design

5.4.1 GA Implementation Package

This application used GAOT (Genetic Algorithm Optimization Toolbox) for MATLAB, which implements simulated evolution processes in the MATLAB environment using binary, floating, and real representations for individual chromosomes. GAOT implementation is flexible in its genetic operators, fitness evaluation functions, selection functions, and termination functions that can be chosen and used [51,59].

5.4.2 Solution Structure Representation

A CandiNode, whose creation method is described in Section 5.2, is randomly selected as a DC based on its population's weight. A CandiNode with a bigger population has a larger chance to be selected and will comprise the initial GA population. A binary representation for each solution is used. For example, if the initial chromosome has 2,000 bits, for each bit:

0: this CandiNode is not chosen as a DC

1: this CandiNode is chosen as a DC

The total number of 1-valued bits in a chromosome is the total number of DCs that will cover the contiguous U.S. The locations of those DCs are also determined since the longitude and latitude for those nodes are known.

5.4.3 Fitness Evaluation Functions and GA Parameters

Each chromosome is a binary sequence, which represents a PLN layout. Each chromosome is evaluated by the analytical model [60, 61, 62] to obtain average travel time for packages from their origin to their final destination (this average travel time for packages is taken over all packages delivered in a 24-hour period) for this corresponding PLN layout. This average package travel time for the current chromosome (PLN layout) is used as a fitness value, which determines if one solution is better than the other.

Note that for a chromosome (a PLN layout), since we know the binary sequence for that layout, therefore we know the DCs locations, the population each DC covers, the direct connections between two DCs, and their corresponding distance, etc. We can run analytical model and use Equation (2.2) to compute the packages average travel time for current PLN layout. In the analytical model, the truck average headway is calculated by Equation (2.10) and the headway ratio is 5.5 (refer to Chapter 3 for detail about how the headway ratio is obtained).

The following GA options and parameters are used in this PLN design:

Population size: 30

Roulette wheel selection

Simple crossover operation

Binary mutation with probability 0.5%

Maximum generation 50 as termination operator

Minimum difference threshold 1e-6

5.5 Results

For the analytical model, the following parameters [60, 61, 62] were used to compute average package travel time:

Loading and unloading time for trucks at each DC is five minutes, TimeLU = 5;

Proximity factor (which determines package demand distribution among DCs) is 2, ProxFac = 2;

Number of packages to transport per day in the PLN is $1x10^7$ (which is approximately the totally package UPS handles in one day), Nopkg = 1E7;

Truck capacity is 200, TrCap = 500;

Truck travel speed is 60 mph, TrSpeed = 60;

Truck average load factor is 80% (on average, trucks are 80% full), LdFac = 0.8;

Headway ratio is 5.5 (product of headway ratio and headway is the approximation for the package average waiting time [62]), HWRatio = 5.5;

5.5.1 Optimal Number of DCs in a PLN

The GA program runs three replications and it reports that the optimal number of DCs for the whole U.S. is 128, which results in an average package travel time of 19.8 hours.

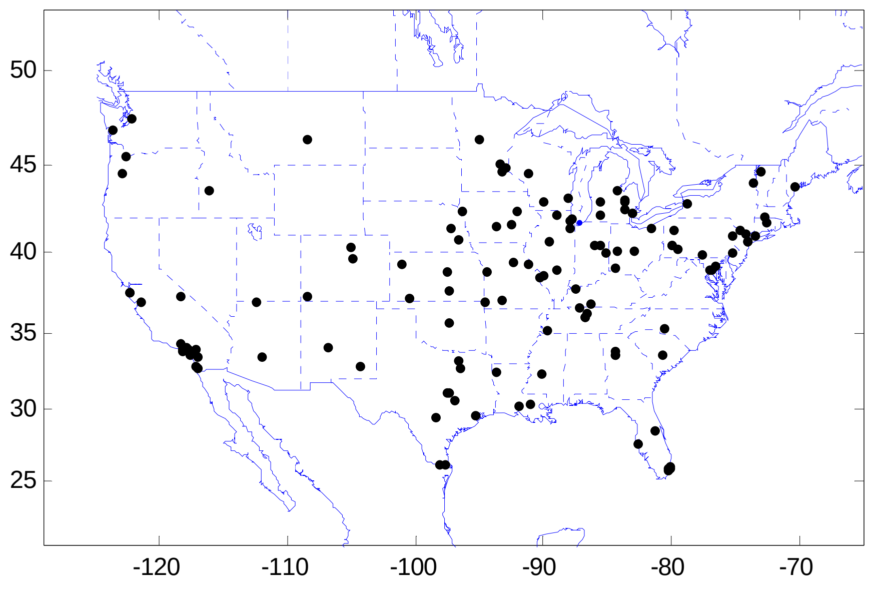


Figure 5.13. Optimal DCs Layout across U.S. Contiguous States (TimeLU=5, Nopkg=1x10^7)

Figure 5.13 shows the optimal DC layout (128 DCs, the dark dots on figure) across the U.S. contiguous states.

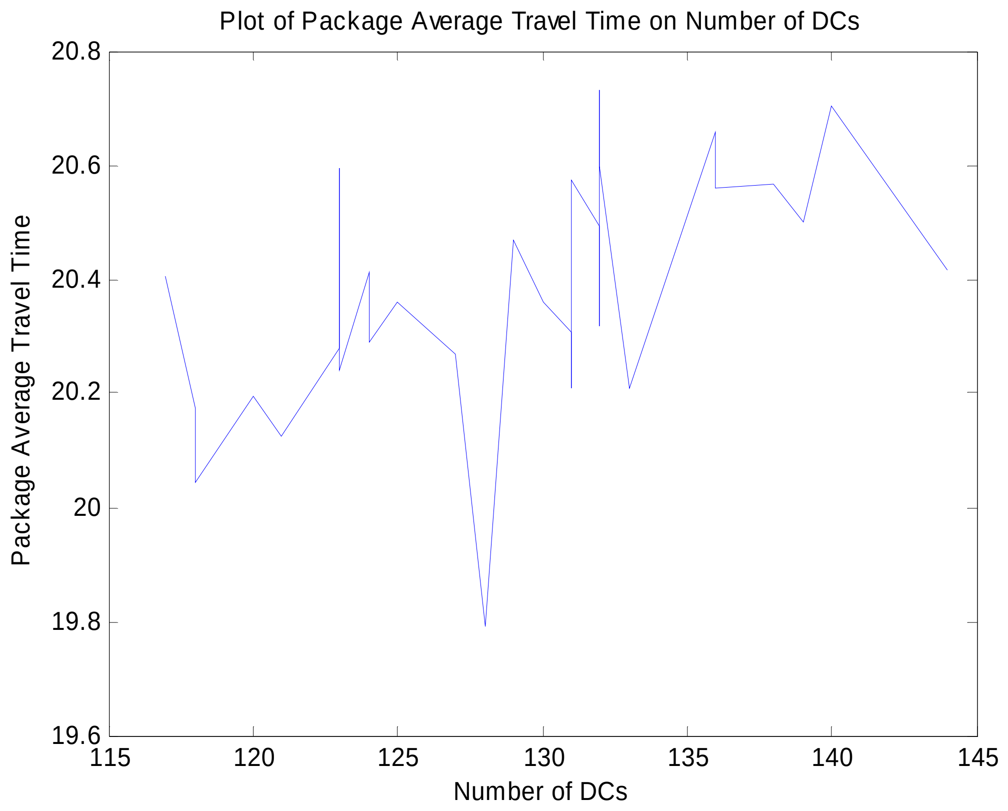


Figure 5.14. Plot of Average Package Travel Time on Number of DCs in the PLN (TimeLU=5, Nopkg=1x10^7)

Figure 5.14 shows the relationship between the average package travel time and the number of DCs in the PLN. At point 128 on the Number of DCs axis, we found the optimal solution for average package travel time.

5.5.2 Comparison between Two U.S. Southeastern PLN Graph

The 36-DC PLN graph is a hypothetical design that does not try to minimize the package average travel time. It is a balanced design between the population weight of those DCs and their locations.

In the 128-DC PLN, the southeastern part (longitude –87.0970 to –70.8554, latitude 28.9514 to 39.5132) is composed of 24 DCs. Figure 5.15 shows the difference between the 36-DC PLN graph, and the southeastern part of optimal PLN, which minimizes the average package travel time.

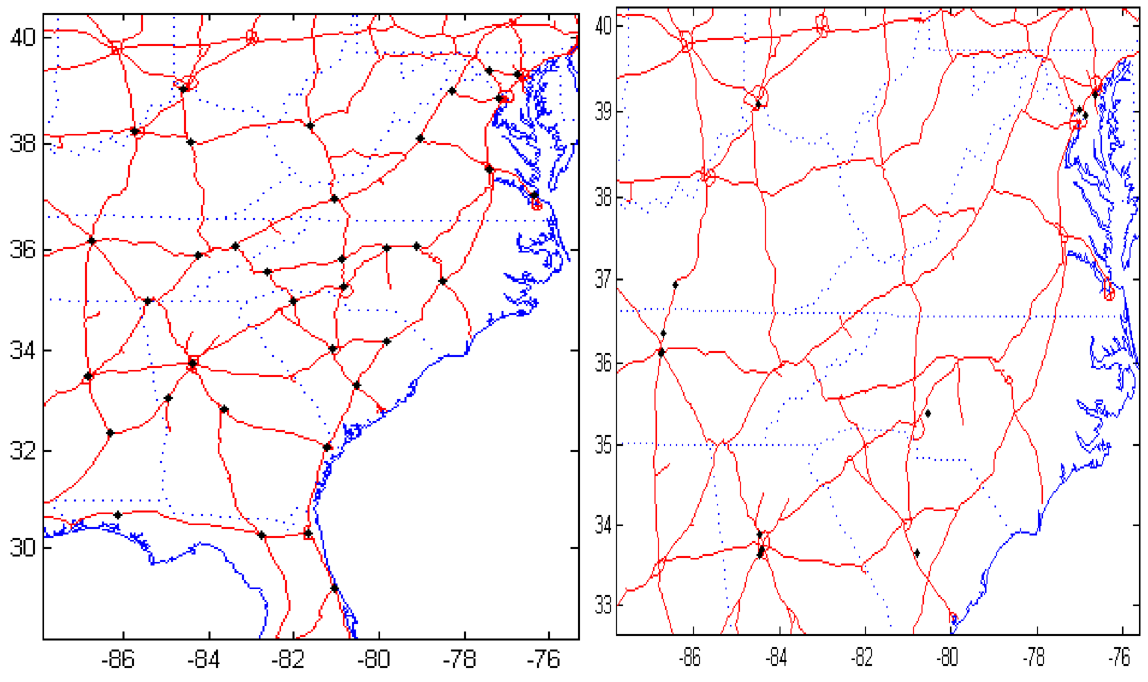


Figure 5.15. Comparison (36 VS 13 DCs) between two U.S. Southeastern Part PLN (TimeLU=5, Nopkg=1x10^7)

5.5.3 Parameters Sensitivity for PLN Design

The optimal PLN design varies with different parameters. We increased the total package demand by a factor of 5 (Nopkg=5x10^7) as in the example shown above. The optimal number of DCs across the whole U.S. turns out to be 195 (shown in Figure 5.16) with the package average travel time 17.6 hours, and the number of DCs covering the southeastern region of the U.S. is 20 (shown in Figure 5.17).

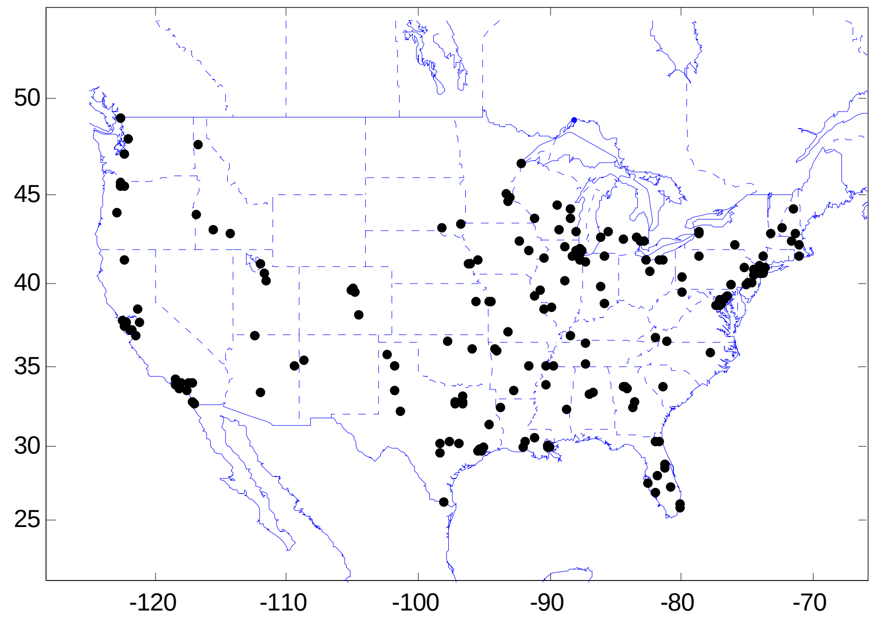


Figure 5.16. Optimal DCs Layout across U.S. Contiguous States (TimeLU=5, Nopkg=5x10^7)

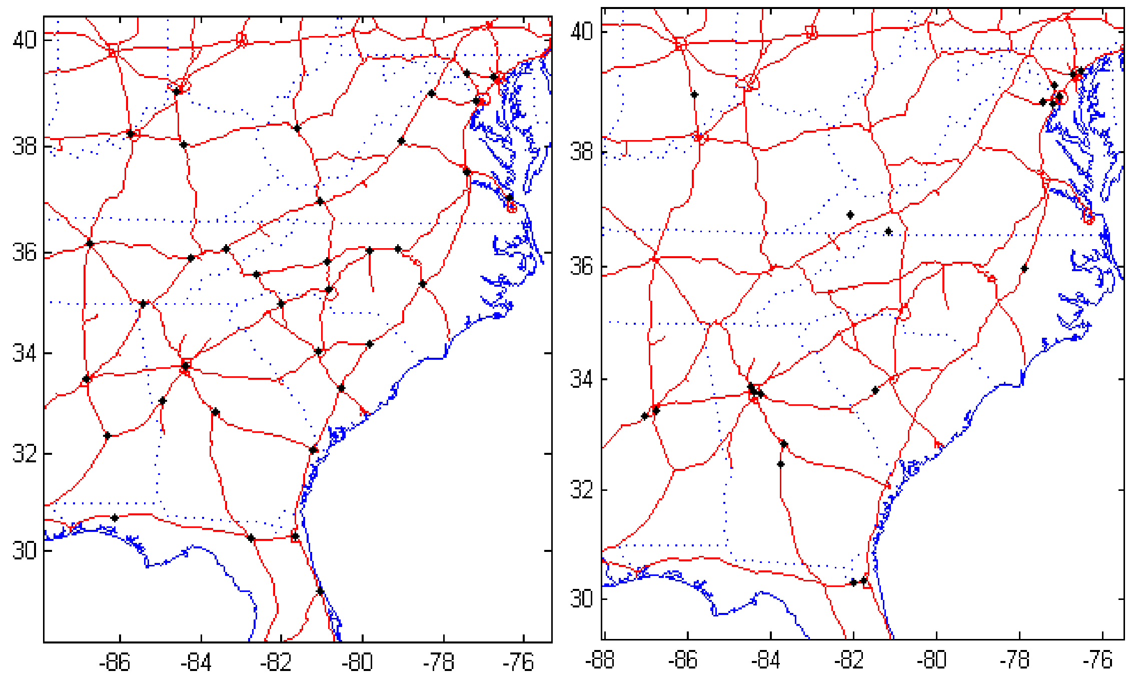


Figure 5.17. Comparison (36 vs. 20 DCs) between two U.S. Southeastern Part PLN (TimeLU=5, Nopkg=$5x10^7$)

Table 5.1 shows the result for the PLN's optimal number of DCs across the continental U.S. in different parameters. If total package demand increases, it will require more DCs and trucks to meet the demand in a PLN. If truck loading and unloading times increase, then it takes more time for trucks to stop at DCs (it increases package average travel time when other conditions are the same), therefore this will decrease the number of DCs to make a trade off. The same analysis can be carried out for other parameters.

Table 5.1. Summary Result for PLN Optimal Number of DCs in Different Parameters

PLN Optimal No. of DCs	Nopkg=1x10^7	Nopkg=5x10^7
TimeLU=5	128	195
TimeLU=10	124	171

5.5.4 Other Constrains Imposed on PLN Design

From Figure 5.15, we can see that some of the DCs are clustered together in some metro areas, which makes sense since those areas have a larger population and larger demand. This clustering reduces the number of DCs in the more sparsely populated areas, which will require local transport to travel longer distances to reach the closest DC, and for local regions to take longer to deliver. In order to make the DCs more spread out thus reducing the longest delivery time, we imposed a constraint on the minimum distance between any pair of DCs. Any two DCs must be at least 50 miles away from each other. We tried 25 miles, but that turned out to be too short. We found that 50 miles is far enough away. When we generate the PLN underlying network, the distance from each CandiNode to another is calculated. If any distance less than 50 miles is found, then those CandiNodes with less population will be eliminated, which ensures at least a 50-mile distance between any two CandiNodes in the final underlying network.

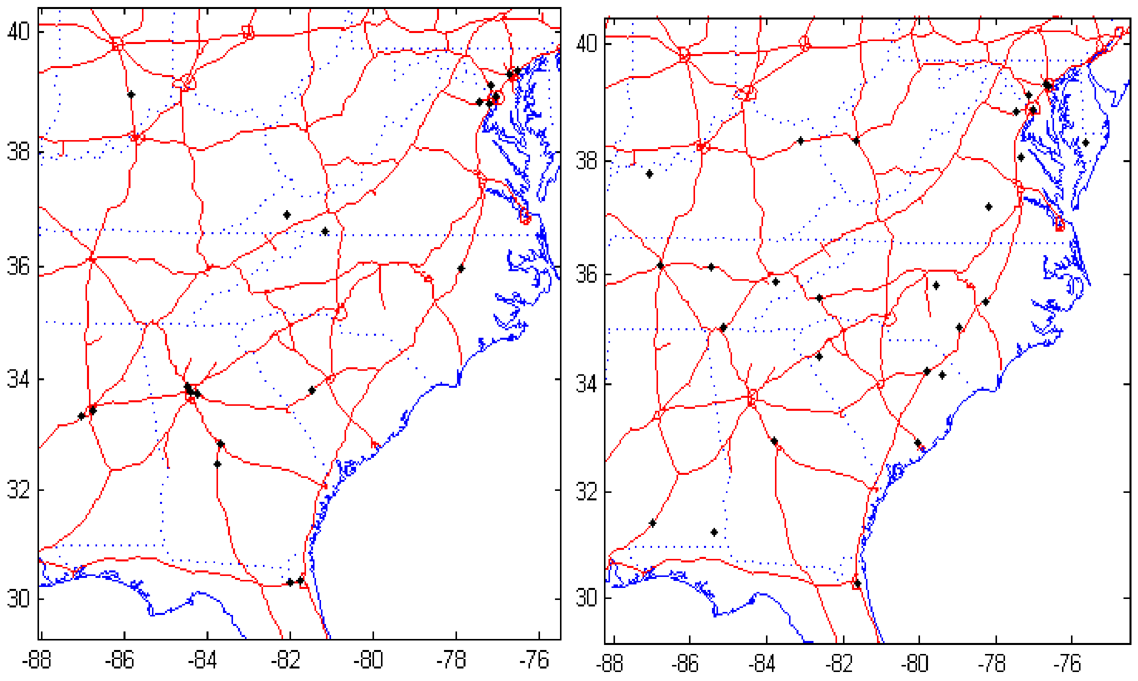


Figure 5.18. Comparison between Two PLN after Imposing 50 Miles Distance Constraint on DC Location (Left Side Graph does not have this Constraint. Right Side Graph has this Constraint)

Figure 5.18 shows, after imposing the 50-mile constraint on DC locations, the DCs became much more spread out and made the travel time from one DC to its local region much shorter.

5.6 Conclusion

This chapter used truck headway and a headway ratio to estimate the average package wait-for-trucks time in a package queue. The average package travel time was used as a criterion to design the optimal PLN. A genetic algorithm was applied to search the solution space, and to determine the optimal number of DCs and the layout for that number of DCs across the contiguous 48 states of the U.S. The optimal number of DCs turned out to be 128, which results in the average package travel time of 19.8 hours. The PLN parameter elasticity and the impact on the PLN design have been investigated. The PLN layout is sensitive to some parameters, such as total package demand, truck loading and unloading time, etc. At the end, we imposed other constraints on DC location to make DCs more spread out and more accessible to local regions that are far away from where DCs cluster.

5.7 Future Work

Future work can potentially focus on the following two areas:

1. Designing network pruning techniques to improve the performance in terms of average package travel time given a set number of DCs, and the initial arc connections between those DCs.
2. From an economic analysis point of view, such as DC land costs, building costs, operating costs, etc., how do these factors influence the layout of DCs? By incorporating these factors, it becomes a multi-criteria PLN design problem. It will make a final PLN layout a trade-off among all the criteria in consideration.

6 Conclusion and Future Work

6.1 Conclusion

One of the research goals of this dissertation is to design an optimal PLN resulting in a minimum package average travel time from a package's origin to its final destination (averaged over all packages delivered over a 24-hour period). Package average travel time is intended to be used as a criterion to evaluate the performance of a PLN network. However, a closed-form solution for package average travel time in a PLN is difficult to obtain since a PLN is a priority queuing network with bulk arrivals and bulk services. An analytical model and a simulation model have been created to study PLN operations, collect statistics, and approximate package average travel time in a PLN.

An extension of Kingman's equation is applied to approximate the package average waiting time in a G/G/1 queue. The simulation model has verified that this approximation is fairly accurate when the truck utilization is high (around 80%) in the PLN. In this approximation, Kingman's equation can be represented as follows: the package average waiting time is the product of truck headway and the headway ratio. The coefficient of variation of the package interarrival time, the coefficient of variation of the truck interarrival time, and truck headway have been observed in a simulation and then fed into Kingman's equation to obtain the package average waiting time approximation. By investigating different types of PLNs, we performed a regression analysis of the results from the analytical and simulation models to conclude that the headway ratio is around 5.5 in a fairly large class of proposed PLNs for the contiguous 48 states of the U.S. By using truck headway and headway ratio, we see that the package average waiting time could be quickly and accurately approximated.

Potential locations (search space) for the DCs are found using the U.S. network of interstate and highways. The DCs ideally should be located at the highway intersections close to large population areas. A genetic algorithm is applied to search for the optimal PLN (number of DCs, their locations, and direct connections between pairs of DCs) in terms of package average travel time. The optimal PLN turns out to have 128 DCs, which results in an average package travel time of 19.8 hours. We also incorporate some other constraints to see how the optimal PLN layout changes. For example, imposing a minimum of 50 miles between any two DCs results in DCs being more spread out and more accessible for the local regions, which are far away from where DCs cluster.

A PLN simulation with advanced features, such as a package bidding scheme and agent behaviors, has been investigated as an extension to the basic simulation model. Packages bid for their trips along the way. The highest bidding packages get the highest priority among competing packages. Trucks simply take the loads with the highest bids. The bidding process and the load accepting process are controlled by agents (intelligent software).

6.2 Future Work

6.2.1 Protocols and Agents

PLN protocol design is a potential research area that would extend the work presented in this dissertation. Different network-control protocols result in different performance on the network level. From the perspective of the PLN's daily operations, what kind of criteria can be used to gauge if a PLN is operating at the optimal or near-optimal level? How should we design PLN protocols to make a PLN operate more efficiently and economically? These kinds of questions need more research.

Another possible area of future research is package and truck agent behavior in a PLN. Package and truck agents collect all information provided by DC services and utilize intelligent pricing functions and game theory practices so that packages can decide on their trip bids and trucks can select loads to transport. One natural goal for agents is to minimize cost and maximize profit. Agent decision-making pattern design, the PLN performance under different agent behaviors, etc., should be the focus of future investigations.

6.2.2 PLN Design

Future work can focus on designing network pruning techniques to improve the performance in terms of package average travel time, given a set number of DCs, and the initial arc connections between those DCs.

Another area for potential future work is to incorporate other objectives such as minimizing package travel time variance, minimizing 90th percentile of package travel time (taken over all packages delivered in a 24-hour period), minimizing total PLN logistics costs, etc., as the criteria for designing optimal PLNs. The criteria used here is a trade-off between those metrics. Use of these other metrics can avoid so-called worst-case scenarios (package average travel time is short, but some packages may have excessive travel time), and guarantee that packages can be delivered within a specific time period.

REFERENCES

1. Kay, M.G., 2004, Protocol Design for a Public Logistic Network, Progress in Material Handling Research, pp.181–188.

2. Kay, M.G. and Parlikad, A.N., 2002, Material Flow Analysis of Public Logistics Networks, Progress in Material Handling Research, pp.205–218.

3. Jain, A., 2004, Protocol Design for A Public Logistics Network, Thesis (M.S.), Dept. of Industrial and System Engineering, North Carolina State University.

4. Kay, M.G., and Jain, A., 2005, Implementing a Pricing Mechanism for Public Logistics Networks, in Proceedings of the Industrial Engineering Research Conference, Atlanta, GA.

5. Bansal, A., 2004, Design of A Public Logistics Network, Thesis (M.S.), Dept. of Industrial and System Engineering, North Carolina State University.

6. Telford, J. S., 2007, Discrete Event Simulation Library in C#, Master Thesis (Master of Integrated Manufacturing System Engineering), North Carolina State University.

7. Kingman, J. F., 1961, The Single Server Queue in Heavy Traffic. Proc. Camb. Phil. Soc, 57, pp.902–904.

8. Computer Simulation, http://en.wikipedia.org/wiki/Computer_simulation

9. Law, A. and Kelton, W, 1991, Simulation Modeling and Analysis, second ed., McGraw–Hill, New York.

10. Banks, J. and Carson, J., 1996, Discrete–Event System Simulation, Prentice–Hall, Upper Saddle River, NJ.

11. Sadoun, B., 2000, Applied System Simulation: A Review Study, Information Sciences [0020–0255] vol.124 iss.1 pp.173–192

12. Wolff, R.W., 1982, Poisson Arrivals See Time Averages, Operations Research, Vol. 30, No. 2. pp. 223-231.

13. Dijkstra's Algorithm, http://en.wikipedia.org/wiki/Dijkstra's_algorithm.

14. Golding, T., 2005, Professional .NET 2.0 generics, John Wiley, Indianapolis, IN.

15. Hopp, W., and Spearman, M., 2000, Factory Physics, McGraw–Hill/Irwin.

16. Kleinrock, L., 1975, Queueing Systems Volume 1: Theory, Wiley Interscience, New York.

17. Bailey, N. T., 1954, On Queueing Processes with Bulk Service, J. Roy. Stat. Soc. B 16, pp.80–87.

18. Downton, F., 1955, Waiting Time in Bulk Service Queues, J. Roy. Stat. Soc. B 17, pp.256–26.

19. Miller, H. C., 1959, A Contribution to the Theory of Bulk Queues, J. Roy. Stat. Soc. B 21, pp.320–337.

20. Neuts, M. F., 1967, A General Class of Bulk Queues with Poisson Input, Ann. Math. Stat. 38, pp.759–770.

21. Cohen, W., 1969, The Single Server Queue, North Holland, London.

22. Chaudhry, M. I., 1979, The Queueing System M/G/l and Its Ramifications, Naval Res. Logist. Quart. 26, pp.667–674.

23. Powell, W. B., 1966, Iterative Algorithms for Bulk Arrival, Bulk Service Queues with Poisson and Non–Poisson Arrivals, Trans. Sci.

24. Powell, W. B. and Sheffi, Y., 1983, The Load Planning Problem of Motor Carriers: Problem Description and a Proposed Solution Approach, Trans. Res. 17A, pp.471–480.

25. Powell, W. B., 1986, Approximate, Closed Form Moment Formulas for Bulk Arrival, Bulk Service Queues. Trans. Sci. 20, pp.13–23.

26. Tegiilm, L. and Lois–Teghem, J., 1969, Modeles d'attente M/G/I et GI/M/I a arrivkes et servzces en groups. Lecture Notes in Operations Research and Mathematical Economics, 8. Springer–Verlag, New York.

27. Borthakur, A., and Medhi, J. 1974, A Queueing System with Arrival and Service in Batches of Variable Size. Cakiers Cent. Etudes Rech. Opnl. 16, pp.117–126.

28. Whitt, W., 1982, On the Heavy–Traffic Limit Theorem for GI/G/co Queues, Advances in Applied Probability, vol.14, pp.171–190.

29. Whitt, W., 1984, Minimizing Delays in the GI/G/ 1 Queue, Operations Research, vol.32, pp.41–51.

30. Marchal, W. G. 1978, Some Simpler Bounds on Mean Queueing Time. Operations Research, vol.26, pp.1083–1088.

31. Chao, X., Miyazawa, M. and Pinedo, M., 1999, Queueing Networks, Customers, Signals and Product Form Solutions, John Wiley, New York.

32. Gelenbe, E. and Pujolle G., 1998, Introduction to Queueing Networks, John Wiley, New York.

33. Van Dijk, N., 1993, Queueing Networks and Product Forms, John Wiley, New York.

34. Chen, H. and Yao D., 2001, Fundamentals of Queueing Networks, Performance, Asymptotics, and Optimization, Springer-Verlag, New York.

35. UPS website. http://wwwapps.ups.com/calTimeCost?loc=en_US.

36. Beamon, B., 1998, Supply Chain Design and Analysis: Models and Methods, International Journal of Production Economics, vol. 55, no. 3, pp. 281–294.

37. Magnanti, T. and Wong, R., 1984, Network Design and Transportation Planning: Models and Algorithms, Transportation Science, vol. 18, no. 1, pp. 1–55.

38. Hanta, V., 2002, Planar Multifacility Location–The Location–Allocation Problem, Proceedings of Algoritmy, Conference on Scientific Computing, pp. 260–267.

39. Wardrop, J., 1952, Some Theoretical Aspects of Road Traffic Research. Proceedings of the Institution of Civil Engineering, Part II 1, pp. 325–362.

40. Lamar, B. and Sheffi, Y., 1987, Implicit Enumeration Method for LTL Network Design, Transportation Research Record, pp. 1–11.

41. Zhang, J., Wu Y., and Liu P., 2007, Routing Problem for Hybrid Hub–and–Spoke Transportation Network: A Case Study of a LTL Carrier, 2007 IEEE International Conference on Automation and Logistics.

42. Kay, M., Matlab Toolbox Matlog. http://www.ise.ncsu.edu/kay/matlog/index.htm.

43. Delaunay Triangulation, http://en.wikipedia.org/wiki/Delaunay_triangulation

44. Genetic Algorithm, http://en.wikipedia.org/wiki/Genetic_algorithm

45. Houck, C., Joines, J., and Kay, M., 1996, Comparison of Genetic Algorithms, Random Restart, and Two–Opt Switching For Solving Large Location–Allocation Problems, Computers and Operations Research, vol. 23, no.6, pp.587–596.

46. Houck, C., Joines, J., Kay, M., and Wilson, J., 1997, Empirical Investigation of the Benefits of Partial Lamarckianism, Evolutionary Computation.

47. Holland, J., 1975, Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, Michigan.

48. Goldberg, D. 1989, Genetic Algorithms in Search, Optimization and Machine Learning. Addison–Wesley.

49. Davis, L., 1991, The Handbook of Genetic Algorithms. Van Nostrand Reingold, New York.

50. Michalewicz, Z., 1994, Genetic Algorithms + Data Structures = Evolution Programs, AI Series, Springer–Verlag, New York.

51. Houck, C., Joines, J., and Kay, M., 1995, A Genetic Algorithm for Function Optimization: A Matlab Implementation, NCSU–IE TR 95–09.

52. Sugahara, K., 1986, Linear Programming Design of IIR Digital Phase Network, Electronics and Communications in Japan, Part 1 (Communications), vol. 69, no. 5.

53. Gersht, A., Weihmayer, R., 1986, Mixed Integer/Linear Programming Approach to Communication Network Design, Proceedings of the IEEE Conference on Decision and Control Including The Symposium on Adaptive Pro, pp. 2113–2120

54. Singhal, J., 1982, Fixed Order Branch–and–Bound Methods for Mixed Integer Programming, PhD thesis, University of Arizona, Tucson, AZ.

55. Gen, M., Kumar, A., and Kim, J., 2005, International Journal of Production Economics, vol. 98, no. 2, Nov 18, Production Research: Facing the Challenges in the New Millennium, pp. 251–261.

56. Gen, M., Cheng, R., and Oren, S., 2001, Advances in Engineering Software, vol. 32, no. 9, September, pp. 731–744.

57. Vitetta, A., 1997, Genetic Algorithms for Transportation Network Design, Advances in Intelligent Systems, pp. 267 – 272.

58. Cunha, C., and Silva, M., 2007, A Genetic Algorithm for The Problem Of Configuring A Hub–And–Spoke Network for A LTL Trucking Company in Brazil, European Journal of Operational Research, vol. 179, no. 3, Jun 16, pp. 747–758.

59. GAOT toolbox for Matlab, http://www.ise.ncsu.edu/mirage/GAToolBox/gaot/

60. Xiang, L, Kay, M.G and Telford, J, 2007, Public Logistics Network Protocol Design and Implementation, Industrial Engineering Research Conference 2007.

61. Xiang, L., Kay, M.G and Telford, J, 2008, Waiting Time Approximation in Public Logistics Network, Industrial Engineering Research Conference 2008.

62. Xiang, L., Kay, M.G and Wilson, J., Average Waiting Time Approximation for a Public Logistics Network, working paper.

APPENDICES

Appendices includes two parts, Appendix A, MATLAB code, and Appendix B, C# code.

Appendix A MATLAB Code

MATLAB code part one: package average waiting time approximation

1. WTApprox.m is function to approximate package average waiting time. It calls several other functions, such as analytical, simulation codes.

```
% Truck Ratio's problem
% SimuDCs   timeLU  ProxFac nopkg   TrCap, TrSpeed, LdFac, HWRatio,
TruckRatio
% WTApprox([15,20,25, 30, 35, 40, 45, 50, 55,60],0, 2,[250,450,
650],[5,10],60, 0.8, 0.5,[1.5,1.7,2.0]);
% WTApprox([36, 37],[0:5],2,[4000,6000],[7,10],[55, 60],[0.6,0.95],0.5,
1);
% WTApprox(36,0,2,300,10,60, 0.8, 0.5, 1);
% WTApprox([36 37],[0,5],2,300,12,60, 0.8, 0.5, 1);
% WTApprox([20,36,50,55],0,2,300,10,60, 0.8, 0.5, 1);
% WTApprox([15,20],0,2,250,[5,10],60, 0.8, 0.5, 1.5);
% WTApprox(15,0,2,250,5,60, 0.8, 0.5, 1.5)
% WTApprox([20:5:40],0,2,[250:50:450],5,60, 0.8, 0.5, [1.5,2]);

function WTApprox(varargin)
% %delete old data directory and create new one
% WRDir =
'J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\Wilson';
% cd (WRDir);
% rmdir('Data','s');
% mkdir 'Data';
cd
'J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\Wilson';
GeneralDir
='J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\Wilson\
';
% CVTrackFile = [GeneralDir 'CVfromSimulation.xls'];
DCs=varargin{1};
DCCases=length(DCs);  %get how many NoDC cases this need to run
SimuPath = 'C:\PLN\LogisticsNetwork\LogisticsNetwork\bin\Release\';
AnalyProgPath =[GeneralDir 'PLNanalytical\'];
TrCap_input = varargin{5};
%check if simulation output_title.txt exists there, if it is, delete it.
if exist([SimuPath 'output_title.txt']) ~=0
    delete ([SimuPath 'output_title.txt']);
end;

TR_in = varargin(end);
SASAnaly =cell(1,length(TR_in{1}));
for i=1:length(TR_in{1})
    SASAnaly{i}= [GeneralDir 'Data\' num2str(TR_in{1}(i)) '\'];
```

```
end

%generate parameter sets by input parameter
PSet = ParaSet(varargin{:});  %call ParaSet function to get the parameter
set
size1 = size(PSet);  %get the size of parameter set array
NoSet = size1(1,1);
NoPara = size1(1,2);
ParaSet_in = cell(NoSet,NoPara);   %initialize the cell
ParaSet_in = mat2cell(PSet, ones(1, NoSet), ones(1,NoPara));  %convert the
array to cell, which is beneficial to be taken as the

% NoSetEachDC = NoSet/DCCases;  %no of parameter if only deal with one DC
at a time(parameter sets without DC as a parameter)


for TR_run=1: length(TR_in{1}) %loop through truck ratios
    varargin(end) = {TR_in{1}(TR_run)};  %only take single Trucks ratio
parameter to run individually
    if exist(SASAnaly{TR_run}) ==0  %if that directory doesn't exist, make
that directory
        mkdir (SASAnaly{TR_run});
    end;
    %start to loop through different DC cases, each time dealing with one
DC
    for i = 1:DCCases %for each DC case;
        NoDC = DCs(i);
        AnalyNetConfPath = [GeneralDir 'PLNDesign\outputfiles\DC'
num2str(NoDC) '\'];
        %since parameter set are not keeping DC unchanged and letting
other parameters permutate, rows only for current DC should be extracted
        idx=find(cell2mat(ParaSet_in(:,1))==NoDC &
cell2mat(ParaSet_in(:,end))==TR_in{1}(1,TR_run));  %idx index which row
corresponds to current DC
        CallAnaly_input = ParaSet_in(idx,:);
        NoSetEachDC = size(CallAnaly_input,1);
        [NoArc,ArcWeight_OD]=GenePLN(NoDC, SimuPath);  %call GenePLN to
generate specific no of DC network

        %indicate the location ID in simulation output file, to be used to
grab simulation output
        NoDCStart = 1; %in simulation output file, output.txt, which row
the NoDC is
        ArcDelayPerPackageStart=14;
        ArcDelayPerPackageApproxStart = ArcDelayPerPackageStart + NoArc +
1;
        ArcPackageCVStart=ArcDelayPerPackageApproxStart+ NoArc + 1;
        ArcPackageIntArrMeanStart = ArcPackageCVStart+ NoArc + 1;
        ArcPackageIntArrVarianceStart = ArcPackageIntArrMeanStart + NoArc
+ 1;
        ArcTrucksCVStart=ArcPackageIntArrVarianceStart+ NoArc + 1;
        ArcTrucksIntArrMeanStart =ArcTrucksCVStart+ NoArc + 1;
        ArcTrucksIntArrVarianceStart = ArcTrucksIntArrMeanStart+ NoArc +
1;
        %loop through each parameter set in a single DC scenario
        for j=1:NoSetEachDC %for each parameter sets case;
            [SimuPara, HWVec,Headwayvalue,ArcDis_sort]=
CallAnaly(CallAnaly_input(j,:), AnalyNetConfPath);  %call analytical model
to generate no of Trucks needed by simulation
            ArcDis_OD = ArcDis_sort;   %ArcDis with origin and destination
            ArcDis = ArcDis_OD(:,3);  %ArcDis without origin and
destination
            ArcWeight = ArcWeight_OD(:,3); %arc weight without OD
            ArcHeadwayFromAnaly_OD = HWVec; %Headway with origin and
destination
            ArcHeadwayFromAnaly(:, j) = HWVec(:,(3:end));  %Headway
without origin and destination

            CallSimu(num2cell(SimuPara'), SimuPath);  %use num2cell to
convert input into cell, which is require for varargin in CallSimu
function
            fid=fopen([SimuPath 'output.txt']);
            in=textscan(fid, '%s %f', 'delimiter', '\t');
            fclose(fid);
            SimuTextOut = in{1};
            SimuOut = in{2};
            %             Rho = SimuOut(10);   %take loadfactor from
simulation as rho
            Rho = cell2mat(varargin(end-2)); %take loadfactor from
analytical model as rho
            TraDens = Rho/(1-Rho);
            TrCap = SimuOut(5);
            %             TrCap = 1;
            ArcDelayPerPackage(:, j) = SimuOut(ArcDelayPerPackageStart+1:
ArcDelayPerPackageStart+NoArc);
            ArcDelayPerPackageApprox(:, j) =
SimuOut(ArcDelayPerPackageApproxStart+1:
ArcDelayPerPackageApproxStart+NoArc); %each column is arc info for one
parameter set
            %             diff(:,j)=100*(ArcDelayPerPackage(:, j)-
ArcDelayPerPackageApprox(:, j))./ArcDelayPerPackage(:, j);
            %arc statistics collection
            ArcPackageCV(:, j) = SimuOut(ArcPackageCVStart+1:
ArcPackageCVStart+NoArc); %each column is arc info for one parameter set
            ArcPackageIntArrMean(:, j) =
SimuOut(ArcPackageIntArrMeanStart+1: ArcPackageIntArrMeanStart+NoArc);
%each column is arc info for one parameter set
```

```
            ArcPackageIntArrVariance(:, j) =
SimuOut(ArcPackageIntArrVarianceStart+1:
ArcPackageIntArrVarianceStart+NoArc); %each column is arc info for one
parameter set
            ArcTrucksCV(:, j) = SimuOut(ArcTrucksCVStart+1:
ArcTrucksCVStart+NoArc); %each column is arc info for one parameter set
            ArcTrucksIntArrMean(:, j) =
SimuOut(ArcTrucksIntArrMeanStart+1: ArcTrucksIntArrMeanStart+NoArc); %each
column is arc info for one parameter set
            ArcTrucksIntArrVariance(:, j) =
SimuOut(ArcTrucksIntArrVarianceStart+1:
ArcTrucksIntArrVarianceStart+NoArc); %each column is arc info for one
parameter set
            ArcRho(:, j) =min(ArcTrucksIntArrMean(:,
j)./ArcPackageIntArrMean(:, j),0.9);  %if ArcRho is higher than 0.9, set
it as 0.9. otherwise it blows up
            ArcTraDens(:, j)= ArcRho(:, j) ./ (1-ArcRho(:, j));
            ArcCVPart(:, j) = 0.5*(ArcPackageCV(:, j).^2+ArcTrucksCV(:,
j).^2);
            HandCal_SimuHW(:, j) = ArcTraDens(:, j).*ArcCVPart(:,
j).*ArcTrucksIntArrMean(:, j); %approx by headway given by simulation
            HandCal_AnalyHW(:, j) = ArcTraDens(:, j).*ArcCVPart(:,
j).*ArcHeadwayFromAnaly(:, j); %approx by headway given by analytical
model
            ArcErr_ApproxSimuHW(:, j) = (HandCal_SimuHW(:, j) -
ArcDelayPerPackage(:, j))./ArcDelayPerPackage(:, j);
            ArcErr_ApproxAnalyHW(:, j) = (HandCal_AnalyHW(:, j) -
ArcDelayPerPackage(:, j))./ArcDelayPerPackage(:, j);

            Paraset(j, :)= SimuOut(NoDCStart:ArcDelayPerPackageStart-1);
%each row is arc info for one parameter set
            %             HandCal(:,j)=TraDens * 0.5 * (ArcPackageCV(:,
j).^2+ArcTrucksCV(:, j).^2).*ArcHeadwayFromAnaly(:,j)/TrCap;
            %             HandCal_Trucksonly(:,j)=TraDens
*0.5*(ArcTrucksCV(:, j).^2).*ArcHeadwayFromAnaly(:,j)/TrCap;
            %             HandCal_pkgonly(:,j)=TraDens
*0.5*(ArcPackageCV(:, j).^2).*ArcHeadwayFromAnaly(:,j)/TrCap;

            %calculate whole network level waiting time
HandCal_WholeLevel, HandCal_Trucksonly_WholeLevel and
HandCal_pkgonly_WholeLevel
            %             PackageCV(:, j) = mean(ArcPackageCV(:, j));
            %             TrucksCV(:, j) = mean(ArcTrucksCV(:, j));
            %             Headway_WholeLeve = mean(ArcHeadwayFromAnaly);
            %             HandCal_WholeLevel = TraDens * 0.5 *
(PackageCV(:, j).^2 + TrucksCV(:, j).^2).*Headway_WholeLeve./TrCap;
            %             HandCal_Trucksonly_WholeLevel = TraDens * 0.5 *
TrucksCV(:, j).^2 .*Headway_WholeLeve./TrCap;
```

```
            %             HandCal_pkgonly_WholeLevel = TraDens * 0.5 *
PackageCV(:, j).^2 .*Headway_WholeLeve./TrCap;
            %             DelayPerPackage = mean(ArcDelayPerPackage);
        end

        ParaNames=SimuTextOut(NoDCStart:ArcDelayPerPackageStart-1)';
%each row is name for each parameter in that parameter set.It could only
obtained once, so it is not in the loop
        %get the label for each arc, its origin and destination. Label is
used in output spreedsheet to indicate what this whole row is.
        Lable_ArcDelayPerPackage=SimuTextOut(ArcDelayPerPackageStart+1:
ArcDelayPerPackageStart+NoArc);

Lable_ArcDelayPerPackageApprox=SimuTextOut(ArcDelayPerPackageApproxStart+1
: ArcDelayPerPackageApproxStart+NoArc);
        Lable_diff=SimuTextOut(ArcDelayPerPackageStart+1:
ArcDelayPerPackageStart+NoArc);
        Lable_ArcPackageCV=SimuTextOut(ArcPackageCVStart+1:
ArcPackageCVStart+NoArc);
        Lable_ArcTrucksCV=SimuTextOut(ArcTrucksCVStart+1:
ArcTrucksCVStart+NoArc);

        %simulation and analytical arc OD is different, simu start with 0,
analytical starts with 1
        a_temp = char(Lable_ArcPackageCV);
        b_temp = str2num(a_temp(:,14:end)) + 1;  %add one to origin and
destination index to match with analytical output, which starts from 1
instead of 0 (in simulation).
        c_temp = a_temp(:,1:3);  %extract 'Arc' string only
        Lable_Arcs =[c_temp num2str(b_temp)]; %combine 'Arc' and its
origin and destination
        if(isequal(b_temp(:,1),ArcDis_OD(:,1), ArcWeight_OD(:,1),
ArcHeadwayFromAnaly_OD(:,1)) & isequal(b_temp(:,2),ArcDis_OD(:,2),
ArcWeight_OD(:,2), ArcHeadwayFromAnaly_OD(:,2)))
            disp('Arcs origin and destination are in same sequence');
        end

        %dump everything for this Trucks ratio to this directory
        cd (SASAnaly{TR_run});
        %check existence of old result file, if exists, delete it
        if exist([ num2str(NoDC) 'DC.xls']) ~=0
            delete ([ num2str(NoDC) 'DC.xls']);
        end;
        warning off MATLAB:xlswrite:AddSheet;  %turn off the warning for
adding new sheet

        %tailored for ArcDelayPerPackage variable and paraset variable
        HD_ArcDelayPerPackage=cell(1,NoSetEachDC);  %HD means head title,
which indicated what this is, such as ArcDelayPerPackage
```

```
        HD_Paraset=cell(1,ArcDelayPerPackageStart-1);
        HD_Paraset = ParaNames;
        for i=1:NoSetEachDC
            HD_ArcDelayPerPackage{1,i} = ['ArcDelayPerPackageP'
num2str(i)];
        end;


        %write headers to output spreadsheet in worksheet "Alldata",
"ArcDelayPerPackage"
        xlswrite([ num2str(NoDC) 'DC.xls'], SimuTextOut, 'Alldata', 'A1');
        xlswrite([ num2str(NoDC) 'DC.xls'], SimuOut, 'Alldata', 'B1');
        xlswrite([ num2str(NoDC) 'DC.xls'], Lable_ArcDelayPerPackage,
'ArcDelayPerPackage', 'A2');
        xlswrite([ num2str(NoDC) 'DC.xls'], HD_ArcDelayPerPackage,
'ArcDelayPerPackage', 'B1');
        xlswrite([ num2str(NoDC) 'DC.xls'], ArcDelayPerPackage,
'ArcDelayPerPackage', 'B2');
        xlswrite([ num2str(NoDC) 'DC.xls'], HD_Paraset, 'Paraset');
        xlswrite([ num2str(NoDC) 'DC.xls'], Paraset, 'Paraset', 'A2');

        %call ColAverage function to get the column average for those
statistics
        [ArcDis,ArcWeight,ArcDelayPerPackage, ArcPackageCV,ArcTrucksCV,...
            ArcHeadwayFromAnaly, ArcPackageIntArrMean,
ArcPackageIntArrVariance,ArcTrucksIntArrMean,...
            ArcTrucksIntArrVariance, ArcRho, ArcTraDens, ArcCVPart,
HandCal_SimuHW,...
            ArcErr_ApproxSimuHW, HandCal_AnalyHW, ArcErr_ApproxAnalyHW] =
ColAverage(ArcDis,ArcWeight,ArcDelayPerPackage,
ArcPackageCV,ArcTrucksCV,...
            ArcHeadwayFromAnaly, ArcPackageIntArrMean,
ArcPackageIntArrVariance,ArcTrucksIntArrMean,...
            ArcTrucksIntArrVariance, ArcRho, ArcTraDens, ArcCVPart,
HandCal_SimuHW,...
            ArcErr_ApproxSimuHW, HandCal_AnalyHW, ArcErr_ApproxAnalyHW);

        %call OutputStat to dump them to spreadsheet first four parameters
must be NoDC,
        %NoDC,Lable_Arcs,ArcDis,ArcWeight
        OutputStat(NoDC,Lable_Arcs,ArcDis,ArcWeight,ArcDelayPerPackage,
ArcPackageCV,ArcTrucksCV,...
            ArcHeadwayFromAnaly, ArcPackageIntArrMean,
ArcPackageIntArrVariance,ArcTrucksIntArrMean,...
            ArcTrucksIntArrVariance, ArcRho, ArcTraDens, ArcCVPart,
HandCal_SimuHW,...
            ArcErr_ApproxSimuHW, HandCal_AnalyHW, ArcErr_ApproxAnalyHW);
        %clear those variable for next DC senario, otherwise different DC
        %has different no of arc, will incur errors
        clear Lable_Arcs ArcDis ArcWeight ArcHeadwayFromAnaly
ArcDelayPerPackage  ArcDelayPerPackageApprox ArcPackageCV ...
            ArcPackageIntArrMean ArcPackageIntArrVariance ArcTrucksCV
ArcTrucksIntArrMean...
            ArcTrucksIntArrVariance  ArcRho ArcTraDens ArcCVPart
HandCal_SimuHW HandCal_AnalyHW...
            ArcErr_ApproxSimuHW ArcErr_ApproxAnalyHW Paraset HandCal
HandCal_Trucksonly HandCal_pkgonly;
    end
end
cd (AnalyProgPath);
```

2. `OutputStat.m` to output desired variables to spreadsheet

```
function OutputStat(varargin)
NoDC = varargin{1};
ArcOD = varargin{2};
Lable_Arcs = 'ArcOD'; %label for Arc's OD
varargin = varargin(3:end);  %remove no of DC and ArcOD from input
parameter list
NoStat=length(varargin); %get no of input statistics
NoParaSet= size(varargin{3},2); %get how many parameter set this run has,
which is the number of columns for varargin(4)

for j=1: NoParaSet %different paraset is dumped to different worksheet
    for i = 1 : NoStat
     Header{i} =inputname(i+2);  %offset 2, which are the 1st and 2nd
input parameter, NoDC and Lable_Arcs
        if size(varargin{i},2) ~= NoParaSet
            Data{i}=varargin{i}; %num2cell(varargin{i},2);
        else
             Data{i}= varargin{i}(:, j);
        end
    end
    xlswrite([ num2str(NoDC) 'DC.xls'], [Lable_Arcs Header],
['OutputP',num2str(j)], 'A1');
    xlswrite([ num2str(NoDC) 'DC.xls'], num2cell(ArcOD,2),
['OutputP',num2str(j)], 'A2');
    xlswrite([ num2str(NoDC) 'DC.xls'], [Data{:}], ['OutputP',num2str(j)],
'B2');
end
```

3. ColAverage.m calculates average for a whole column and append it to the end of that column

```
%This function calculates average for a whole column and append it to the
end of that column
% input: arr
% output: arr (which has appened column averge at the bottom)
function [varargout] = ColAverage(varargin)


% varargin = varargin(3:end);  %remove no of DC and ArcOD from input
parameter list
NoStat=nargin; %get no of input statistics
NoParaSet= size(varargin{3},2); %get how many parameter set this run has,
which is the number of columns for ArcDelayPerPackage
for i = 1 : NoStat  %loop through each statistics.
    if i <3 % 1st and 2nd don't are single column
        arr(:,1)=varargin{i}(:, 1); %copy out data for current statistics
variable and for current paraset
        arr(end+1,1)= mean(arr); %appened column averge at the bottom
    else
        for j=1: NoParaSet  %loop through each parameter set's record
            No_arc = length(varargin{i}(:, j));
            arr(1:No_arc,j)=varargin{i}(:, j); %copy out data for current
statistics variable and for current paraset
            arr(No_arc+1,j)= mean(arr(1:No_arc,j)); %appened column averge
at the bottom
        end
    end
    varargout{i}=arr;
    clear arr;
end

% disp('done');
```

4. Routes_1step.m Transfer the routes_cell matrix to routes matrix

```
function [routes]=Routes_1step(routes_cell)
%Transfer the routes_cell matrix to routes matrix, which contains only the
next step in the package route to its destination.
% routes_cell: input path cell array, which is the ouput of pred2path
function
% routes: routes matrix which only contains next step in the path (one
step after one step on the path)

V_length=size(routes_cell,1);
H_length=size(routes_cell,2);
routes = ones(V_length, H_length);

for i=1:V_length
    for j=1:H_length
```

```
        if i==j
            routes(i,j) = routes_cell{i,j};
        else
            temp = routes_cell{i,j};
            routes(i,j) = temp(2);
        end
    end
end
```

5. GenePLN.m generate underlying road network

```
function [NoArc, Arcweight]=GenePLN(NoDC, SimuPath)

load data20k  %data20k has all the road network for cities with population
more than 20k. it seems be generated by UnderRoadNetworkv3.m, but not very
sure right now.
WT=c;   % weight is the population for that node. c = InitWt(XY);

% NoDC = 36; %specify how many DC graphy you need either 15, 36, 55
outdir_general =
'J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\Wilson\P
LNDesign\outputfiles\';
outdir = [outdir_general 'DC' num2str(NoDC) '\'];
centerXY = city2lonlat('Charlotte', 'NC');
Nottooclose = 50; %threshhold for minimun distance b/w any DCs
% global d0
% d0 = dijkstra(sparse(list2adj(IJD)),1:size(XY,1));  %dijkstra function
to output the shortpath distance matrix to d0

Nonode=size(XY,1);  %total number of nodes in data20k
p=0.04; %threshhold for that node to be selected as candidate DC
prox = 2; %proximity factor

% Save the current state and repeat the sequence. in order to replicate
results
% s=sum(100*clock);
% save curseed s;
load curseed;  %load saved seed to replicate results
rand('state',s);

% wtrandperm(w,2)adfadf
CandiDC=zeros(1,Nonode);
b=find(rand(1,length(WT)) < p * WT/mean(WT));  %Selection of DCs with
probability using the population around them asvthe weight.
CandiDC=idx2is(b,Nonode); %candidate DC is indexed by one
cXY = XY(CandiDC',:); %candidate DC's XY
cPop = WT(CandiDC)';
%create IJD for candidate DCs
T = delaunay(cXY(:,1),cXY(:,2));
cIJ=tri2list(T);
```

```
cIJD = zeros(length(cIJ),3);
for i=1:length(cIJ)
    cIJD(i,:) = [cIJ(i,:) 1.2*dists(cXY(abs(cIJ(i,1)),:),
cXY(abs(cIJ(i,2)),:), 'mi')];  %get distance b/w i and j to create IJD
end

%get rid of cities where it is within some threshhold (such as 25
miles)distance
id= find(cIJD(:,3) > Nottooclose);  %only pick out arc distance which is
greater than threshhold value
[sXY,sIJD,sisXY,sisIJD] = subgraph(cXY,[],cIJD,idx2is(id, length(cIJD)));
cXY = sXY;
cIJD = sIJD;
cPop = cPop(sisXY);
%make destination directory to store network configuration files
if exist(outdir)==0
    mkdir (outdir);
end;
%output XY_NoDC, IJD_NoDC, Pop_NoDC are DCs' XY, IJD and population
[XY_NoDC,IJD_NoDC,Pop_NoDC, FigHandle]=genePLNtoPlot(cXY,cIJD,cPop, NoDC,
centerXY);
saveas(FigHandle, [outdir num2str(NoDC) 'DC.emf']);  %save current DC
configuration plot
close(FigHandle);  %close current graph

NoArc = size(IJD_NoDC, 1)*2;
%get DC structure which has XY and population
[DC]=PopAndPos(XY_NoDC);
D = dists(XY_NoDC, XY_NoDC);  %distance matrix for DCs
w= Pop_NoDC / sum(Pop_NoDC);
ProxVec = proxfac(D,w,prox);  %the W matrix, which is NoDC * NoDC matrix
%generate proximity, distances, routes, arcdistances and arcweight matrix
to be output to file
Proxfile = sum(ProxVec,2);
[Distances, P1]=dijk(list2adj(IJD_NoDC));
P2 = pred2path(P1);
Routes = Routes_1step(P2);
ArcDistance = full(list2adj(IJD_NoDC));
idx=find(list2adj(IJD_NoDC)); %find the index for arc's origin and
destination
Arcweight = ProxVec(idx); %pick out the weight corresponding to arcs
[r c]=find(list2adj(IJD_NoDC)); %find the row and col index for arcs
Arcweight = [r c Arcweight];  %combine origin, destination and arcweight
into a array, has the same configuration with IJD

PLNmat = ['PLNex' num2str(NoDC)];
%output files for simulation model
%check if that directory exists
```

```
cd (outdir);
% if exist([outdir PLNmat '.mat']) ~=0
%     delete ([outdir PLNmat '.mat']);
% end;
if exist([outdir PLNmat '.mat']) ~=0
    delete ([outdir PLNmat '.mat']);
end;
if exist([outdir 'Routes.txt']) ~=0
    delete ([outdir 'Routes.txt']);
end;
if exist([outdir 'Arcweight.txt']) ~=0
    delete ([outdir 'Arcweight.txt']);
end;
if exist([outdir 'ArcDistance.txt'])~=0
    delete ([outdir 'ArcDistance.txt']);
end;
if exist([outdir 'Distances.txt']) ~=0
    delete ([outdir 'Distances.txt']);
end;
if exist([outdir 'Proximity' num2str(prox) '.txt']) ~=0
    delete ([outdir 'Proximity' num2str(prox) '.txt']);
end;

% print(FigHandle);
% delete(FigHandle);

IJD = IJD_NoDC; %because in the pln mat dataset, IJD is the name used
save (PLNmat, 'DC', 'IJD');
fid_p = fopen(['Proximity' num2str(prox) '.txt'], 'at');
fid_d = fopen('Distances.txt', 'at');
fid_a = fopen('ArcDistance.txt', 'at');
fid_r = fopen('Routes.txt', 'at');
% fid_aw = fopen('Arcweight.txt', 'at');

%output proximity file
fprintf(fid_p, '%f\n',Proxfile);
%output Distances,ArcDistance, Arcweight and Routes files
for i=1:NoDC
    fprintf(fid_d, '%4.1f',Distances(i,1));
    fprintf(fid_a, '%4.1f',ArcDistance(i,1));
    fprintf(fid_r, '%2.0f',Routes(i,1));
    for j=2:NoDC  %start from 2nd column
        fprintf(fid_d,'\t%4.1f',Distances(i,j));
        fprintf(fid_a, '\t%4.1f',ArcDistance(i,j));
        fprintf(fid_r, '\t%2.0f',Routes(i,j));
    end
    if i~=NoDC
        fprintf(fid_d, '\n');
        fprintf(fid_a, '\n');
        fprintf(fid_r, '\n');
    end
end

fclose(fid_p);
fclose(fid_d);
fclose(fid_a);
fclose(fid_r);


cd (SimuPath);  %go to simulation path to copy network configuration file
to it. same name file will be overwritten without warning
copyfile([outdir 'Proximity' num2str(prox) '.txt'],SimuPath);
copyfile([outdir 'Distances.txt'], SimuPath);
copyfile([outdir 'ArcDistance.txt'], SimuPath);
copyfile([outdir 'Routes.txt'], SimuPath);
```

6. genePLNtoPlot.m plot generated PLN

```
 function [XY_NoDC,IJD_NoDC,Pop_NoDC, FigHandle] =
genePLNtoPlot(XY,IJD,Pop,NoDC,centerXY)

% AverageTravelTime gives the average transport time on a network.
%
% [XY_NoDC,IJD_NoDC,Pop_NoDC] = AverageTravelTime(DC_idx,tXY,tIJD,WT,d0)
%   DC_idx  = Binary representation of Selected nodes(DC's), among which
%   tXY  = XY cordinates of all input nodes (DCs).
%   tIJD = Arc list of the input network.
%   WT   = Population assigned to each node (DC in the network).
%   d0   = Shortest distance matrix between all the nodes in the network
(DC's).
% xIJD = added arc IJD
% XY = final DC candidate lon and lat
% IJD = final DC candidates IJD
% IJ = IJD after Delaunay triangulation processing
% rIJD = removed arc IJD
% IJD_Oldidx = DC candidates IJD with input original index
% IJD_Newidx = DC candidates IJD with new reduced network index
% IJDplot = road network IJD???
% XYplot = road network lon and lat ???
% h1=figure handle

Dis2center = dists(XY, centerXY);
[Dis2center_sort, I] = sort(Dis2center);  %sort by their distance from
center city(centerXY)
cXY = XY(I(1:NoDC)',:); %candidate DC's XY
cPop = Pop(I(1:NoDC))';
%create IJD for candidate DCs
T = delaunay(cXY(:,1),cXY(:,2));
```

```
cIJ=tri2list(T);
cIJD = zeros(length(cIJ),3);
for i=1:length(cIJ)
    cIJD(i,:) = [cIJ(i,:) 1.2*dists(cXY(abs(cIJ(i,1)),:),
cXY(abs(cIJ(i,2)),:), 'mi')];  %get distance b/w i and j to create IJD
end
XY_NoDC = cXY;
IJD_NoDC=cIJD;
Pop_NoDC=cPop;

makemap(XY_NoDC);
FigHandle = gcf;  %get current figure handle
pplot(IJD_NoDC,XY_NoDC,'g-');
pplot(XY_NoDC,'NumNode'); %number DCs
```

7. PopAndPos.m Return a structure with Position, Population and distance

```
 function [DC]=PopAndPos(nXY)

% PopAndPop Return a structure with Position, Population and average
% weighted distance of a node(XY cordinates of a DC).
%
%   DC = PopAndPos(nXY)
%   nXY =   XY cordinates of nodes(DC's)
%   DC  =   Structure with 3 fields(XY, Pop, Dist)
%       DC.XY   =   XY cordinates of nodes(DC's).
%       DC.Pop  =   Population assigned to each node (DC).
%       DC.Dist =   Avg. weighted distance between a node (DC) and all
%                   USZIP5 locations covered by it.

% Extracting all XY cordinates and Population from US 5-digit ZIP code
data
% excluding Hawai, Alaska and Puerto Rico.
[zipXY,zipPop]=uszip5('XY','Pop',~strcmp('HI',uszip5('ST'))&~strcmp('AK',u
szip5('ST'))&~strcmp('PR',uszip5('ST')));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
% Find which uszip5 location is covered or assigned to which node(DC)
based
% on the proximity of a uszip5 location to a node.

% For approximate and fast computation.
T = delaunay(nXY(:,1),nXY(:,2));  % Delaunay triangulation of input nodes
n = prod(size(nXY(:,1)));   % No. of nodes
S = sparse(T(:,[1 1 2 2 3 3]),T(:,[2 3 1 3 1 2]),1,n,n);
idx = dsearch(nXY(:,1),nXY(:,2),T,zipXY(:,1),zipXY(:,2)); % Indexes of
nodes to which uszip5 is assigned.
```

```
% For precise and time consuming computation.
% idx=argmin(dists(nXY,zipXY,'mi'));    % Indexes of nodes to which uszip5
is assigned.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

for i=1:size(nXY,1)
    indexes=find(idx==i);   % All USZIP5 locations covered by a node.
    if ~isempty(indexes)
        WT(i)=sum(zipPop(indexes));     % Total population covered by a
node(DC).
        sumidx2=dists(zipXY(indexes,:),nXY(i,:),'mi').*zipPop(indexes); %
Vector of Distance X Population
        Dist(i)=sum(sumidx2)/WT(i); % Avg. weighted distance between a
node(DC) and all USZIP5 locations covered by it.
    else
        WT(i)=0;    % If no USZIP5 locations are covered by a node then
its 0.
        Dist(i)=Inf;    % If no USZIP5 locations are covered by a node
then its Inf.
    end
end

DC.XY=nXY;
DC.Pop=WT;
DC.Dist=Dist;
```

8. Varnames,m Converts variables to cell array

```
function [name,val] = varnames(varargin)
%VARNAMES Convert variables to cell array of variable names and vector of
values.
% [name,val] = varnames(varargin)

for i = 1:length(varargin)
   name{i} = inputname(i);
   val{i} = varargin{i};
end

name = name(:);

if all(cellfun('length',val) == 1)
   val = [val{:}]';
end
```

9. avgtranstime.m calculate average transport time

```
function [t,Tt,Tlu,Td,P,Ad,Aw,MHeadway] =
avgtranstime(At,W,tLU,noloads,Hr,HWRatio)
%AVGTRANSTIME Average transport time = travel + L/U + wait for truck time.
```

```
[P,Aw] = shorttravtime(W,At,tLU);
[Ad, MHeadway] = arcwaittime(Aw,noloads,Hr, HWRatio);
[Tt,Tlu,Td] = transporttime(At,tLU,Ad,P);
t = sum(sum(W.*(Tt+Tlu+Td)));
Td = sum(sum(W.*Td));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [P,Aw] = shorttravtime(W,At,tLU)
%Shortest travel time.

[T,P] = dijk(At);
P = pred2path(P);
Aw = sparse(length(W),length(W),0); %Aw = all paths total weight adj
matrix
for i = 1:length(Aw)
   for j = 1:length(Aw)
      if i > j, P{i,j} = fliplr(P{j,i}); end  % To make symmetric
      p = P{i,j};  % When i == j, 0 added to Aw
      Aw = Aw + sparse(p(1:end-1),p(2:end),W(i,j),length(Aw),length(Aw));
   end
end
Aw = triu(Aw) + triu(Aw,1)';  % To make symmetric


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Ad, MHeadway] = arcwaittime(Aw,noloads,Hr,HWRatio)
%Wait for truck delay along arc adj matrix.
% HWRatio = 1;
Ad = Aw;
Ad(Ad~=0) = Hr./(Aw(Aw~=0) * noloads);
MHeadway = Ad;  %headway matrix
Ad = Ad * HWRatio;  %Delay is a function of headway. It is a sigle value
number


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Tt,Tlu,Td] = transporttime(At,tLU,Ad,P)
%Total wait for truck time along path from DC i to DC j.

Tt = zeros(size(At)); Td = Tt;
Tlu = (cellfun('size',P,2) - 1) * tLU;
for i = 1:length(Tt)
   for j = 1:length(Tt)
      p = P{i,j};  % When i == j, 0 added to Td(i,j)
      Tt(i,j) = sum(diag(At(p(1:end-1),p(2:end))));
      Td(i,j) = sum(diag(Ad(p(1:end-1),p(2:end))));
   end
end
```

10. CallAnaly.m call analytical model with all parameter sets

```
%Function: call analytical model with all parameter sets
% timeLU: loading and unloading time, default 5 minutes, other values
could be [10,15]
% ProxFac: Proximity factor, default 0, [2,6,]
% nopkg: no of packages in this model. default is 3000 [5000, 8000]
% TrCap: truck capacity. default 5, [10]. Truck capacity above 10 is a
rare case
% TrSpeed: truck traveling speed, default 60, [55, 60, 70]
% LdFac: load factor for trucks, 0.8, [0.6, 0.8, 0.95]
% HWRatio: Headway ration, which is used to determine the waiting time for
packages, default 0.5, [0.5: 0.1: 1]
% one simple example of use would be: CallAnaly(36,5,2,3000,10,60, 0.8,
0.5)
% another example of use would be: CallAnaly(36,5,2,3000,[5, 10],60, 0.8,
0.5)
% Run a full set of parameters:
CallAnaly(36,[0:5:10],2,[4000,6000,8000],[7,10,12],[55, 60,70],[0.6, 0.8,
0.95], 0.5)
% fh = @(a) sum((ad - a*hw).^2); a = fminsearch(fh,.2); fh = @(ab) sum((ad
- ab(1) - ab(2)*hw).^2); ab = fminsearch(fh,[0 0])
% CallAnaly20DC(10,2,700,12,70, 0.6, 0.5)

function [SimuPara,HWVec,Headwayvalue,Arcdis_sort]=CallAnaly(varargin,
AnalyNetConfPath)
%[names,val] = varnames(varargin);
format long g;
% AnalyNetConfPath = char(varargin(end));  %extract AnalyNetConfPath from
input arguement
% varargin(end) = '';
NoDC = varargin{1}; %change
% outputdir =
['J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\PLNanal
ytical' num2str(NoDC) 'DC\'];
% outputdir =
'J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\PLNanaly
tical';

PSet = ParaSet(varargin{:});  %call ParaSet function to get the parameter
set
size1 = size(PSet);  %get the size of parameter set array
NoSet = size1(1,1);
NoPara = size1(1,2);

ParaSet_in = cell(NoSet,NoPara);   %initialize the cell
ParaSet_in = mat2cell(PSet, ones(1, NoSet), ones(1,NoPara));  %convert the
array to cell, which is beneficial to be taken as the
%input parameter in "maketranstime" function)
```

```
model=['This is ' num2str(NoDC) 'DC from Analytical model:'];
disp(model);

% if exist([outputdir 'Mheadway.xls']) ~=0
%     delete ([outputdir 'Mheadway.xls']);
% end;
% if exist([outputdir 'results.xls']) ~=0
%     delete ([outputdir 'results.xls']);
% end;
% if exist([outputdir 'arcdis.xls']) ~=0
%     delete ([outputdir 'arcdis.xls']);
% end;
for i = 1 : NoSet
[NoDC, names,val,MHeadway,Arcdis_sort]=maketranstime(ParaSet_in{i,:},
AnalyNetConfPath);  %call maketranstime to get statistics for analytical
model %change
SimuPara(:, i) = val(1:8); %take the first 8 parameter as input parameter
for simulation model. Each col of it is a paraset for simulation. The last
one is TruckRatio
SimuPara(end-1, i)=ceil(SimuPara(end-1, i)*SimuPara(end, i)); %adjust no
of trucks to bee feed into simulation
[r c]=find(MHeadway);  %find that headway's orgin and destination
HWVec(:,1)=r;  %HWVec's first col is that headway's origin
HWVec(:,2)=c; %HWVec's second col is that headway's destination
idx = find(MHeadway); %find index for that headway value corresponding
that orgin and destination pair
HWVec(:,i+2) = MHeadway(idx); % append headway vector to that array, whose
first and second col is origin and destination
% HWVec(:,i)=MHeadway(MHeadway~=0); %all headway for one para set should
be formatted as a vector
Headwayvalue = val(11);
end;
HWVec=full(HWVec);  %make it as a normal matrix, each col contains headway
for one paraset.
%arcdis_sort is the arc distance in one column, sorted by destination DC.
%It will be used in excel file: Para4SimuTemplate20DC.xls
% fid = fopen([outputdir 'arcdis.xls'],'a');
% fprintf(fid,'%s\n','Arcdis');
% fprintf(fid,'\n');
% fprintf(fid,'\n');
% fprintf(fid,'%2.0f\t %2.0f\t %4.1f\t\n',Arcdis_sort');
% fclose(fid);
```

11. CallSimu.m call .NET PLN simulation program

```
 %script to call .NET PLN simulation program
%Input parameter: LoadingUnloading Time, Proximity factor, no of packages,
truck capacity, truckspeed
```

```
% timeLU: loading and unloading time, default 5 minutes, other values
could be [10,15]
% ProxFac: Proximity factor, default 0, [2,6,]
% nopkg: no of packages in this model. default is 3000 [5000, 8000]
% TrCap: truck capacity. default 5, [10]. Truck capacity above 10 is a
rare case
% TrSpeed: truck travelling speed, default 60, [55, 60, 70]
% trucks: no of trucks in simulation
% one example of calling this function is: CallSimu([5:5:15], [0,2,6],
[3500:500:4500], [30:5:40],[55:5:65])
% another example with just one set parameter: CallSimu(5, 0, 3000,5,
60,1910)
% Output is written in the file named: output.txt
% Run a full set of parameters:
% CallSimu([0:5:10],0,[3000,5000,8000],[5,10],[55, 60,70],???) refer to
J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\Result\Si
mu VS Analy\Don'tDelete_Para4SimuTemplate.xls for determining how many
trucks needed in each parameter set.

function CallSimu(varargin, SimuPath)
% NoDC = 20; %change
% outputdir = ['C:\PLN\LogisticsNetwork' num2str(NoDC)
'DC\LogisticsNetwork\bin\Release\'];
outputdir = SimuPath;
PSet = ParaSet(varargin{:});  %call ParaSet function to get the parameter
set
size1 = size(PSet);  %get the size of parameter set array
NoSet = size1(1,1);
NoPara = size1(1,2);

ParaSet_in = cell(NoSet,NoPara);   %initialize the cell
ParaSet_in = mat2cell(PSet, ones(1, NoSet), ones(1,NoPara));  %convert the
array to cell, which is beneficial to be taken as the
%input parameter in "maketranstime" function)
% output = [outputdir 'output.xls'];
output_title = [outputdir 'output_title.txt'];
cd (outputdir);   %switch to the directory where the simulation program
resides

%check if output.txt is existing there, if it is, delete it. It MUST be
deleted, since each loop dealing with only one paraset
if exist('output.txt') ~=0
    delete 'output.txt';
end;
% if exist('output_title.txt') ~=0  %shouldn't delete, since it is on a
% loop, it will deleted everytime, and result in nothing at the end.
%     delete 'output_title.txt';
% end;
```

```
%output No. of input parameters, No. of parameter sets, and input
parameter value for each parameter
fid = fopen(output_title,'at');
fprintf(fid,'This experiment has %2.0f parameters, totally %8.0f parameter
sets. Input parameters are: \n', NoPara, NoSet);
for i =1:length(varargin);
    InputPara= varargin{i};
fprintf(fid,'%8.1f',InputPara);
end;
fprintf(fid,'\n');
fclose(fid);


for i = 1 : NoSet
    for j=1 : NoPara
        p(j) = ParaSet_in{i,j};
    end;
timeclock = datestr(now, 'mmmm dd, yyyy HH:MM:SS AM'); %get current time
fid = fopen(output_title,'at');
fprintf(fid,'\nNo.%4.0f run, at timeclock %s\n parameters are:\n', i,
timeclock);
for m =1:length(p);
    CurrentPara= p(m);
fprintf(fid,'%8.1f',CurrentPara);
end;
fprintf(fid,'\n');
fclose(fid);
%identify which run this is for simulation model. just for comparison
reason
% fid = fopen(output, 'at');
% fprintf(fid, 'Run No: %4.0f\n', i);
% fclose(fid);
%call simulation program
eval(['!LogisticsNetwork.exe ', num2str(p)]);
% eval(['!LogisticsNetwork.exe ', num2str(5), space, num2str(0), space,
num2str(18880), space, num2str(40), space,num2str(60)]);
% eval(['!Network.exe ', num2str(Rep), space, num2str(TruckCapacity(i))])
end

% [num text]=xlsread([SimuPath 'output.xls']);
% size(num);
```

12. Maketranstime.m to compute transit time

```
% Trans Time Script
% Scenario: US SE - Half Headway?
%Calculate the average transit time for packages using half headway as the
%packages average waiting time for incoming trucks
```

```
function [NoDC, names,val,MHeadway,Arcdis_sort]=maketranstime(NoDC,
timeLU, ProxFac, nopkg, TrCap,TrSpeed,LdFac, HWRatio,TruckRatio,
AnalyNetConfPath) %change
% timeLU: loading and unloading time, default 5 minutes, other values
could be [10,15]
% ProxFac: Proximity factor, default 0, [2,6, 10]
% nopkg: total packages in this analytical model, default is 450000
[450000, 500000 ]
% TrCap: truck capacity. default 1000, [1500, 2000]
% TrSpeed: truck travelling speed, default 60, [55, 65, 70]
% TruckRatio: the ratio for no of trucks b/w analytical model and
simulation
% One example execution is: maketranstime(5, 0, 450000, 500,60)
% (Attention! LdFac(load factor) has been hardcoded as 0.8, which is the
economic operation for trucks in general

% close all
% cd
(['J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\AnalyticalModel\PLNDes
ign\outputfiles\DC' num2str(NoDC)]);
cd (AnalyNetConfPath);
load (['plnex' num2str(NoDC)]); %change
% TrSpeed = 60;
% LdFac = 1;  %100% loadfactor is just for calculating the lower bound for
# of trucks
Hr = 24;
% timeLU = 10;
% ProxFac = 0;
% nopkg = 1888000; % Total Pkg per Day
% TrCap = 240;
Adis = list2adj(IJD); %Arc distance for directly connected DCs
TdMatrix = zeros(size(Adis));
At = list2adj(IJD)/TrSpeed;
W = proxfac(dists(DC.XY,DC.XY,'mi'),DC.Pop/sum(DC.Pop),ProxFac);
avgtrload = TrCap*LdFac;
noloads = nopkg/avgtrload;
tLU = 2*timeLU/60;

[AvgTransTime,Tt,Tlu,Td,P,Ad,Aw,MHeadway] =
avgtranstime(At,W,tLU,noloads,Hr,HWRatio);
% AvgTransTime
% NoTrucks = sum(sum(At))/36/36 * noloads / Hr;
NoTrucks = sum(sum(W.*(Tt + Tlu))) * noloads / Hr;
Headway = Td / HWRatio; %single number, which is total headway for this
whole network
NoTrucks = round(NoTrucks);
LdFacA=LdFac;  %LdFacA mean this is load factor for analytical model
```

```
[names,val] =
varnames(NoDC,timeLU,ProxFac,nopkg,TrCap,TrSpeed,NoTrucks,TruckRatio,LdFac
A,HWRatio,Headway,AvgTransTime);
% [names,val] = varnames(timeLU,ProxFac,nopkg,TrCap,TrSpeed, NoTrucks);
%match up output format from simulation
%maketranstime(timeLU, ProxFac, LdFacA, Hr, PrcntgUPS, TrCap,TrSpeed)
%[names1, val1] =
varnames(TotalPkg,timeLU,ProxFac,AvgTransTime,TrCap,NoTrucks);
% [names,val] = varnames(LdFacA, NoTrucks ,HWRatio,Td);
% wp=sum(W,2);

str1 = mdisp(val,names); disp(' '), disp(str1);   %display the verbose
output
%str1 = mdisp(val1,names1); disp(' '), disp(str1);   %display the simple
output
% output Mheadway matrix, since it is sparse matrix, it is hard to output
it. so just use diary

% diary([outputdir 'Mheadway.xls']);
% MHeadway
% diary;
% fid = fopen([outputdir 'results.xls'],'a');
% fprintf(fid,'%s\n',str1);
% fclose(fid);
%make and ouput IJD, which is sorted by destination node
IJD_temp = IJD;
IJD_temp(:,2)=abs(IJD(:,2));  %make 2nd column positive
IJD_temp1 = [IJD_temp(:,2), IJD_temp(:,1), IJD_temp(:,3)]; %swith first
and 2nd column
Arcdis_sort = [IJD_temp; IJD_temp1]; %stack two IJD matrix vertically
Arcdis_sort=sortrows(Arcdis_sort,2); %sort rows by the 2nd column
```

13. ParaSet.m to get the combination of parameters

```
%function to get the combination of parameters---the parameter set
%Input is some vector, delimited by comma. such as ParaSet([1:3], [5:9]);
%Output(AllParaSet) is an array, each row of that array is a parameter
set. The number of
%that array's row denotes how many combinations the parameter could
have(for
% each combination, each parameter will have a value, no missing value
allowed).

function AllParaSet=ParaSet(varargin)  %variable number of input arguments

out=varargin;  %let out has same structure with input argument cell.
notice varargin is a cell
[out{:}] = ndgrid(varargin{:});  %ndgrid function output is written into
cell out
```

```
cellout = cellfun(@(x) x(:), out, 'UniformOutput',false);  %convert array
in cell into only one column, then it could be feed into following
combination function

X = [cellout{:}] ; %combination function to generate the parameter
combination
AllParaSet = X;
```

14. Proxfac.m calculate proximity factor

```
function W = proxfac(D,w,p)
%PROXFAC Order-based proximity factor.
% W = proxfac(D,w,p)
%     D = n x n distance matrix
%     w = n-element marginal weight
%     p = proximity factor (p == 0 => no proximity adjustment)
%       = 6.668, default
%     W = n x n weight matrix, where, for p = 0, W(i,j) = w(i)*w(j)
%
% Default based on LS fit to 1997 State-to-State Commodity Flow data
% on the value of shipments (http://www.bts.gov/ntda/cfs/cfs97od.html)
% with 95% confidence interval [6.5281, 6.8079].

% Copyright (c) 1994-2003 by Michael G. Kay
% Matlog Version 7 25-Sep-2003

% Input Error Checking
****************************************************
if nargin < 3 | isempty(p), p = 6.668; end
% End (Input Error Checking)
**********************************************

m = length(D);
W0 = w(:)*w(:)';
I = argsort(argsort(D,2),2);
I = triu(I) + triu(I,1)';
R = ((1-p/m).^(I-1))/(mean((1-p/m).^((1:m)-1)));
W = R.*W0./sum(sum(R.*W0));
```

MATLAB code part two: PLN design

1. RunGA.m is main script to run GA and design PLN

```
% This script runs GA algorithm to determine optimal number of DC and
their
% location in US
%
% By supplying the intended number of DC, each run use on intended number
of
% DC, GA figure out its DC optimal location and average travel time.
% compare those average travel time for different intended number of DC,
% whichever has shortest travel time is the optimal solution.
%Usage: [Case_track_final]=RunGA([150:160]);
%[Case_track_final]=RunGA([170:190]);[Case_track_final]=RunGA([210:220]);
%timeLU=5,nopkg=1e7, Nottooclose = 25, use
[Case_track_final]=RunGA([150:155]);
%timeLU=5,nopkg=1e7, Nottooclose = 50, use
[Case_track_final]=RunGA([165:170]);
%timeLU=5,nopkg=5e7, Nottooclose = 25, use
[Case_track_final]=RunGA([205:210]);
%timeLU=10,nopkg=1e7, Nottooclose = 25, use
[Case_track_final]=RunGA([145:145])
%timeLU=10,nopkg=5e7, Nottooclose = 25, use
[Case_track_final]=RunGA([185:190]);
function [Case_track_final]=RunGA(DCSets)
%make four parameter in evelfunc(traveltime) global, so they don't have to
be passed in when GA call it to evaluate chrome.
global XY IJD Pop mainpath path_optimal path_optimal_DC TargetNoDC rep
startpoptype timeLU ProxFac nopkg TrCap TrSpeed LdFac HWRatio
idx_awayfromCentroid Nottooclose;
Starttime=clock; %record program start run time
No_rep=3;
%define parameters for PLN
% timeLU=5;  %minutes;
timeLU=10;  %minutes;
ProxFac=2;
% nopkg=1E7;
nopkg=5E7;
TrCap=500;
TrSpeed=60;
LdFac=0.8;
HWRatio=5.5;
Nottooclose = 25; %threshhold for too close DC, 25 mile.
% Nottooclose = 50;
idx_awayfromCentroid=0; %minimum distance for any DC away from a zip3 area
centroid
Idx_CombRe='N';%combine result?
startpoptype='PopWeight';
% startpoptype='Random';
```

```
NoDCSets = size(DCSets,2);
MinCityPopLim = 50000;
%based on all US city has more than 50k population
[XY,IJD]= UnderRoadNetwork(MinCityPopLim);
[NoDC,DC,IJD1]=IJDfromXY(XY);
Pop=DC.Pop;

mainpath =
'J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\PLNDesign\Ling\RandomDes
ign';
path_optimal =
['J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\PLNDesign\Ling\RandomDe
sign\Optimal' '\LU' num2str(timeLU) '\Nopkg' num2str(nopkg) '\Nottooclose'
num2str(Nottooclose)];
% path_optimal=[path_optimal '\LU' num2str(timeLU)];  %for different
TimeLU, use different directory
cd (mainpath);

NoDC_input=size(XY,1);  %total no of DC from input data20k data
caseno = 0;
Case_track_allrep =cell(0,4);
Case_track_final=zeros(0,3);
GAPopSize=30;  %population size for GA
% Save the current state and repeat the sequence. in order to replicate
results
% s=sum(100*clock);
% save curseed s;
load curseed;  %load saved seed to replicate results
rand('state',s);
for rep = 1:No_rep  %for each input DCSets, do multiple reps
    Case_track = cell(0,4); %keep track of result (caseno, NoDC,
traveltime) for each run; clear it after every rep
    fprintf(1, 'This is rep: %d\n', rep);  %display current rep
    %start to loop through each DC case, such as 190 DC case
    for j = 1:NoDCSets
        gen_startpop = 0; %for each new DCSets, this is initial
generation, so make it to be gen zero;
        caseno = caseno+1;
        TargetNoDC= DCSets(j);
        [StartPop]=GenStartPop(GAPopSize,TargetNoDC,XY,Pop);
        for i=1:GAPopSize
            fprintf(1, 'Rep %d. Generating startpop(padd travel time to
StartPop) %d ',rep,i);  %display which target DC we are currently at

[StartPop(i,:),v_travelTime(i,1)]=TravelTime(StartPop(i,:),gen_startpop);
%call travel time function to calculate travel time and append it to last
column of start population
            StartPop(i,end) =v_travelTime(i,1);    %  the last column is
the average transport time for that set (corresponding row) of DC
```

```
        end

        % Crossover Operators
        xOverFNs = 'simpleXover';
        xOverOps = [1]; % number of crossover performance.

        % Mutation Operators
        mutFNs = 'binaryMutation';
        mutOps = [1 0.005]; % number of mutation performance and
probability of mutation

        % Termination Operators
        termFN = 'maxGenTerm';
        termOps = [50]; % 50 Generations

        % Selection Function
        selectFN = 'roulette';
        selectOps = [];

        % Evaluation Function
        evalFN = 'TravelTime';
        evalOps = [ ];

        % Bounds on the variables
        bounds=[zeros(NoDC_input,1) ones(NoDC_input,1)];
        % bounds=[];
        opts=[1e-6 1 1];

        %StartPop = initializega(10,bounds,'triangle2',[],[1e-6 0]);

        [x endPop bestPop
trace]=ga(bounds,evalFN,evalOps,StartPop,opts,...

termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFNs,mutOps);

        if caseno==1  %if first iteration, then initialize the track of
optimal case
            TravelTime_optimal = -x(end); %travel time is made negative to
accomendate minimization.
            idx_optimal=x(1:end-1);
            b=find(idx_optimal);
        elseif -x(end) < TravelTime_optimal  %find a better solution,
store it
            TravelTime_optimal = -x(end);
            idx_optimal=x(1:end-1);
            b=find(idx_optimal);
        end
        Case_track(end+1,:)={caseno sum(x(1:end-1)) -x(end) b};
    end
    rows1=size(Case_track,1);  %grab how many rows it has in Case_track
array
    Case_track_allrep(end+1:end+rows1,:)=Case_track;  %append each rep's
case track to Case_track_allrep array
end
%process after get a cell array contain No_rep row, each row is info for
each rep
rows2=size(Case_track_allrep,1);
Case_track_final(end+1:end+rows2,:)=cell2mat(Case_track_allrep(:,1:3));
%convert cell into normal array
[value idx_opt]= min(Case_track_final(:,3));  %find out minimum travel
time
idx_optDC = cell2mat(Case_track_allrep(idx_opt,4));  %index for DC of
optimal solution
XY_optimal = XY(idx_optDC',:); %XY for DC of optimal solution

Case_track_final=sortrows(Case_track_final,2);  %sort rows of
Case_track_final by the 2nd column, the no of DCs
%create PLN, includes DC structure, and IJD
[NoDC_optimal,DC_optimal, IJD_optimal]=IJDfromXY(XY_optimal);
sol_opti = [idx2is(idx_optDC,NoDC_input) value];
%plot optimal PLN
makemap(DC_optimal.XY);
FigHandle = gcf;  %get current figure handle
pplot(DC_optimal.XY, 'k.'); %number DCs

%save current graph to that directory
path_optimal_DC = [path_optimal '\' num2str(NoDC_optimal) 'DC'];
if exist(path_optimal_DC) ~=0
    rmdir(path_optimal_DC,'s') ;
end;
mkdir (path_optimal_DC);
saveas(FigHandle, [path_optimal_DC '\' num2str(NoDC_optimal) 'DC.emf']);
%save current DC configuration plot
close(FigHandle);  %close current graph
%save PLN data structure with XY, IJD, Pop
cd (path_optimal_DC);
PLNmat = ['PLN' num2str(NoDC_optimal)];
save (PLNmat, 'DC_optimal', 'IJD_optimal');
DCTTimeRec = ['PLN' num2str(NoDC_optimal) 'TTimeRecord'];
save (DCTTimeRec, 'Case_track_final');
GAPara=['PLN' num2str(NoDC_optimal) 'GAPara'];
save (GAPara, 'xOverOps', 'mutOps', 'termOps','selectOps', 'evalOps',
'bounds','opts');
%plot the southeast part of this PLN
FigHandle=PlotSouthEast(DC_optimal.XY, IJD_optimal);
saveas(FigHandle, [path_optimal_DC '\' num2str(NoDC_optimal)
'DC_SE.emf']);  %save current DC configuration plot
close(FigHandle);  %close current graph
```

```
%save current console output to file
diaryfile = [path_optimal '\' 'dairy.txt'];
diary(diaryfile);
disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~');
%display input command
disp('DCSets:');
disp(DCSets);
disp('No of Reps:');
disp(No_rep);
%display runtime, and optimal solution on command prompt
Endtime=clock;  %record current time stamp as end time
Runtime=etime(Endtime, Starttime)/3600;  %convert it to hours
disp('Starttime:'); disp(datestr(Starttime, 'mmmm dd, yyyy HH:MM:SS AM'));
disp('Endtime:');disp(datestr(Endtime, 'mmmm dd, yyyy HH:MM:SS AM'));
disp('Total Runtime(Hrs):'); disp(Runtime);
disp('Optimal No of DCs:');
disp(NoDC_optimal);
disp('Average Package Travel Time (Hrs):');
disp(TravelTime_optimal);
%draw graph of relationship between NoDC and Package Average travel time
plot(Case_track_final(:,2),Case_track_final(:,3));
xlabel('Number of DCs');
ylabel('Package Average Travel Time');
title('Plot of Package Average Travel Time on Number of DCs');
FigHandle2 = gcf;  %get current figure handle
saveas(FigHandle2, [path_optimal_DC '\' num2str(NoDC_optimal)
'NoDC_TravelTimeRelationship.emf']);  %save this plot
close(FigHandle2);  %close current graph
%process results
if Idx_CombRe=='Y';
    if timeLU==5
        if nopkg==1E7 & Nottooclose==25
            Case_track_final=CombRe1(128, 117);
            if nopkg==1E7 & Nottooclose==50
                Case_track_final=CombRe3(138,131);
            end
        else if nopkg==5E7
                Case_track_final=CombRe1(195, 177);
            end
        end
%     else if timeLU==10
%              Case_track_final=CombRe2(138,129,124,116);
%         end;
    end;
end;
%call traveltime func again to display the all parameters, including
headway, etc
% in this case, gen is made as -1 to indicate this is the optimal solution
case
```

```
[whatever1, whatever2]=TravelTime(sol_opti,-1);
diary off %turn off diary
```

2. TravelTime.m calculate the average transport time on a network

```
function [sol,TravelTimeT] = TravelTime(sol,gen)
global XY IJD Pop mainpath TargetNoDC rep timeLU ProxFac nopkg TrCap
TrSpeed LdFac HWRatio idx_awayfromCentroid;
% TravelTime gives the average transport time on a network.
%
% [sol,avgTtime] = TravelTime(sol,XY,IJD,Pop,d0)
%   sol  = Binary representation of Selected nodes(DC's) with one more
%          dimension at the end, which is the average travel time.
%   XY  = XY cordinates of all nodes (DCs).
%   IJD = Arc list of the network.
%   Pop   = Population assigned to each node (DC in the network).
%   d0   = Shortest distance matrix between all the nodes in the network
(DC's).

%function [sol,avgTtime] = TravelTime(sol,options)


if gen==0  %generating startpop
    fprintf(1, 'for input NoDC: %d\n',TargetNoDC);  %display which target
DC we are currently at
elseif gen==-1  %optimal solution case
    fprintf(1, 'Rep %d. Optimal solution found with those parameters: \n',
rep);  %display which target DC we are currently at
else
    fprintf(1, 'Rep %d. This is generation %d for input NoDC:
%d\n',rep,gen, TargetNoDC);  %display which target DC we are currently at
end

[sol,NoDC,DC,nIJD]=GeneFixedPLN(sol,XY,IJD,Pop,mainpath);  %create current
PLN layout
if idx_awayfromCentroid >0  %if impose "away from centroid" constraint

[XY_zip,zipPop]=uszip3('XY','Pop',~strcmp('HI',uszip3('ST'))&~strcmp('AK',
uszip3('ST'))&~strcmp('PR',uszip3('ST')));
    if min(min(dists(DC.XY,XY_zip,'mi')))<idx_awayfromCentroid  %if any DC
are away from each other less than 100 miles, then make this solution
invalid (make travel time as inf)
        TravelTimeT = inf;
        sol(end)=inf;
        disp('too close to centroid DC found');
    end
    if ~isinf(TravelTimeT)  %only deal with valid DC layout, DC are away
from each other at least 100 miles
```

```
[NoDC,names,val,MHeadway,Arcdis_sort]=maketranstime(NoDC,DC,nIJD,timeLU,
ProxFac, nopkg, TrCap,TrSpeed,LdFac, HWRatio); %change
        TravelTimeT=-val(end);  %since this is minimization problem, and
GA only do maximization. so make value negative
    end
else

[NoDC,names,val,MHeadway,Arcdis_sort]=maketranstime(NoDC,DC,nIJD,timeLU,
ProxFac, nopkg, TrCap,TrSpeed,LdFac, HWRatio); %change
    TravelTimeT=-val(end);  %since this is minimization problem, and GA
only do maximization. so make value negative
end
if gen==-1 %optimal solution case, save PLN parameter to mat file, and
also output it on screen
    NoDC_opti = sum(sol(1:end-1));
    PLNPara_Optimal = ['PLN' num2str(NoDC_opti) 'OptimalPara'];
    save (PLNPara_Optimal, 'val', 'names');
end
```

3. PrepInitSol.m impose the distance constraints

```
function [solnew] = PrepInitSol(sol,tXY,ThreshDistance)

% PrepInitSol removes the nodes such that the distance between any two
% nodes is greater than given threshold value.
%
% [solnew] = PrepInitSol(sol,tXY,ThreshDistance)
%

still=0;

while(~still)

    x=sol;
    b=find(x);  %Find the actual indexes of the selected nodes.
    nXY=tXY(b,:);   %It stores the actual XY cordinates of the selected
nodes.
    [DC]=PopAndPos(nXY);
    dt=dists(nXY,nXY,'mi');
    [e,r]=sort(dt,2);  %sort each row

    bin=[];
    for i=1:size(nXY,1);
        if ~prod(size(find(bin==i)))>0
            if dt(i,r(i,2))<ThreshDistance;
                if (DC.Pop(i)>DC.Pop(r(i,2)))
                    bin=[bin,r(i,2)];
                else
```

```
                    bin=[bin,i];
                end
            end
        end
    end

    if ~isempty(bin)
        nodesnottokeep=setxor(b(bin),b);
        solnew=idx2is(nodesnottokeep,prod(size(sol)));
    else
        solnew=sol;
        still=1;
    end
    sol=solnew;
end

solnew=sol;

PlotSouthEast.m plot south eastern part of a PLN
function [FigHandle]=PlotSouthEast(XY, IJD)
%interstate network
[XY_I,IJD_I,isXY_I,isIJD_I] = subgraph(usrdnode('XY'),
usrdnode('NodeFIPS')~= 72 & usrdnode('NodeFIPS')~= 15 &
usrdnode('NodeFIPS')~= 02 & usrdnode('NodeFIPS')~= 88 &
usrdnode('NodeFIPS')~= 91, ...
        usrdlink('IJD'),usrdlink('Type')=='I');
%limits for southeastern part
SouthEastXLim=[-87.0970 -70.8554];
SouthEastYLim=[28.9514 39.5132];
idx_X1=SouthEastXLim(1)<= XY(:,1);
idx_X2=XY(:,1)<=SouthEastXLim(2);
idx_Y1=SouthEastYLim(1)<= XY(:,2);
idx_Y2=XY(:,2)<=SouthEastYLim(2);
idx=idx_X1&idx_X2&idx_Y1&idx_Y2;

[XY,IJD,isXY,isIJD]=subgraph(XY,idx,IJD);
NoDC=size(XY,1);
%plot southeast part of this pln
makemap(XY);
FigHandle = gcf;  %get current figure handle
title(['Plot of Southeastern Part of a PLN with ' num2str(NoDC) ' DC']);
pplot(IJD_I,XY_I,'-+r','LineWidth',0.5, 'MarkerSize',0.0001);
% pplot(IJD_I,XY_I,'r-','LineWidth',0.5,'Tag','U.S. Interstate Highway
Network');
% pplot_number(XY,'NumNode'); %number DCs
pplot(XY, '.k', 'MarkerSize',13);
```

4. IJDfromXY.m calculate XY from IJD

```
% IJDfromXY calculate XY from IJD
function [NoDC,DC, IJD]=IJDfromXY(cXY)
T = delaunay(cXY(:,1),cXY(:,2));
cIJ=tri2list(T);
cIJD = zeros(length(cIJ),3);
for i=1:length(cIJ)
    cIJD(i,:) = [cIJ(i,:) 1.2*dists(cXY(abs(cIJ(i,1)),:),
cXY(abs(cIJ(i,2)),:), 'mi')];  %get distance b/w i and j to create IJD
end
NoDC = size(cXY,1);
[DC]=PopAndPos(cXY); %create DC structure to be stored as matlab workspace
variable (PLN mat dataset)
IJD=cIJD;  %because in the pln mat dataset, IJD is the name used
```

5. GenStartPop.m generate start population to feed into GA

```
 %This function generate start population to feed into GA.
function [StartPop]=GenStartPop(GAPopSize,TargetNoDC,XY,Pop)
global startpoptype;
NoCandiNodes=size(XY,1);
StartPop=zeros(GAPopSize,NoCandiNodes);  %initialize start population.
append last column as travel time for corresponding row of DC layout at
the end of this program
if strcmp(startpoptype,'Random')    %totally randomly generate start
population
    for i=1:GAPopSize
        idx_random=randperm(NoCandiNodes); %generate random permutation,
which serves as index to pick candidate DC
        idx_candi=idx_random(1:TargetNoDC); %only take first TargetNoDC
index to pick candidate DC
        StartPop(i,:) = idx2is(idx_candi,NoCandiNodes);
    end;
end

%following generating start pop is based on population weighted random
permutation
if strcmp(startpoptype,'PopWeight')
    for i=1:GAPopSize
        idx_random=wtrandperm(Pop,TargetNoDC); %generate population
weighted random permutation
        StartPop(i,:) = idx2is(idx_random,NoCandiNodes);
    end;
end

StartPop(:,end+1)=0; %The last column is travel time corresponds to that
row of DC layout. initialize them as zero to start
```

6. UnderRoadNetwork.m generate PLN underlying road network

```
function [XY,IJD]= UnderRoadNetwork(MinCityPopLim)
%This function generate PLN underlying road network.
```

```
%it first finds the Underlying Road Network primarily composed of all US
%Interstates and the part of US highways, then find out nodes among
%those are close the u.s. cities with population(default is 50k), call
%Nodeclose2City. use delauny to generate IJ array for Nodeclose2City and
%then use dijkstra to find out which nodes are on the path from one
%Nodeclose2City node to another Nodeclose2City node. those on the path
node
%should also be included in the underlying network. Finally combine
%Nodeclose2City node and On the path node (use addconnector func) to come
%up with final PLN underlying network.

% input: threshhold for city min population, default is 50k
% ouput: XY and IJD for underlying network
global idx_awayfromCentroid Nottooclose;
if nargin ==0
    MinCityPopLim=50000;  %threshhold for city min population, default is
50k
end
% SelectionRef ='50kCity';
SelectionRef ='Zip3';
UpdateNetwork ='N';
UpdateNodeonPath='N';
GenInstUSSeperately='N'; %indicator to see if generate interstate and US
highway map separately at this code bottome part
mainpath='J:\.eos\lockers\research\ie\kay\pln.dir\PLN_Ling\PLNDesign\Ling\
UnderlyingRoadNetwork';
path_graph=[mainpath '\Graph\' SelectionRef];
% if exist(path_graph) ~=0  %check if path_graph exists
%     rmdir(path_graph,'s') ;
% end;
if exist(path_graph) ==0
mkdir (path_graph);
end;

savefile_instUS = [mainpath '\' 'workspace_InstUS.mat'];
savefile_NodeonPath = [mainpath '\' 'workspace_NodeonPath.mat'];
if UpdateNetwork=='Y' %if update network is needed. since update involves
dijkstra algorithm, which is very slow (around 5 hour to complete)
    UpdateNodeonPath = UpdateNetwork; %if update network, then need to
update node on path
    %get us road link object
    o=usrdlink;
    IJD_temp=o.IJD;  %get IJD from us road link object
    % Extracting the Interstates and US highways from the whole US
Roadways
    % Network excluding Canada, Mexico and Puerto Rico, Alaska(AK 02),
Hawaii(HI 15).
    [XY,IJDi,isXY,isIJDi] = subgraph(usrdnode('XY'),
usrdnode('NodeFIPS')~= 72 & usrdnode('NodeFIPS')~= 88 &
usrdnode('NodeFIPS')~= 91 & usrdnode('NodeFIPS')~= 15 &
usrdnode('NodeFIPS')~= 02,...
        IJD_temp,usrdlink('Type')=='U' | usrdlink('Type')=='I');
    % Thin the US highways and Interstate Road Network.
    [tIJD,idxIJD] = thin(IJDi);
    [stXY,stIJD,isstXY,isstIJD] = subgraph(XY,[],tIJD);
    %remove unconnected notes
    [stXY,stIJD,isstXY2,isstIJD2]=RemoveUnconnNodes(stXY,stIJD);
    save (savefile_instUS, 'stXY','stIJD','isstXY2','isstIJD2');
else
    load (savefile_instUS);  %if no updated needed, load pre-stored mat
files
end
    %only if update node on path is needed
if strcmp(SelectionRef,'50kCity')  %if use 50k population city as
selection reference point
    s=uscity10k;  %load u.s. city with 10k population
    k=s.Pop>MinCityPopLim;
    XY_city20k=s.XY(k,:);  %XY for u.s. city with 20k population
elseif strcmp(SelectionRef,'Zip3') %if use zip3 area as selection
reference point

[XY_city20k,zipPop]=uszip3('XY','Pop',~strcmp('HI',uszip3('ST'))&~strcmp('
AK',uszip3('ST'))&~strcmp('PR',uszip3('ST')));
end

dist_mx=dists(stXY,XY_city20k,'mi');  %distance matrix b/w highway
intersection and XY_city20k
if idx_awayfromCentroid >0  %if impose "away from centroid" constraint
    idx_temp=(dist_mx>=idx_awayfromCentroid); %idx for distance larger
than idx_awayfromCentroid
    idx_pool=find(sum(idx_temp,2)==size(idx_temp,2));  %find out highway
intersections which are at least idx_awayfromCentroid away from any DC
    stXY=stXY(idx_pool,:);  %only consider valid intersections, which are
at least idx_awayfromCentroid away from any DC
    dist_mx=dists(stXY,XY_city20k,'mi');  %distance matrix b/w highway
intersection and XY_city20k
    % dist_mx(find(~idx_temp))=inf; %make nonqualified distance becomes
inf, in that way, it will not be selected as cloesest candidate to city
end
idx=argmin(dist_mx);    % Indexes of nodes closest to each city
NodeClose2City=unique(idx);
XY_NodeClose2City=stXY(NodeClose2City,:);  % Nodes closest to the cities.

T=delaunay(XY_NodeClose2City(:,1),XY_NodeClose2City(:,2)); %Delaunay
triangulation of selected nodes.
IJ=tri2list(T);
A=list2adj(stIJD);
%plot delaunay triangulation for NodeClose2City
```

```
makemap(XY_NodeClose2City);
title('Delaunay Triangulation');
FigHandle = gcf;  %get current figure handle
pplot(IJ,XY_NodeClose2City,'r-');
pplot(XY_NodeClose2City,'k.')
saveas(FigHandle, [path_graph '\' 'DT_NodeClose2City.emf']);  %save
current DC configuration plot
close(FigHandle);  %close current graph

if UpdateNodeonPath=='Y'
    NodeonPath=[];
    p=size(IJ,1);
    for i=1:p
        i
        [d2,p2]=dijk(A,NodeClose2City(IJ(i,1)),NodeClose2City(-IJ(i,2)));
%shortest routes between the cities.
        NodeonPath=[NodeonPath p2]; % All the nodes are collected.
    end
    save (savefile_NodeonPath, 'NodeonPath');
else
    load (savefile_NodeonPath);
end
%find the nodes along the path, those should be included in the underlying
network
NodeonPath = unique(NodeonPath);
if ~isempty(NodeonPath)
    [XYfinal,IJDfinal,isXYfinal,isIJDfinal] =
subgraph(stXY,idx2is(NodeonPath,size(stXY,1)),stIJD);
else
    error('Something is wrong, NodeonPath is empty');
end

%thin the underlying network
[tIJC,idxIJC] = thin(IJDfinal);
[XY,IJD,isXY,isIJD] = subgraph(XYfinal, [],tIJC);
% get rid of cities within threshhold distance, such as 25 miles
id= find(IJD(:,3) > Nottooclose);  %only pick out arc distance which is
greater than threshhold value
[XY,IJD,isXY,isIJD] = subgraph(XY,[],IJD,idx2is(id, length(IJD)));

%plot underlying network without adding connector
makemap(XY);
title('Plot Underlying Network without Adding Connector');
FigHandle = gcf;  %get current figure handle
pplot(IJD,XY,'r-');
pplot(XY_NodeClose2City,'k.')
saveas(FigHandle, [path_graph '\' 'w.o.connector.emf']);  %save current DC
configuration plot
close(FigHandle);  %close current graph
```

```
% connect some point in XY_NodeClose2City to underlying network
[IJC11,IJC12,IJC22] = addconnector(XY_NodeClose2City,XY,IJD);
IJC12n = IJC12(IJC12(:,3)>0,:); %remove cost=0 arc
IJD=[IJC12n;IJC22];  %combine IJD
XY=[XY_NodeClose2City;XY]; %combine XY
%get final XY and IJD for underlying network
%
[XY,IJD]=subgraph(XY,[~idx2is(IJC12(IJC12(:,3)==0,1),size(XY_NodeClose2Cit
y,1))',(XY(size(XY_NodeClose2City,1)+1:end,1)>-1000)'],IJD);
[XY,IJD]=subgraph(XY,[~idx2is(IJC12(IJC12(:,3)==0,1),size(XY_NodeClose2Cit
y,1))',(XY(size(XY_NodeClose2City,1)+1:end,1)>-
1000)'],IJD(IJD(:,3)<900,:));

%plot underlying network
makemap(XY);
title('FINAL Underlying Network with Adding Connector');
FigHandle = gcf;  %get current figure handle
pplot(IJD,XY,'r-');
pplot(XY_NodeClose2City,'k.')
saveas(FigHandle, [path_graph '\' 'PLNunderlying.emf']);  %save current DC
configuration plot
close(FigHandle);  %close current graph

%following code is to show interstates and interstate plus us highway.
%DON NOT DELETE
if GenInstUSSeperately=='Y';
%     path_graph=[mainpath '\Graph'];
    [XY_I,IJD_I,isXY_I,isIJD_I] = subgraph(usrdnode('XY'),
usrdnode('NodeFIPS')~= 72 & usrdnode('NodeFIPS')~= 15 &
usrdnode('NodeFIPS')~= 02 & usrdnode('NodeFIPS')~= 88 &
usrdnode('NodeFIPS')~= 91, ...
        usrdlink('IJD'),usrdlink('Type')=='I');
    % Extracting the Interstates and US highways from the whole US
Roadways
    % Network excluding Canada, Mexico and Puerto Rico.
    [XY,IJD,isXY,isIJD] = subgraph(usrdnode('XY'), usrdnode('NodeFIPS')~=
72 & usrdnode('NodeFIPS')~= 15 & usrdnode('NodeFIPS')~= 02 &
usrdnode('NodeFIPS')~= 88 & usrdnode('NodeFIPS')~= 91, ...
        usrdlink('IJD'),usrdlink('Type')=='U' | usrdlink('Type')=='I');
    makemap(XY_I);
    FigHandle = gcf;  %get current figure handle
    pplot(IJD_I,XY_I,'-+r','LineWidth',0.5, 'MarkerSize',0.0001);
    saveas(FigHandle, [path_graph '\' 'USInterstate.emf']);  %save current
DC configuration plot
    close(FigHandle);  %close current graph

    makemap(XY);
    FigHandle = gcf;  %get current figure handle
    pplot(IJD,XY,'-+r','LineWidth',0.5, 'MarkerSize',0.0001);
    saveas(FigHandle, [path_graph '\' 'USInterstateandState.emf']);  %save
current DC configuration plot
    close(FigHandle);  %close current graph
    fprintf(1, 'total intersection in interstate and US highway network:
%d\n', size(XY,1));
end;
```

7. RemoveUnconnNodes.m Removes the nodes which are not connected to the whole network

```
function [xy,ijd,isxy,isijd]=RemoveUnconnNodes(stxy,stijd)
% Removes the nodes which are not connected to the whole network.
%   [xy,ijd] = RemoveUnconnNodes(stxy,stijd)
%       stxy  = Input nodes.
%       stijd = Input Arcs.
%       xy    = Output Nodes
%       ijd   = Output Arcs.

% DI=dijkstra(sparse(list2adj(stijd)),1:size(stxy,1));
DI=dijk(list2adj(stijd));
[I,J] = find(DI==Inf);
Rnodes=I(find(J==1));  % Any no. can be taken in place of (1). It should
be less than the the no. of nodes.
[xy,ijd,isxy,isijd] = subgraph(stxy,~idx2is(Rnodes,size(stxy,1)),stijd);
```

Appendix B Simulation (C #) Code

1. Main script to run simulation

```
using System;
using System.Collections.Generic;
using System.Text;
using LogisticsNetwork.Properties;
using System.Diagnostics;
using Simulation.Statistics;
using System.IO;

namespace LogisticsNetwork
{
    class Program
    {
        static void Main(string[] args)
        {
            Stopwatch watch = new Stopwatch();
            watch.Start();
            int reps = Settings.Default.Rep;  //hardcoded for 10
replications
            if (args.Length > 0)
            {
                //reps = int.Parse(args[0]);   //set the number of
replications we will do
                Settings.Default.NumberOfDCs = int.Parse(args[0]);
                Settings.Default.LoadUnloadTime = float.Parse(args[1]);
//set the loading and unloading time
                Settings.Default.Proximity = float.Parse(args[2]);
                Settings.Default.PackagesPerDay = float.Parse(args[3]);
                Settings.Default.TruckCapacity = int.Parse(args[4]);
                Settings.Default.TruckSpeed = float.Parse(args[5]);
                Settings.Default.Trucks = int.Parse(args[6]);
                Settings.Default.TruckRatio = float.Parse(args[7]);
                //NetworkSimulator.TruckCapacity = int.Parse(args[4]);
            }
            //create ObservationBasedStatistic to keep track of result for
each replication, and then output their mean value as final result
            //ObservationBasedStatistic _truckload = new
ObservationBasedStatistic();
            ObservationBasedStatistic packageFlowtime = new
ObservationBasedStatistic();
            ObservationBasedStatistic packageOnGotime = new
ObservationBasedStatistic();
            ObservationBasedStatistic packageWaitTime = new
ObservationBasedStatistic();
            ObservationBasedStatistic TruckUtilization = new
ObservationBasedStatistic();
```

```
            ObservationBasedStatistic PackageDelieveredRec = new
ObservationBasedStatistic();
            ObservationBasedStatistic PackageCreatedRec = new
ObservationBasedStatistic();
            ObservationBasedStatistic PackagesArrivalSCV = new
ObservationBasedStatistic();
            ObservationBasedStatistic TrucksArrivalSCV = new
ObservationBasedStatistic();
            ObservationBasedStatistic Truckload = new
ObservationBasedStatistic();
            ObservationBasedStatistic TruckIdleTime = new
ObservationBasedStatistic();
            ObservationBasedStatistic TruckIdleTimePortion = new
ObservationBasedStatistic();
            NetworkSimulator.TruckCapacity =
Settings.Default.TruckCapacity;
            double[,] _ArcDelayAvg = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            int[,] _ArcDemandAvg = new int[Settings.Default.NumberOfDCs,
Settings.Default.NumberOfDCs];
            double[,] _ArcDelayAvgHour = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcDelayApprox = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            float[,] _ArcIndex = new float[Settings.Default.NumberOfDCs,
Settings.Default.NumberOfDCs];
            double[,] _ArcPackageIntArrVariance = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcPackageIntArrMean = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcTrucksIntArrVariance = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcTrucksIntArrMean = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcPackageCV = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            double[,] _ArcTrucksCV = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
            //ArcDistance _ArcDistance = new
ArcDistance(Settings.Default.NumberOfDCs);

            for (int i = 0; i < reps; i++)  //do i replication
            {   //Settings.Default.Seed = Random.Equals  //todo indicate
uniform rand seed to be used in the line next.
                NetworkSimulator simulator = new
NetworkSimulator(Settings.Default.Seed + i);

simulator.Run(TimeSpan.FromDays(Settings.Default.RunLength));  //default
run length is 200 days, which will let the simulation get into stable
status
                //_truckload.Record(simulator.TruckLoad.Mean);
                TruckUtilization.Record(simulator.TruckUtilization);
                packageFlowtime.Record(simulator.PackageFlowtime.Mean);
                packageOnGotime.Record(simulator.PackageOnGoTime.Mean);
                packageWaitTime.Record(simulator.PackageWaitTime.Mean);
                PackageDelieveredRec.Record(simulator.PackagesDelivered);
                PackageCreatedRec.Record(simulator.PackagesCreated);

PackagesArrivalSCV.Record(simulator.PackagesArrivalSCV.CoefficientOfVariat
ion);

TrucksArrivalSCV.Record(simulator.TrucksArrivalSCV.CoefficientOfVariation)
;
                Truckload.Record(simulator.TruckLoad.Mean); //truckload
here is different from truck utilization, so we should use truck
utilization to match up with load factor from analytical model
                foreach(Truck truck in simulator.Trucks)  //for each
truck, go through the loop to record its idle time
                {
                    TruckIdleTime.Record((simulator.EventCalendar.Now -
truck.TotalTravelTime).TotalHours);
                }
                TruckIdleTimePortion.Record(TruckIdleTime.Sum /
(simulator.EventCalendar.Now.TotalHours * Settings.Default.Trucks));
                for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
                {
                    for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                    {
                    _ArcDelayAvg[k,j] += simulator.ArcDelay[k,j];
                    _ArcDelayApprox[k, j] += simulator.ArcDelayApprox[k,
j];
                    _ArcDemandAvg[k, j] += simulator.ArcDemand[k, j];
                    _ArcPackageIntArrMean[k, j] +=
simulator.ArcPackageIntArrMean[k, j];
                    _ArcPackageIntArrVariance[k, j] +=
simulator.ArcPackageIntArrVariance[k, j];
                    _ArcTrucksIntArrMean[k, j] +=
simulator.ArcTrucksIntArrMean[k, j];
                    _ArcTrucksIntArrVariance[k, j] +=
simulator.ArcTrucksIntArrVariance[k, j];
                    _ArcIndex[k,j] =
simulator.ArcDistance.GetArcDistance(k,j);  //indicate real directedly
connected arcs
                    }
                }
                // simulator.ReturnODPair();
            }
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    _ArcDelayAvgHour[k,j] = _ArcDelayAvg[k,j] / reps;
//get average value over replications
                    _ArcDelayApprox[k, j] = _ArcDelayApprox[k, j] / reps;
                    _ArcDemandAvg[k, j] = _ArcDemandAvg[k, j] / reps;
                    _ArcPackageIntArrMean[k, j] = _ArcPackageIntArrMean[k,
j] / reps;
                    _ArcPackageIntArrVariance[k, j] =
_ArcPackageIntArrVariance[k, j] / reps;
                    _ArcTrucksIntArrMean[k, j] = _ArcTrucksIntArrMean[k,
j] / reps;
                    _ArcTrucksIntArrVariance[k, j] =
_ArcTrucksIntArrVariance[k, j] / reps;
                    if (_ArcIndex[j, k] != 0)
                    { _ArcPackageCV[k, j] =
Math.Sqrt(_ArcPackageIntArrVariance[k, j]) / _ArcPackageIntArrMean[k, j];
}
                    if (_ArcIndex[j, k] != 0)
                    { _ArcTrucksCV[k, j] =
Math.Sqrt(_ArcTrucksIntArrVariance[k, j]) / _ArcTrucksIntArrMean[k, j]; }

                }
            }

            //output the results
            if (Settings.Default.OutputToFile)
            {
                Console.SetOut(new StreamWriter("output.txt", true));
            }
            Console.WriteLine("SimuDCs\t" + Settings.Default.NumberOfDCs);
            Console.WriteLine("timeLU\t" +
Settings.Default.LoadUnloadTime);
            Console.WriteLine("ProxFac\t" + Settings.Default.Proximity);
            Console.WriteLine("nopkg\t" +
Settings.Default.PackagesPerDay);   //No. of package per day
            Console.WriteLine("TrCap\t" + Settings.Default.TruckCapacity);
            Console.WriteLine("Packages Created\t" +
PackageCreatedRec.Mean);
            Console.WriteLine("Packages Delievered\t" +
PackageDelieveredRec.Mean);
            Console.WriteLine("TrSpeed\t" + Settings.Default.TruckSpeed);
            Console.WriteLine("NoTrucks\t" + Settings.Default.Trucks);
            //Console.WriteLine("Average Truck Load(Load factor)\t" +
Truckload.Mean / Settings.Default.TruckCapacity);
            //Console.WriteLine("LdFacS\t" + TruckUtilization.Mean);
            Console.WriteLine("PrntTruckIdleTime\t" +
TruckIdleTimePortion.Mean );
            Console.WriteLine("TruckRatio\t" +
Settings.Default.TruckRatio);
            Console.WriteLine("AvgTransTime\t" + packageFlowtime.Mean);
            Console.WriteLine("AvgWaitTime\t" + packageWaitTime.Mean);
            /*Console.WriteLine("ArcDelay\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                    Console.WriteLine("ArcDelay " + j + "," + k + " \t" +
_ArcDelayAvgHour[j, k]);
                    //Console.WriteLine(j + " " + k + ":" +
_ArcDelayAvgHour[j, k]);
                }
            }*/
            Console.WriteLine("ArcDelayPerPackage\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcDelayPerPackage " + j + " "
+ k + " \t" + _ArcDelayAvgHour[j, k]/_ArcDemandAvg[j,k]);
                }
            }
            Console.WriteLine("ArcDelayApprox\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcDelayApprox " + j + " " + k
+ " \t" + _ArcDelayApprox[j, k]);
                }
            }
            Console.WriteLine("ArcPackageCV\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
```

```
                        Console.WriteLine("ArcPackageCV " + j + " " + k +
" \t" + _ArcPackageCV[j, k]);
                }
            }
            Console.WriteLine("ArcPackageIntArrMean\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcPackageIntArrMean " + j + "
" + k + " \t" + _ArcPackageIntArrMean[j, k]);
                }
            }
            Console.WriteLine("ArcPackageIntArrVariance\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcPackageIntArrVariance " + j
+ " " + k + " \t" + _ArcPackageIntArrVariance[j, k]);
                }
            }

            Console.WriteLine("ArcTrucksCV\t" );
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcTrucksCV " + j + " " + k + "
\t" + _ArcTrucksCV[j, k]);
                }
            }
            Console.WriteLine("ArcTrucksIntArrMean\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcTrucksIntArrMean " + j + " "
+ k + " \t" + _ArcTrucksIntArrMean[j, k]);
                }
            }
            Console.WriteLine("ArcTrucksIntArrVariance\t");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0)   //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcTrucksIntArrVariance " + j +
" " + k + " \t" + _ArcTrucksIntArrVariance[j, k]);
                }
            }

            /*Console.WriteLine("ArcDemand \t ");
            for (int k = 0; k < Settings.Default.NumberOfDCs; k++)
            {
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                {
                    if (_ArcIndex[j, k] != 0) //only output arc value
(directly connected arc)
                        Console.WriteLine("ArcDemand " + j + "," + k + "
\t" + _ArcDemandAvg[j, k]);
                        //Console.WriteLine(j + " " + k + ":" +
_ArcDemandAvg[j, k]);
                }
            }*/
            Console.WriteLine("Packages Interarrival Time SCV\t" +
PackagesArrivalSCV.Mean);
            Console.WriteLine("Trucks Interarrival Time SCV\t" +
TrucksArrivalSCV.Mean);
            Console.WriteLine();
            Console.Out.Flush();
            Console.OpenStandardOutput();
            if (!Settings.Default.OutputToFile && args.Length == 0)  //if
output to console instead of file
            {
               //Console.WriteLine("Done in " + watch.Elapsed);  //output
computing time used
               Console.ReadLine();
            }
        }
    }
}
```

2. Class NetworkSimulator create simulator objects

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using Simulation;
using LogisticsNetwork.Properties;
using Simulation.Expressions;
using Simulation.Statistics;
using System.Collections;
using Simulation.Output;

namespace LogisticsNetwork
{
    class NetworkSimulator : Simulator
    {
        private static int _TruckCapacity;
        private DC[] _DCs;
        private List<Truck> _Trucks = new List<Truck>();
        private double[,] _ArcDelay; //total wait time for packages going 
through that arc
        private double[,] _ArcDelayApprox;
        private double _InitialValue = 0.1;  //starting value for 
_ArcPackageIntArrMean, etc. if starting value is zero, then CV=NaN;


        private int[,] _ArcDemand;  //total packages going through that 
arc
        private double[,] _ArcPackageIntArrMean;  //arc level package 
inter arrival time mean
        private double[,] _ArcPackageLastArr;  //arc level package last 
arrival time, corresponding to each arc
        private double[,] _ArcPackageIntArrVariance;  //arc level package 
inter arrival time variance
        private int[,] _ArcPackageIntArrCt;  //arc level package inter 
arrival time count (count for packages arrival)
        private double[,] _ArcLoadSizeMean;  //arc level package inter 
arrival time mean

        public double InitialValue
        {
            get { return _InitialValue; }
        }
        public double[,] ArcLoadSizeMean
        {
            get { return _ArcLoadSizeMean; }
        }

        public double[,] ArcPackageLastArr
        {
            get { return _ArcPackageLastArr; }
        }
        public double[,] ArcLoadsLastArr
        {
            get { return _ArcTrucksLastArr; }
        }

        public int[,] ArcPackageIntArrCt
        {
            get { return _ArcPackageIntArrCt; }
        }

        public double[,] ArcPackageIntArrMean
        {
            get { return _ArcPackageIntArrMean; }
        }
        public double[,] ArcPackageIntArrVariance
        {
            get { return _ArcPackageIntArrVariance; }
        }
        public double[,] ArcTrucksIntArrMean
        {
            get { return _ArcTrucksIntArrMean; }
        }
        public double[,] ArcTrucksIntArrVariance
        {
            get { return _ArcTrucksIntArrVariance; }
        }
        private ObservationBasedStatistic [,] _PackageODPairTrack = new
ObservationBasedStatistic [Settings.Default.NumberOfDCs,
Settings.Default.NumberOfDCs];
        private double[,] _ODPairTransTime = new
double[Settings.Default.NumberOfDCs, Settings.Default.NumberOfDCs];
        private double _TruckUtilization;  //utilization = sum(package
hours)/(# of trucks*simulation length)


        //private ArrayList _PackageArrayList = new ArrayList();
        private Routing _Routing;
        private ArcDistance _ArcDistance;
        private Random _Random;
        private Expression<int> _RandomDC;
        private PackageGenerator _PackageGenerator;
        //public int _PackageID;
        private int _PackagesCreated;
        private int _PackagesDelivered;
        private int _PackagesInSystem;
        private ObservationBasedStatistic _TruckLoad = new
ObservationBasedStatistic();
        private ObservationBasedStatistic _PackageFlowtime = new
ObservationBasedStatistic();
        private ObservationBasedStatistic _PackageOnGoTime = new
ObservationBasedStatistic();
```

```
        private ObservationBasedStatistic _PackageWaitTime = new
ObservationBasedStatistic();
        private ObservationBasedStatistic _PackagesArrivalSCV = new
ObservationBasedStatistic();
        private ObservationBasedStatistic _TrucksArrivalSCV = new
ObservationBasedStatistic();

        public ObservationBasedStatistic TrucksArrivalSCV
        {
            get { return _TrucksArrivalSCV; }
        }
        public ObservationBasedStatistic PackagesArrivalSCV
        {
            get { return _PackagesArrivalSCV; }
        }
        public int PackagesCreated
        {
            get { return _PackagesCreated; }
        }
        public int PackagesDelivered
        {
            get { return _PackagesDelivered; }
        }

        public double TruckUtilization
        {
            get { return _TruckUtilization; }
        }
        public ObservationBasedStatistic PackageWaitTime
        {
            get { return _PackageWaitTime; }
        }
        public ObservationBasedStatistic PackageOnGoTime
        {
            get { return _PackageOnGoTime; }
        }
        public ObservationBasedStatistic PackageFlowtime
        {
            get { return _PackageFlowtime; }
        }
        public static int TruckCapacity
        {
            get { return _TruckCapacity; }
            set { _TruckCapacity = value; }
        }
        public double[,] ODPairTransTime
        {
            get { return _ODPairTransTime; }
        }
        /*public ObservationBasedStatistic TruckLoad*/
        public ObservationBasedStatistic TruckLoad
        {
            get { return _TruckLoad; }
        }
        private TimePersistentStatistic _PackageWIP = new
TimePersistentStatistic();

        public PackageGenerator PackageGenerator
        {
            get { return _PackageGenerator; }
        }

        public Expression<int> RandomDC
        {
            get { return _RandomDC; }
        }

        public NetworkSimulator(int seed)
        {
            _Random = new Random(seed);
        }
        public DC GetDC(int index)
        {
            return _DCs[index];
        }
        public Routing Routing
        {
            get { return _Routing; }
        }
        public ArcDistance ArcDistance
        {
            get { return _ArcDistance; }
        }
        private void CreateDCs(int numberOfDCs)
        {
            _DCs = new DC[numberOfDCs];
            for (int i = 0; i < _DCs.Length; i++)
            {
                _DCs[i] = new DC(i, numberOfDCs);
                AddObject(_DCs[i]);
            }
        }
        private bool _FixedValuePackages;
        public bool FixedValuePackages
        {
            get { return _FixedValuePackages; }
        }
        private PackageGenerator CreatePackageGenerator()
        {
           PackageGenerator packageGenerator;
            Expression<TimeSpan> packageInterval =
Expression.SingleToTimeSpan(new ExponentialExpression(_Random,
Settings.Default.PackagesPerDay), TimeSpan.FromDays(1));
            Expression<float> packageValue = new
NormalSingleExpression(new Random(_Random.Next()),
Settings.Default.PackageValueMean, Settings.Default.PackageValueStdDev);
            Expression<float> packageTimeValue = new
NormalSingleExpression(new Random(_Random.Next()),
Settings.Default.PackageTimeValueMean,
Settings.Default.PackageTimeValueStdDev);

            Expression<PackageAgent> agent;
            switch (Settings.Default.Mode)
            {
                case "fixed":
                    agent = new ConstantExpression<PackageAgent>(new
FixedValuePackageAgent());
                    _FixedValuePackages = true;  //flag the package
bidding agent is "fixed"
                    break;
                case "proportional":
                    agent = new ConstantExpression<PackageAgent>(new
ValueBasedPackageAgent());
                    break;
                case "highlow":
                    agent = new BernoulliExpression<PackageAgent>(new
Random(_Random.Next()),0.5f,new HighValuePackageAgent(),new
LowValuePackageAgent());
                    break;
                default:
                    throw new InvalidOperationException("Invalid agent
mode.");
            }
            packageGenerator = new PackageGenerator(packageInterval,
packageValue, packageTimeValue, _RandomDC, _RandomDC, agent);
            AddObject(packageGenerator);
            return packageGenerator;
        }
        private Expression<int> CreateDCRandomizer(string filepath) //if
proximity ratio is 1:1 for 2 DCs, that means 50% of demand will be
generated in each DC, and among those demand, 50% of them will go to
local, another 50% will go to other DC.
        {
            FrequencyExpression<int> randomDC = new
FrequencyExpression<int>(new Random(_Random.Next()));
            using (StreamReader reader = new StreamReader(filepath))
            {
                int i = 0;
                int disasterDC = Settings.Default.DisasterDC;
                while (!reader.EndOfStream)
                {
                    string line = reader.ReadLine();
                    randomDC.AddFrequencyPair(double.Parse(line), i);
                    i++;
                }
            }
            return randomDC;
        }
        private void CreateArcDemand(int numberOfDCs)
        {
            _ArcDemand = new int[numberOfDCs, numberOfDCs];
        }
        private void CreateArcDelay(int numberOfDCs)
        {
            _ArcDelay= new double[numberOfDCs, numberOfDCs];
        }
        private void CreateArcDelayApprox(int numberOfDCs)
        {
            _ArcDelayApprox = new double[numberOfDCs, numberOfDCs];
        }

        private void CreateArcPackageIntArrMean(int numberOfDCs)
        {
            _ArcPackageIntArrMean = new double[numberOfDCs, numberOfDCs];
            for (int i = 0; i < numberOfDCs; i++)
                for (int j = 0; j < numberOfDCs; j++)
                    _ArcPackageIntArrMean[i, j] = _InitialValue;
        }
        private void CreateArcPackageIntArrVariance(int numberOfDCs)
        {
            _ArcPackageIntArrVariance = new double[numberOfDCs,
numberOfDCs];
            for (int i = 0; i < numberOfDCs; i++)
                for (int j = 0; j < numberOfDCs; j++)
                    _ArcPackageIntArrVariance[i, j] = _InitialValue;
        }
        private void CreateArcPackageLastArr(int numberOfDCs)
        {
            _ArcPackageLastArr = new double[numberOfDCs, numberOfDCs];
        }

        private void CreateArcPackageIntArrCt(int numberOfDCs)
        {
            _ArcPackageIntArrCt = new int[numberOfDCs, numberOfDCs];
        }
        private void CreateArcTrucksIntArrMean(int numberOfDCs)
        {
            _ArcTrucksIntArrMean = new double[numberOfDCs, numberOfDCs];
            for (int i = 0; i < numberOfDCs; i++)
                for (int j = 0; j < numberOfDCs; j++ )
                    _ArcTrucksIntArrMean[i, j] = _InitialValue;
        }
        private void CreateArcTrucksIntArrVariance(int numberOfDCs)
        {
            _ArcTrucksIntArrVariance = new double[numberOfDCs,
numberOfDCs];
            for (int i = 0; i < numberOfDCs; i++)
                for (int j = 0; j < numberOfDCs; j++)
                    _ArcTrucksIntArrVariance[i, j] = _InitialValue;
        }
        private void CreateArcTrucksLastArr(int numberOfDCs)
        {
            _ArcTrucksLastArr = new double[numberOfDCs, numberOfDCs];
        }

        private void CreateArcTrucksIntArrCt(int numberOfDCs)
        {
            _ArcLoadsIntArrCt = new int[numberOfDCs, numberOfDCs];
        }
        private void CreateArcLoadSizeMean(int numberOfDCs)
        {
            _ArcLoadSizeMean = new double[numberOfDCs, numberOfDCs];
            for (int i = 0; i < numberOfDCs; i++)
                for (int j = 0; j < numberOfDCs; j++)
                    _ArcLoadSizeMean[i, j] = 1;
        }

        private void LoadData()
        {
            int numberOfDCs = Settings.Default.NumberOfDCs;
            CreateDCs(numberOfDCs);
            CreateArcDelay(numberOfDCs);
            CreateArcDelayApprox(numberOfDCs);
            CreateArcDemand(numberOfDCs);
            CreateArcPackageIntArrMean(numberOfDCs);
            CreateArcPackageIntArrVariance(numberOfDCs);
            CreateArcTrucksIntArrMean(numberOfDCs);
            CreateArcTrucksIntArrVariance(numberOfDCs);
            CreateArcPackageIntArrCt(numberOfDCs);
            CreateArcTrucksIntArrCt(numberOfDCs);
            CreateArcPackageLastArr(numberOfDCs);
            CreateArcTrucksLastArr(numberOfDCs);
            CreateArcLoadSizeMean(numberOfDCs);
            _Routing = new Routing("Routes.txt", "Distances.txt",
numberOfDCs);
            _ArcDistance = new ArcDistance("Arcdistance.txt",
numberOfDCs);  //read in arcdistance matrix into object _ArcDistance
            string filepath = "Proximity" +
Settings.Default.Proximity.ToString() + ".txt";
            _RandomDC = CreateDCRandomizer(filepath);
            _PackageGenerator = CreatePackageGenerator();
            if (Settings.Default.DisasterMode)
            {
                Disaster disaster = new Disaster(_Random, _RandomDC,
TimeSpan.FromDays(Settings.Default.DisasterStart),
TimeSpan.FromDays(Settings.Default.DisasterLength));
                AddObject(disaster);
                AddObject(disaster.DisasterPackageGenerator);
            }
            CreateTrucks(Settings.Default.Trucks);
        }
        public void PackageEnterSystem(Package package)
        {
            if (package.StartDC != package.EndDC)
            {
                _PackagesCreated++;  //only track non-local packages
                package.LastVisitedDC = package.StartDC;  //signal this
package last visited DC is its start DC
                _PackagesInSystem++;
                _PackageWIP.Record(_PackagesInSystem, EventCalendar.Now);

                DC dc = GetDC(package.StartDC);
                package.LastVisitedDC = package.StartDC;
                dc.ReceivePackage(package);
                int destination = Routing.GetNextDC(dc.Number,
package.EndDC);
            }
        }
        public void PackageLeaveSystem(Package package)
        {
            _PackagesDelivered++;
            _PackagesInSystem--;
            _PackageOnGoTime.Record(package.TotalTravelTime.TotalHours);
//Todo this only conside the package delivered, should consider not
delivered package as well
            _PackageFlowtime.Record((EventCalendar.Now -
package.CreateTime).TotalHours);
            _PackageWaitTime.Record((EventCalendar.Now -
package.CreateTime - package.TotalTravelTime).TotalHours);
            _PackageWIP.Record(_PackagesInSystem, EventCalendar.Now);
            //write the package ID, startDC, endDC, and transit time to
arraylist _PackageAL
```

```
            if (package.StartDC != package.EndDC)   //only take into
account non-local packages
            {
                _PackageODPairTrack[package.StartDC,
package.EndDC].Record((EventCalendar.Now -
package.CreateTime).TotalHours);
                //_PackageArrayList.Add(package.ID);
                /*_PackageArrayList.Add(package.StartDC);
                _PackageArrayList.Add(package.EndDC);
                _PackageArrayList.Add(EventCalendar.Now -
package.CreateTime);
                _PackageArrayList.Add("\r\n");*/
            }
        }
        /*static void PrintArray( double[] inputArray )
        {

         foreach ( double element in inputArray )
           Console.Write( element + " " );

         Console.WriteLine( "\n" );
        } // end method PrintArray  */
        public double[,] ReturnODPair()
        {
            return _ODPairTransTime;
        }

        protected override void Terminate()
        {
            for (int i = 0; i < Settings.Default.NumberOfDCs; i++)
                for (int j = 0; j < Settings.Default.NumberOfDCs; j++)
                    _ODPairTransTime[i, j] = _PackageODPairTrack[i,
j].Mean;
            _TruckUtilization = PackageOnGoTime.Sum /
(Settings.Default.TruckCapacity* Settings.Default.Trucks *
Settings.Default.RunLength * 24);

            /*if (Settings.Default.OutputToFile)
            {
                Console.SetOut(new StreamWriter("output.xls", true));
            }
            int areaCount = 0;
            foreach (DC dc in _DCs)
            {
                areaCount += dc.NumberOfLoadingAreas;
            }*/

            //output parameter set and performance indicator
            /*Console.WriteLine("Loading and unloading time: " +
Settings.Default.LoadUnloadTime);
            Console.WriteLine("Proximity factor: " +
Settings.Default.Proximity);
            Console.WriteLine("Average Truck Load(Load factor): " +
_TruckLoad.Mean / Settings.Default.TruckCapacity);
            Console.WriteLine("Packages per day: " +
Settings.Default.PackagesPerDay);
            Console.WriteLine("Simulation length(days): " +
Settings.Default.RunLength);
            Console.WriteLine("Total Package generated: " +
_PackagesCreated);
            Console.WriteLine("Truck capacity: " +
Settings.Default.TruckCapacity);
            Console.WriteLine("Truck speed: " +
Settings.Default.TruckSpeed);
            Console.WriteLine("***************Following are the
performance indicator:**********");
            Console.WriteLine("Number of trucks: " +
Settings.Default.Trucks);*/
            //Console.WriteLine(_PackageFlowtime.Mean.ToString("F"));

            /*for (int i = 0; i < _ODPairTransTime.GetLength(0); i++)
            {
                for (int j = 0; j < _ODPairTransTime.GetLength(1); j++)
                {
                    Console.Write(" {0}", _ODPairTransTime[i,
j].ToString("F"));

                }
                Console.WriteLine();
            }*/



            /*//verification of Little's Law
            //Console.WriteLine("verification of simulation model by
Little's Law");
            //Console.WriteLine("Loading Areas: " + areaCount);
            //Console.WriteLine("Packages Created: " + _PackagesCreated);
            //Console.WriteLine("Packages Delivered: " +
_PackagesDelivered);
            //Console.WriteLine("Package Arrival Rate (per hour): " +
_PackagesCreated / EventCalendar.Now.TotalHours);
            Console.WriteLine("Average Truck Load(Load factor): " +
_TruckLoad.Mean / Settings.Default.TruckCapacity);
            //Console.WriteLine("Real Average Truck Load(Load factor): " +
_TruckLoadFactor.Mean);
```

```
            Console.WriteLine("Package Mean Flowtime (hours): " +
_PackageFlowtime.Mean);
            //Console.WriteLine("Average # Packages in System: " +
_PackageWIP.Mean);
            double expectedWIP = _PackagesCreated /
EventCalendar.Now.TotalHours * _PackageFlowtime.Mean;
            double tolerance = 0.05;
            double percent = (1 - _PackageWIP.Mean / expectedWIP);
            if (Math.Abs(percent) > tolerance) Console.ForegroundColor =
ConsoleColor.Red;
            Console.WriteLine("Percent difference from Little's Law: " +
string.Format("{0:0.##%}", percent));
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.WriteLine();*/

            //foreach (DC dc in _DCs)
            //{
            //    foreach (LoadingArea area in dc.LoadingAreas)
            //    {
            //        foreach (TimeValuePair sample in
area.QueueLength.Data)
            //        {
            //            Console.WriteLine("{0} {1}",
sample.Time.TotalHours, sample.Value);
            //        }
            //    }
            //}

            //Console.Out.Flush();
            //Console.OpenStandardOutput();
            base.Terminate();
        }
        protected override void Initialize()
        {
            base.Initialize();  //override initializing for truck, DC,
packages. Set a break point here and step through you will go right into
this simulation
        }
    }

}


/*Console.WriteLine("Package Array: PackageID, OriginDC, DestinationDC,
TransTime");
    foreach ( Object obj in _PackageArrayList )
Console.Write( "   {0}", obj );*/
```

3. Class ArcDistance create ArcDistance objects

```
using System;
using System.IO;
using System.Text;

namespace LogisticsNetwork
{
    //load route file into _Routes matrix
    class ArcDistance
    {

        float [,] _ArcDistances;
        public ArcDistance(string ArcDistancefile, int numberOfDCs)
        {
            _ArcDistances = new float[numberOfDCs, numberOfDCs];
            using (StreamReader reader = new
StreamReader(ArcDistancefile))
            {
                int i = 0;
                while (!reader.EndOfStream)
                {
                    string line = reader.ReadLine();
                    string[] numbers = line.Split(new string[] { "\t" },
StringSplitOptions.None);
                    for (int j = 0; j < numbers.Length; j++)
                    {
                        _ArcDistances[i, j] = float.Parse(numbers[j]);
                    }
                    i++;
                }
            }
        }

        //getting arc distance given current DC
        public float GetArcDistance(int from, int to)
        {
            return _ArcDistances[from, to];
        }
    }
}
```

4. Class agent create general agent object

```
using System;
using System.Collections.Generic;
using System.Text;

namespace LogisticsNetwork
{
    abstract class PackageAgent
    {
        public abstract float InitialBidForPackage(Package package);
        public abstract bool BidForPackage(Package package, out float
bidAmount);
    }
}
```

5. Class DC create DC objects

```
 using System;
using System.Collections.Generic;
using System.Text;
using Simulation;
using Simulation.Statistics;

namespace LogisticsNetwork
{
    class DC : NetworkSimulationObject
    {
        private int _Number;
        private List<Truck> _BiddingTrucks = new List<Truck>();
        private int[] _LoadingAreaLookup;
        private List<LoadingArea> _LoadingAreas = new List<LoadingArea>();
        private TimeSpan _LastReceived = new TimeSpan();

        /*private ObservationBasedStatistic _DCSCV = new
ObservationBasedStatistic();

        public ObservationBasedStatistic DCSCV
        {
            get { return _DCSCV; }
        }*/


        public List<LoadingArea> LoadingAreas
        {
            get { return _LoadingAreas; }
        }

        public int Number
        {
            get { return _Number; }
        }
        public List<Truck> BiddingTrucks
        {
            get { return _BiddingTrucks; }
        }
        public int NumberOfLoadingAreas
        {
            get { return _LoadingAreas.Count; }
        }
        public DC(int number, int numberOfDCs)
        {
            _Number = number;
            _LoadingAreaLookup = new int[numberOfDCs];
            for (int i = 0; i < _LoadingAreaLookup.Length; i++)
            {
                _LoadingAreaLookup[i] = -1;
            }
        }

        public void ReceivePackage(Package package)
        {
            package.ArrivalTime = EventCalendar.Now; //signal package's
arrival time point

NetworkSimulator.PackagesArrivalSCV.Record(EventCalendar.Now.TotalHours -
_LastReceived.TotalHours);  //record inter arrival time for packages
transported by trucks
            NetworkSimulator.ArcDemand[package.LastVisitedDC,
this._Number] ++;  //sum up total packages going through that arc
            _LastReceived = EventCalendar.Now;  //flag last package
receive time to be calendar.now
            //record the arc level packages interarrival time mean and
variance to be used for package waiting time approximation with queueing
theory
            if (t > 1)
            {

NetworkSimulator.ArcPackageIntArrVariance[package.LastVisitedDC,
this._Number] = (1 - 1 / t) *
NetworkSimulator.ArcPackageIntArrVariance[package.LastVisitedDC,
this._Number] + Math.Pow(period -
NetworkSimulator.ArcPackageIntArrMean[package.LastVisitedDC,
this._Number], 2) / (t - 1); //recursive time variance
            }
            //NetworkSimulator.PackageOnGoTime.Record((EventCalendar.Now -
package.CreateTime).TotalHours);
            NetworkSimulator.ArcPackageLastArr[package.LastVisitedDC,
this._Number] = EventCalendar.Now.TotalHours;  //update last arrival time
for this arc

        }

        public LoadingArea GetLoadingArea(int destinationDC)
        {
            if (area == null)
            {
                area = new LoadingArea(this,
NetworkSimulator.GetDC(destinationDC));
```

```
                _LoadingAreaLookup[destinationDC] = _LoadingAreas.Count;
                _LoadingAreas.Add(area);
            }
            return area;
        }
        public void Offering()
        {
            for (int i = 0; i < _LoadingAreas.Count; i++)
            {
                _LoadingAreas[i].ScheduleOffering();
            }
        }
        public void AreaOffering(LoadingArea area)
        {
            bool accepted = true;
            while (accepted)
            {
                accepted = false;
                for (int i = 0; i < _BiddingTrucks.Count; i++)
                {
                    Truck truck = _BiddingTrucks[i];
                    if (truck.Agent.Offer(truck, area))
                    {
                        accepted = true;
                        _BiddingTrucks.RemoveAt(i);
                        area.TruckAccepted(truck, EventCalendar.Now);
                        break;
                    }
                }
            }
        }
    }
}
```

6. Class LoadingArea creates LoadingArea objects

```
 using System;
using System.Collections.Generic;
using System.Text;
using Simulation;
using Simulation.Output;
using LogisticsNetwork.Properties;

namespace LogisticsNetwork
{
    class LoadingArea
    {
        private TimeDataSeries _QueueLength = new TimeDataSeries();

        public TimeDataSeries QueueLength
        {
            get { return _QueueLength; }
        }
        private bool _OfferingScheduled;
        private int _AcceptedLoads;  //how many loads have been accepted
by trucks
        private float _WatchedLoadTotalBid;
        private DC _LocationDC;
        private DC _DestinationDC;

        public DC LocationDC
        {
            get { return _LocationDC; }
        }
        public DC DestinationDC
        {
            get { return _DestinationDC; }
        }
        public int AcceptedLoads
        {
            get { return _AcceptedLoads; }
        }
        public float WatchedLoadTotalBid
        {
            get { return _WatchedLoadTotalBid; }
        }
        List<Package> _Packages = new List<Package>();
        public void Add(Package package, TimeSpan time)
        {
            int index;
            package.Bid = package.Agent.InitialBidForPackage(package);
                _Packages.Add(package);  //add current package into the
package queue
            }
            //record package queue length
            _QueueLength.Record(_Packages.Count, time);
            if (index < (_AcceptedLoads + 1) *
NetworkSimulator.TruckCapacity)
            {
                CalculateWatchedLoadTotalBid();
            }
        }
        public LoadingArea(DC locationDC, DC destinationDC)
        {
            _LocationDC = locationDC;
            _DestinationDC = destinationDC;
        }
        public void RoundOfBidding()
        {
            if (!this._LocationDC.NetworkSimulator.FixedValuePackages) //
if package bidding pattern is fixed, we don't need to sort the package
queue since they are bidding the same. just save computation time
            {
                _Packages.Sort();  //sort the package queue after package
rebid
            }
            CalculateWatchedLoadTotalBid();
        }

        private void CalculateWatchedLoadTotalBid()
        {
            float totalBid = 0;
            int start = NetworkSimulator.TruckCapacity * _AcceptedLoads;
            if (start < _Packages.Count)
            {
                int count = Math.Min(_Packages.Count - start,
NetworkSimulator.TruckCapacity);
                for (int i = start; i < start + count; i++)
                {
                    totalBid += _Packages[i].Bid;
                }
            }
            _WatchedLoadTotalBid = totalBid;
            ScheduleOffering();
        }

        public void LoadTruck(Truck truck, TimeSpan time)
        {
            int loadSize = Math.Min(NetworkSimulator.TruckCapacity,
_Packages.Count);
            for (int i = 0; i < loadSize; i++)
            {
                truck.Packages.Add(_Packages[i]);
                _Packages[i].LastVisitedDC = _LocationDC.Number; //signal
last visited DC is this DC, which is origin for current trip
            }
            _AcceptedLoads--;
            RoundOfBidding();
        }

        public void TruckAccepted(Truck truck, TimeSpan time)   //truck
accept the load
        {
            truck.ContractedArea = this;
            int start = NetworkSimulator.TruckCapacity * _AcceptedLoads;
//start point for watchload
            _AcceptedLoads++;
            if (truck.Location == _LocationDC)
            {
                LoadTruck(truck, time);
                truck.DepartFor(_DestinationDC);
            }
            //ScheduleOffering();
        }
        //public IEnumerable<Package> GetWatchedPackages()
        //{
        //    int start = NetworkSimulator.TruckCapacity * _AcceptedLoads;
        //    if (start < _Packages.Count)
        //    {
        //        int count = _Packages.Count %
NetworkSimulator.TruckCapacity;
        //        for (int i = start; i < start + count; i++)
        //        {
        //            yield return _Packages[i];
        //        }
        //    }
        //}
        private void Offering()
        {
            _OfferingScheduled = false;
            _LocationDC.AreaOffering(this);
        }

        public void ScheduleOffering()
        {
            if (!_OfferingScheduled)
            {
                _OfferingScheduled = true;
                _LocationDC.EventCalendar.Schedule(new
MethodEvent(_LocationDC.EventCalendar.Now, Offering));
            }
        }
    }
}
```

7. Class Package create Package objects

```
using System;
using System.Collections.Generic;
using System.Text;

namespace LogisticsNetwork
{
    class Package : IComparable<Package>
    {
        //public int lastDC;
        private int _ID;


        private int _StartDC;
        private int _EndDC;

        private float _Bid;
        private float _Value;  //pkg itself value. It is not the Time
value
        private float _TimeValue;  //pkg itself value. It is not the Time
value
        private float _TotalCost;  //total cost for pkg to get delievered
to destination
        private bool _DisasterPackage;

        private TimeSpan _CreateTime;
        private TimeSpan _TotalTravelTime;
        private TimeSpan _ArrivalTime;
        private int _LastVisitedDC;
        /*private TimeSpan _TotalWaitingTime;*/
        private PackageAgent _Agent;
        private Truck _AcceptedTruck;

        public int LastVisitedDC
        {
            get { return _LastVisitedDC; }
            set { _LastVisitedDC = value; }

        }
        public int ID
        {
            get { return _ID; }
            set { _ID = value; }
        }
        public Truck AcceptedTruck
        {
            get { return _AcceptedTruck; }
            set { _AcceptedTruck = value; }
        }

        public int StartDC
        {
            get { return _StartDC; }
            set { _StartDC = value; }
        }
        public int EndDC
        {
            get { return _EndDC; }
            set { _EndDC = value; }
        }

        public float Bid
        {
            get { return _Bid; }
            set { _Bid = value; }
        }
        public float Value
        {
            get { return _Value; }
        }
        public float TimeValue
        {
            get { return _TimeValue; }
        }
        public float TotalCost
        {
            get { return _TotalCost; }
            set { _TotalCost = value; }
        }

        public bool DisasterPackage
        {
            get { return _DisasterPackage; }
            set { _DisasterPackage = value; }
        }
        public TimeSpan CreateTime
        {
            get { return _CreateTime; }
        }
        public TimeSpan TotalTravelTime
        {
            get { return _TotalTravelTime; }
            set { _TotalTravelTime = value; }
        }
        public TimeSpan ArrivalTime
        {
            get { return _ArrivalTime; }
            set { _ArrivalTime = value; }
        }
        /*public TimeSpan TotalWaitingTime
        {
            get { return _TotalWaitingTime; }
        }*/
        public PackageAgent Agent
        {
            get { return _Agent; }
        }
```

```
        public Package(int ID, TimeSpan createTime, float value, float
timeValue, int startDC, int endDC, PackageAgent agent)
        {
            _ID = ID;
            _CreateTime = createTime;
            _Value = value;
            _TimeValue = timeValue;
            _StartDC = startDC;
            _EndDC = endDC;
            _Agent = agent;
        }

    }
}
```

8. Class PackageGenerator create PackageGenerator objects

```
 using System;
using System.Collections.Generic;
using System.Text;
using Simulation;
using Simulation.Expressions;

namespace LogisticsNetwork
{
    class PackageGenerator : NetworkSimulationObject
    {
        int _PackageID = 1;
        Expression<int> _RandomStartDC;
        Expression<int> _RandomEndDC;
        Expression<float> _PackageValue;
        Expression<float> _PackageTimeValue;
        Expression<TimeSpan> _GenerationInterval;
        RepeatingMethodEvent _PackageCreateEvent;
        Expression<PackageAgent> _PackageAgent;
        private bool _Enabled = true;

        public bool Enabled
        {
            get { return _Enabled; }
            set { _Enabled = value; }
        }
        public PackageGenerator(Expression<TimeSpan> generationInterval,
Expression<float> packageValue, Expression<float> packageTimeValue,
Expression<int> randomStartDC, Expression<int> randomEndDC,
Expression<PackageAgent> packageAgent)
        private void GeneratePackage()
        {
            if (_Enabled)
            {
                int start = 0;
                int end = 0;
                while (start == end) //eliminate local demand
                {
                    start = _RandomStartDC.Evaluate();
                    end = _RandomEndDC.Evaluate();
                }
            }
        }
    }
}
```

9. Class Routing create Routing objectives

```
 using System;
using System.IO;
using System.Text;

namespace LogisticsNetwork
{
    //load route file into _Routes matrix
    class Routing
    {
        int[,] _Routes;
        float[,] _Distances;
        public Routing(string routesFile, string distancesFile, int
numberOfDCs)
        {
            _Routes = new int[numberOfDCs, numberOfDCs];  //create blank
array
            _Distances = new float[numberOfDCs, numberOfDCs];

        //getting next DC it should go to, given the current DC
        public int GetNextDC(int current, int end)
        {
            return _Routes[current, end];
        }
        public float GetDistance(int from, int to)
        {
            return _Distances[from, to];
        }
    }
}
```

10. Class Truck creates truck objectives

```
 using System;
using System.Collections.Generic;
using System.Text;
using Simulation;
using LogisticsNetwork.Properties;

namespace LogisticsNetwork
{
    class Truck : NetworkSimulationObject
    {
        private TruckAgent _Agent;
        private LoadingArea _ContractedArea;
        private int _ID;
        private int _LastDCID;

        private TimeSpan _TruckLastArrivalTime;
        private TimeSpan _LastTravelTime;
        private TimeSpan _TotalTravelTime;
        private int _LastLoadSize;
        private double Rho;   //Truck utilization on arc level is suppose
to be 0.8;

        public int LastLoadSize
        {
            get { return _LastLoadSize; }
            set { _LastLoadSize = value; }

        }

        public TimeSpan TotalTravelTime
        {
            get { return _TotalTravelTime; }
        }
        public int ID
        {
            get { return _ID; }
        }
        public int LastDCID
        {
            get { return _LastDCID; }
        }

        public LoadingArea ContractedArea
        {
            get { return _ContractedArea; }
            set { _ContractedArea = value; }
        }
        public TruckAgent Agent
        {
            get { return _Agent; }
        }
        private DC _Location;
        public DC Location
        {
            get { return _Location; }
        }
        private static float TruckSpeed = Settings.Default.TruckSpeed;
//truck speed 60mph
        private static TimeSpan LoadUnloadTime = TimeSpan.FromMinutes(2 *
Settings.Default.LoadUnloadTime);  //loading and unloading time

        private List<Package> _Packages = new List<Package>();

        public ICollection<Package> Packages
        {
            get { return _Packages; }
        }
        protected override void Initialize()
        {
            base.Initialize();
            DC dc =
NetworkSimulator.GetDC(NetworkSimulator.RandomDC.Evaluate());
            ArriveAt(dc);
            dc.BiddingTrucks.Add(this);
        }
        public void DepartFor(DC dc)
        {
            _LastDCID = _Location.Number;
            TimeSpan travelTime =
TimeSpan.FromHours(NetworkSimulator.Routing.GetDistance(_Location.Number,
dc.Number) / Truck.TruckSpeed);
            //travelTime += LoadUnloadTime;
            _TotalTravelTime += travelTime; //accumulate total travel time
for this truck
            for (int i = 0; i < _Packages.Count; i++)
            {
                NetworkSimulator.ArcDelay[_Location.Number, dc.Number] +=
EventCalendar.Now.TotalHours - _Packages[i].ArrivalTime.TotalHours;  //sum
up package's wait at that arc
                //double trafficdensity_old = 0.8;
                //double trafficdensity =
(NetworkSimulator.ArcLoadSizeMean[_Location.Number,
dc.Number]/Settings.Default.TruckCapacity) / (1 -
(NetworkSimulator.ArcLoadSizeMean[_Location.Number,
dc.Number]/Settings.Default.TruckCapacity));
                //if (trafficdensity > 0.999)
                //{
                //    trafficdensity = trafficdensity_old;
                //}
                //trafficdensity_old = trafficdensity;
                Rho = 0.8;
                double trafficdensity = Rho / (1 - Rho);
                double CVPart = 0.5 *
NetworkSimulator.ArcTrucksIntArrVariance[_Location.Number, dc.Number] /
Math.Pow(NetworkSimulator.ArcTrucksIntArrMean[_Location.Number,
dc.Number], 2) +
NetworkSimulator.ArcPackageIntArrVariance[_Location.Number, dc.Number] /
Math.Pow(NetworkSimulator.ArcPackageIntArrMean[_Location.Number,
dc.Number], 2);
                NetworkSimulator.ArcDelayApprox[_Location.Number,
dc.Number] += trafficdensity * CVPart *
NetworkSimulator.ArcTrucksIntArrMean[_Location.Number, dc.Number];
            }
            EventCalendar.Schedule(new MethodEvent<DC>(EventCalendar.Now +
travelTime, ArriveAt, dc));
            _LastTravelTime = travelTime;
            _Location = null;
            _ContractedArea = null;
            dc.BiddingTrucks.Add(this);
            dc.Offering();
        }
        private void ArriveAt(DC dc)
        {

NetworkSimulator.TrucksArrivalSCV.Record(EventCalendar.Now.TotalHours -
_TruckLastArrivalTime.TotalHours);
            _TruckLastArrivalTime = EventCalendar.Now;


            if (EventCalendar.Now != TimeSpan.Zero)  //initially trucks
are generated and travelling empty to first DC. From there they load
packages. so excludes those truck from loadfactor.
            {
                NetworkSimulator.TruckLoad.Record(_Packages.Count);
                //record the arc level loads interarrival time mean and
variance to be used for package waiting time approximation with queueing
theory
                NetworkSimulator.ArcTrucksIntArrCt[this._LastDCID,
dc.Number]++; //record how many trucks arrivals it has so far
                double t =
NetworkSimulator.ArcTrucksIntArrCt[this._LastDCID, dc.Number];
                double period = EventCalendar.Now.TotalHours -
NetworkSimulator.ArcLoadsLastArr[this._LastDCID, dc.Number];  //current
observation, current interarrival time period
                NetworkSimulator.ArcTrucksIntArrMean[this._LastDCID,
dc.Number] = (1 - 1 / t) *
NetworkSimulator.ArcTrucksIntArrMean[this._LastDCID, dc.Number] + period /
t;  //recursive time average
```

```
                if (t > 1)  //recursive time variance only starts at t=2. 
assume variance=0.1 when t=1
                {
NetworkSimulator.ArcTrucksIntArrVariance[this._LastDCID, dc.Number] = (1 - 
1 / t) * NetworkSimulator.ArcTrucksIntArrVariance[this._LastDCID, 
dc.Number] + Math.Pow(period - 
NetworkSimulator.ArcTrucksIntArrMean[this._LastDCID, dc.Number], 2) / (t - 
1); //recursive time variance
                }
                NetworkSimulator.ArcLoadsLastArr[this._LastDCID, 
dc.Number] = EventCalendar.Now.TotalHours; //update last arrival time for 
this arc
                int CurrLoadSize = this._LastLoadSize;  //current 
observation
                NetworkSimulator.ArcLoadSizeMean[this._LastDCID, 
dc.Number] =Math.Min( (1 - 1 / t) * 
NetworkSimulator.ArcLoadSizeMean[this._LastDCID, dc.Number] + CurrLoadSize 
/ t, Settings.Default.TruckCapacity);
            }
            //NetworkSimulator.TruckLoadFactor.Record(_Packages.Count / 
Settings.Default.TruckCapacity);
            _Location = dc;  //indicate this truck has arrived this DC
            for (int i = 0; i < _Packages.Count; i++)
            {
                _Packages[i].TotalTravelTime += _LastTravelTime;
                _Location.ReceivePackage(_Packages[i]);
            }
            _Packages.Clear();

            if (_ContractedArea != null)  //if truck has load accepted 
already
            {
                _ContractedArea.LoadTruck(this, EventCalendar.Now);
                DepartFor(_ContractedArea.DestinationDC);
            }  // if the truck agent value is 0, then it will accept then 
empty load, go to next DC. it is equivalent to wandering to a random dc, 
truck is not sticking around.
            //dc.Offering();
        }
        public Truck(TruckAgent agent, int id)
        {
            _ID = id;
            _Agent = agent;
        }
    }
}
```
11. Class TruckAgent creates truckAgent objects
```
using System;
using System.Collections.Generic;
using System.Text;

namespace LogisticsNetwork
{
    abstract class TruckAgent
    {
        public abstract bool Offer(Truck truck, LoadingArea area);
    }
}
```

Class ValueBasedPackageAgent creates valueBasedPackageAgent objects

```
 using System;
using System.Collections.Generic;
using System.Text;

namespace LogisticsNetwork
{
    class ValueBasedPackageAgent : PackageAgent
    {
        public override float InitialBidForPackage(Package package)
        {
            return package.Value * 0.1f;
        }

        public override bool BidForPackage(Package package, out float
bidAmount)
        {
            bidAmount = 0;
            return false;
        }
    }
}
```