Summer 2018

Optimal Supply Delivery Under Military Specific Constraints

TaLena Fletcher
Georgia Southern University

OPTIMAL SUPPLY DELIVERY UNDER MILITARY SPECIFIC CONSTRAINTS

by

TA'LENA FLETCHER

(Under the Direction of Ionut E. Iacob)

ABSTRACT

Through-out military history, the need to safely and effectively allocate resources to various military operations was a task of extreme importance. Satisfying the needs of multiple consumers by optimally pairing with appropriate suppliers falls into the category of vehicle routing problems (VRP), which has been intensively studied over the years. In general, finding the optimal solution to VRP is known to be NP-hard. The proposed solutions rely on mathematical programming and the size of the problems that can be optimally solved is typically limited. In military settings, balancing the needs of multiple consumers with the current operational environment has always been a challenge. This balancing is equally crucial to the survivability of transporters and consumers. The main goal is finding an optimal way of ensuring required delivery while minimizing Soldiers risks. We show that under certain assumptions we can formulate this problem as a linear programming problem with specific constraints.

INDEX WORDS: Vehicle routing problem, Delivery optimization, Military logistics

2009 Mathematics Subject Classification: 90-08, 90C11, 90C90

OPTIMAL SUPPLY DELIVERY UNDER MILITARY SPECIFIC CONSTRAINTS

by

TA'LENA FLETCHER

B.S., Troy University, 2010

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

OPTIMAL SUPPLY DELIVERY UNDER MILITARY SPECIFIC CONSTRAINTS

by

TA'LENA FLETCHER

Major Professor:	Ionut E. Iacob
	Goran Lesaja
Committee:	Scott Kersey
	Hua Wang

Electronic Version Approved:
July 2018

DEDICATION

This thesis is dedicated to my husband, without his support and love I would not have ventured down this path.

ACKNOWLEDGMENTS

I wish to acknowledge Dr. Iacob for agreeing to be my thesis advisor and assist me with modeling an unorthodox problem. Your knowledge and unwavering support will forever be appreciated and remembered.

I would also like to thank MAJ Parker for being my military sounding board, ensuring that the topic of my thesis was relevant and worth researching.

I would also like to thank Ms. Tina Flanagan and Mr. Morris Hayes who agreed to sponsor me and provided me with the data to be modeled.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbols	**Description**
C	Transportation Cost
$N(V, E)$	Routing Network
V	Vertex Set
E	Edge Set
w_{ij}	Weight capacity per edge
b_i	Node identifier
x_{ij}	Commodity Quantity per edge
c_{ij}	Transportation cost per edge
x^c	c denotes type of commodity
o_{ij}	Overhead per edge
q_j^c, Q_j^c	Quantity: lower and upper bound
m_{ij}, M_{ij}	Trucks: lower and upper bound
δ_{ij}	Route identifier

Table 1: Table of symbols

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Throughout military history, the complicated mission of moving supplies whether by air or ground around the battlefield has always been a daunting task. Balancing the needs of multiple consumers in conjunction with the current operational environment is always a challenge. This balancing act is crucial to the survivability of the consumers. So, what is the optimal way of ensuring maximum delivery while minimizing Soldiers risk?

Answering this question is not as straightforward as you may think; delivering supplies in a hostile environment creates many challenges. One of those challenges is whether you are operating in a constrained or unconstrained network. A constrained network limits your ability to efficiently deliver supplies to the consumer and an unconstrained network has no barriers. Historically, methods have been developed to address supply movement in the above two networks. However, for this paper we will focus on what is the most optimal way of delivering supplies in a constrained network. So, the new question is, how do you account for these variables while still trying to maximize delivery and minimize Soldiers risk. Since, there are so many challenges we face when moving around the battlefield in a constrained environment, it is best to focus on three items. Those three key items are listed below:

- Fulfillment of supply demands, reach back to threshold, convoy capacity (maximize)
- Distance traveled, Soldier hours of exposure (risk) and number of convoys (minimize)
- Transportation Cost (overhead items) (minimize)

Supply delivery is based on consumer request and driven by consumer demand. Con-

sumer request is determined by the consumption rates of the commodities being used. Now, certain commodities are consumed at a faster rate than other commodities and therefore can drive the deliveries to be executed in an inefficient manner. If we allowed our consumers to determine the delivery of their supplies this would cause deliveries to be handled in an ineffective manner because they are driven by the individual needs of the consumer rather than the needs of the network. Therefore, we have to employ a method that would be cost-effective in terms of resources being employed and timely as well. We need a system where we can forecast the deliveries of the consumers based upon their consumption of the commodities.

1.2 The Vendor Managed Inventory

The Vendor Managed Inventory (VMI) provides us with a mechanism where we can have more oversight of the scheduling of the deliveries and optimization of the delivery routes. The VMI is a system in which the consumer will relinquish resupply request rights to the supplier, meaning the supplier determines when the consumer will receive their good (how much and how often). In turn, the supplier agrees to not allow the consumer to fall below a pre-determined threshold amount [4]. Utilizing this system will allow us to be able to control the scheduling of our deliveries. The VMI combines the Vehicle Routing Problem (VRP) and Inventory Routing Problem (IRP) into one cohesive unit. For the purpose of this thesis we explain both components of the VMI but only provide a solution to the VRP. The purpose for discussing the IRP is to help you understand the complexity of the VMI as whole. It is only for context purposes.

1.2.1 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem and a generalization of the Traveling Salesman Problem (TSP). The TSP determines the most optimal

(cheapest) way of traveling through a network of cities. In its general form the traveling salesman only use one vehicle and to achieve optimality the traveling salesman will start and stop from the same point and the route must go through each city only once without traveling over any roads twice. The transportation cost in a TSP are either symmetric meaning the distance traveled between two cities both ways are the same or asymmetric meaning the distanced traveled between two cities can be different or the path does not exist in the other direction. The TSP has one glaring and obvious limitation, it uses only one vehicle and this vehicle has a limited amount of capacity. When the demand of the network exceeds the capacity it forces another vehicle to be added to fulfill all delivery requirements then the problem becomes a VRP. The VRP answers the question "How do I maximize delivery of my supplies while minimizing transportation cost?" The VRP is broad in scope and can be applied to any industry, and for this thesis the application will be in the construction of a military supply system. The basic definition of the VRP states "that m vehicles initially located at a depot are to deliver discrete quantities of goods to n consumers". The VRP problem determines the optimal routes to deliver the goods to the consumers in such a way that transportation cost is minimized. The VRP started in its infancy with mathematicians George Dantzig and John Ramser in 1959. They developed an algorithm to address optimizing gasoline delivery to service stations then in 1964 Clarke and Wright improved upon this algorithm by employing a greedy heuristic. Through this improvement several variations of the VRP has been developed to address different applications and industries. The basic characteristics of a vehicle routing problem are outlined below [32]:

- **Road Network**: represented by a directed or undirected graph. A directed graph is unidirectional meaning travel can only happen in one direction. An undirected graph is bidirectional meaning travel can happen in both directions. The graph is made up of vertices and edges. The vertices represent the depot and consumers. The edges will be the paths connecting the vertices. The edges will incur a transportation cost

(time and distance) and this is crux of the minimization problem.

- **Customers**: will request goods to be delivered at certain time intervals.
- **Depots**: will fulfill the request of the customers, contains the required amount of commodities to be delivered.
- **Vehicles**: have limited capacity based on the type of vehicle it is and the location of the vehicle.
- **Drivers**: set number of individuals responsible for getting the goods to their proper destination.
- **Operational Constraints**: are limitations placed on the routes, such as distance, vehicle capacity or usable road networks.
- **Objective Functions**: other than transportation cost minimization, do you want to maximize vehicle usage or minimize driver time on the road.

Starting with Dantzig and Ramser and then to Clarke and Wright the VRP has been modeled over the past 80 years to account for the different variations that has arisen through the different applications to solve real world problems. You can now categorize these different models into three basic categories [32]:

- **Vehicle flow formulations**: use integers and counts the number of times a vehicle travels over each edge. This type of model is best suited for cases when you can sum the global transportation cost across all the edges of the network. This model is not suited when constraints are vehicle dependent.
- **Commodity flow formulation**: uses integers as well and is based on the amount of commodity traveling through the network.

- **Set-Partitioning Problem (SPP)**: uses binary variables which is connected to a subset of optimal routes. This model address the grouping of optimal routes with associated minimal cost that delivers to a consumer once and address additional constraints.

All of these models are associated with one of the most well known vehicle routing problems, the Capacitated Vehicle Routing Problem (CVRP). The CVRP is where there is a known demand at each location on the delivery route and the sum of that demand cannot exceed the capacity of the vehicles on that route. In the next section we will examine this problem and its formulations.

1.2.2 The Capacitated Vehicle Routing Problem

Amongst the class of VRP, the CVRP is the one which our problem formulation was constructed. We will first examine the basic notation and formulation of the CVRP and compare and contrast that with our problem formulation. Next, we will examine how we utilized different aspects of other formulations to form our hybrid problem formulation. The CVRP operates using a single depot denoted by 0 from which a fleet of vehicles K leave to service the consumer demand $q_i \geq 0$ along their dedicated routes and returns back to depot. The dedicated routes are identified by $r = i_0, i_1, \cdots, i_{s+1} : i_0 = i_{s+1}$ and the consumers will be a subset of the overall consumer set N. This operation incurs a transportation cost c_{ij} along that particular edge. The characteristics of the vehicle fleet is assumed to be homogeneous in nature, meaning all vehicles are identical from fuel consumption to load capacity Q and therefore incur the same cost. In order to determine route optimality you have to calculate the transportation cost along each edge and identify the type of network you have. As we mentioned before the CVRP can be represented by a directed or undirected graph. The structure of these graphs are the same in that they are complete and there is a set of vertices identifying the consumers and depot denoted by $V = 0 \cup N | N \in 0, 1, \cdots, n$ and a set of edges identifying the routes between the supplier and the consumers. It is the

identification of the edge set in which they differ. This is due to the symmetric or asymmetric nature of the network. Since, the VRP is a generalization of the TSP this is one of the traits it inherits. In a symmetric network, the transportation cost association from supplier to consumer and vice versa is the same and therefore is associated with an undirected graph, $G = (V, E)$. So, the edge set will be identified as $E = \{i, j| \in V, i \neq j\}$. On the other hand if the transportation cost is not the same in different directions along the same edge or that edge connection does not exist in one of the directions then your network is asymmetric, and your associated graph is directed $G = (V, A)$. So, instead of an edge set you have an arc set and it will be identified as $A = \{(i, j)| \in V \times V, i \neq j\}$ [33]. The overall purpose of the CVRP is to identify a set of optimal routes to minimize transportation cost across the network.

In comparison to our problem there are several things we had to do to adjust the standard CVRP to our problem formulation. First, the notation used for the vertex set is structured for a one supplier network which does not address our need to have two suppliers and in some situations possibly more. Currently, the set V is denoted by $V = \{0 \cup N|N \in 0, 1, \cdots, n\}$ and this will change to $V = \{1, 2, \cdots, n\}$; redefining the supplier-consumer set this way affords us the flexibility to account for multiple suppliers in our network. Second, the CVRP is structured as a directed graph or an undirected graph however due to the complexity of our problem we had to change this structure to a bipartite graph. This allowed for us to partition our suppliers into one set and our consumers into another set. Thereby allowing us to properly account for the dual delivery possibilities. For example, our problem contains 2 suppliers and 8 consumers and as you will see in Experiment 1 supplier $C2$ only delivers to 3 of the consumers while supplier $C3$ delivers to all. The way the standard CVRP is structured there was no way to account for multiple depots in the problem formulation. This brings us to the last revision of the standard CVRP, that is the problem formulation. During the initial stages of our problem it was structured as a ver-

sion of the two-index (vehicle-flow) formulation VRP1 [33]. However as the full problem began to take shape we recognized there were several shortcomings in this formulation, which is laid out in detail in chapter 2. A major shortcoming of this formulation was there was no way to account for the additional trucks required to secure a convoy. The current formulation only provides quantities for the amount of trucks needed to carry the commodity but not the amount of trucks needs to secure the convoy. This is because in the civilian sector securing a delivery route is not a requirement however in the military it is one of the most important aspects of supply delivery. For this reason, the problem formulation was revised to include a key aspect of the set partition formulation, binary variables. By adding a binary variable into the problem formulation it provided a way to include the calculation of the overhead and security cost along an edge. Some of the other key items to mention here is the usage of the variable x and the characteristics of the fleet. In VRP1 and subsequent formulations x represents the number of times an edge is traveled, however in our problem formulation it will represent the quantity of a commodity that is traveling across that edge. In standard CVRP formulations the fleet of trucks are thought to be homogenous because there is no way to account for variances in truck capacity but our fleet of vehicles is a heterogeneous mix. Now there are additional formulations you can employ that would partition the vehicles into subsets of like quantities thereby making them homogenous [33] but we didn't want to include this type of formulation because our vehicles are commodity driven and therefore can be utilized as a constant.

1.2.3 Inventory Routing Problem

In their paper Inventory Routing Problems: An Introduction, Bertazzi and Speranaza outlines the basic characteristics of an IRP[4].

- **Time** is the main discriminate in an inventory routing problem. You want to ensure that you deliver the right quantity to the right customer at the right time. All

customers are not going to require deliveries at the same time or on the same day.

  - **Continuous**: there are no restrictions set on shipping, it can happen at anytime.
  - **Continuous with a minimum intershipment time**: the main restriction set on shipment is the intershipment time, which is the time between shipments. This time cannot fall below a preset time. This allows for the customer to have time to receive and set-up the inventory before the next shipment arrives.
  - **Discrete**: This shipment times is related to intershipment time.

- **The planning horizon** determines how long you are supposed to deliver to a particular consumer.

  - **Infinite**: this is used when trying to establish a more permanent distribution plan. One that will be used over a significantly longer time. This plan can be used to assist a company with understanding the amount of resources required to supply a certain number of customers. It can be used as a forecasting tool.
  - **Finite**: this is used for short-term planning.

- How you implement the shipment times and planning horizon is referred to as a **structured policy**. There are several different types of structured policies highlighted in their paper. However, I will only discuss the ones that are pertinent to this model.

  - **Frequency-based**: this policy is referred to as periodic because it establishes how often a shipment will occur based on a predetermined frequency. In our case we have established threshold for each product and this will be used as the determining factor for how often a consumer will be replenished.
  - **Order-up-to-level**: this defines how much you will deliver to your consumer. The goal with this policy is to always deliver enough product to the consumer which will get them to maximum level.

  - **Maximum level**: this policy is a generalization of the order-up-to-level, this just means that a consumer will never exceed their maximum level. Meaning that the supplier cannot deliver more product than the consumer can store.

- The **objective function** of the IRP determines what you are trying to optimize. If the function is not properly well-defined you will not be able to find an optimal solution. In most IRPs the objective is to either minimize transportation cost or inventory cost. There are significant drawbacks experienced in trying to minimize one or the other and it has been shown to be more effective to minimize both of them. In our paper, inventory cost minimization is not as high of a priority as it is to minimize transportation cost. Which is why this portion of the problem will be addressed in a different paper.

- Finally, the most important factor of IRPs, **deciding when and which routes to traverse**, just like VRPs. There are two different types of decisions that can be used but they depend heavily on the type of environment they are employed in. So, let us briefly explain the two different types of environments, retailer managed inventory (RMI) and VMI. A supplier operating in an RMI is held captive based on the customer's wishes. The customers determine when and how much of a product the supplier will deliver. The VMI has been discussed previously and therefore will not be repeated here. In comparing the two inventory managed system the one similarity is determining when and how much of a product, this is an aspect of a decision space. The decision space is defined by two different categories:

  - **Decisions over time only**: With this decision the routes are predetermined for the supplier so the only thing that has to be determined is when and how much to deliver. Generally, this is done by the customer and can therefore be associated with the RMI environment.

– **Decisions over time and space**: With this decision the routes, when the shipment occurs and how much to be delivered are determined at the same time. For this reason, this can be associated with the VMI environment because this is what the supplier will be responsible for figuring out.

1.3 Military Standard Problem

Logistics in the most generic form is the process by which an item is moved from its initial point to its final destination. This definition works well in the civilian sector however in the military, logistics becomes a complex operation. According to the Maneuver Center of Excellence "military logistics is the processes, resources, and systems involved in generating, transporting, sustaining, and redeploying or reallocating materiel and personnel." Military logistics plays a pivotal role in the success of any mission. The ability to sustain the warfighter who is immediately behind enemy lines marks the difference between winning and losing a war and for this reason sustainment operations can be seen as a combat multiplier. Sustainment operations are conducted along eight pillars: integration, anticipation, responsiveness, simplicity, economy, survivability, continuity, and improvisation. While all 8 pillars are key and integral to successful combat operations, in our context there are a few that stand out among the rest[24]. Our problem is structured to operate in a vendor managed inventory (VMI) construct and within this anticipation, simplicity, economy and continuity are of most important. Operating in a VMI environment there is an agreement between the supplier and consumer that the consumer will not fall below a pre-determined threshold therefore anticipation of the consumer needs keeps this agreement intact. The supplier has fore-knowledge of the consumers' requirements through the usage of the commodity consumption rate by each consumer and can therefore forecast out the proper delivery times for each commodity. This brings us to the next two principles, simplicity and economy. Simplicity is having a well-defined resupply plan and clearly es-

tablished chain of command. This principle works in connection with economy because if the plan is not completely laid out then it can lead to inefficiencies. Economy is the ability to effectively and efficiently conduct resupply operations. This translates to the supplier having the necessary available resources on hand to ensure timely delivery of supplies to the consumers. These resources include but not limited to vehicles, personnel, the necessary infrastructure to house the supplies, and they can be all organic or a combination of organic with contracted support. A typical military supply problem is constructed with one supplier and multiple consumers, Figure 1.1 is an example of a standard military resupply operations. Resupply operations start at the port (sea or air) where the supplies are processed for onward movement to either the Theater Distribution Center (TDC) or to a main supplier labeled as C1 - C7 in Figure 1.1. The resupply chain is hierarchial in nature. It starts with the 7 main suppliers and stops at the company labeled as XS throughout the figure. This resupply occurs three different ways:

- **Unit distribution:** supplies are configured in unit sets (battalion/company/platoon, depending on the level of distribution) and delivered to one or more central locations or directly to the unit's final location. Unit distribution maximizes the use of the Brigade Combat Team (BCT) lift capacity of its transportation assets and minimizes the delivery and turnaround time. Figure 1.1 is a typical supply network.

- **Supply point distribution:** requires unit representatives to move to a supply point to pick up their supplies. Supply point distribution is most commonly executed by means of a logistics release point. The logistics release point (LRP) may be any place on the ground where unit vehicles return to pick up supplies and then take them forward to their unit. Occasionally, the logistics release point is the brigade support area (BSA).

- **Throughput:** The supplies will be delivered from the port directly to the requesting

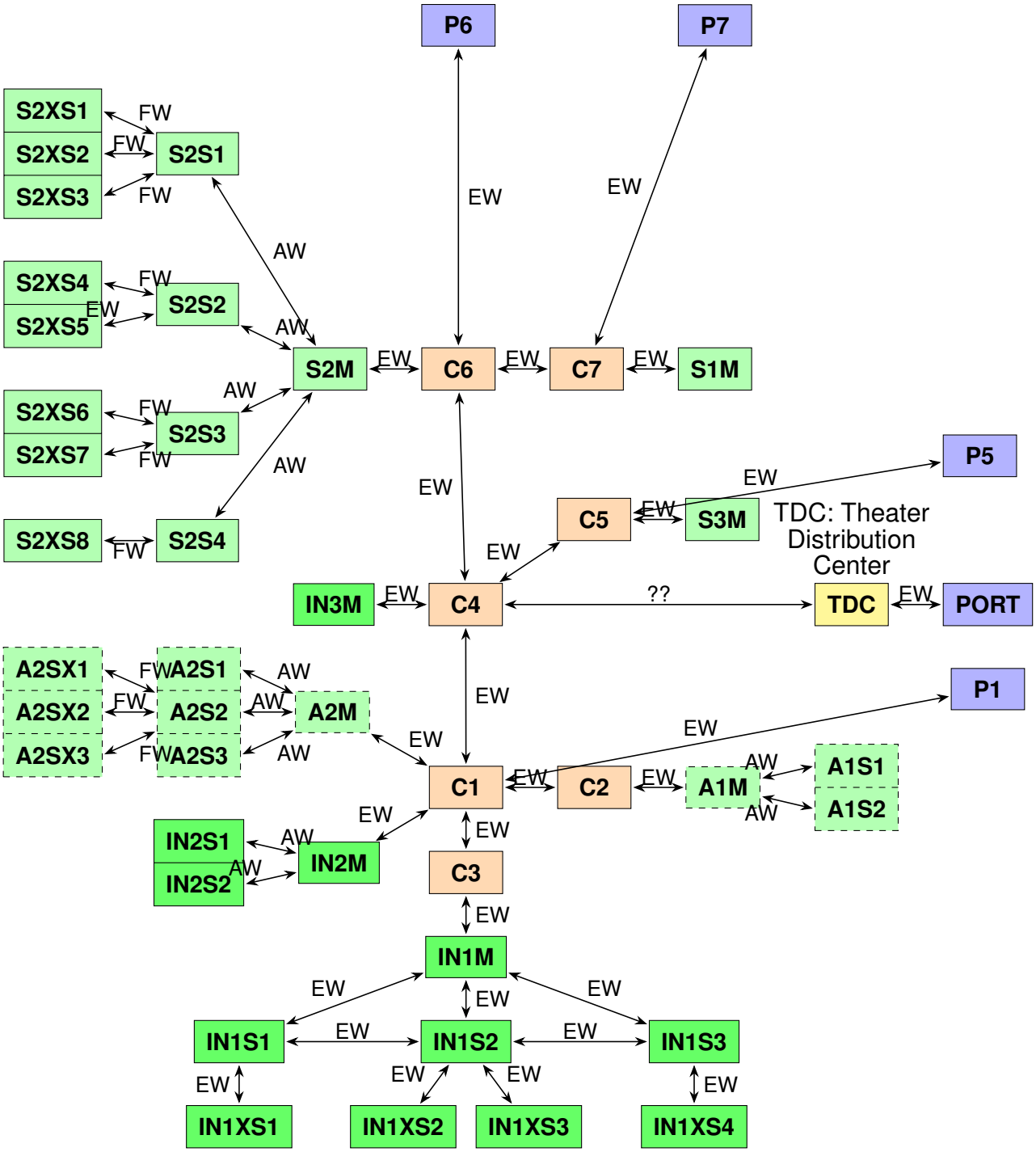

Figure 1.1: A typical military suppliers-consumers network

unit. This process bypasses the main suppliers and the TDC.

The resupply operations discussed above are conducted via convoys or through air

delivery. Air delivery is the safest and most optimal way to deliver supplies to units however mission priorities, weather, location or cost will prohibit continual usage of this asset. Therefore when air assets are not available commodities must be delivered via ground assets through convoy operations. Convoy operations have evolved over time from the Vietnam War through operations in Afghanistan and Iraq. During Vietnam it was thought that daytime operations were best because the enemy owned the night however this proved problematic because during a daytime attack you cannot pinpoint the muzzle flash of the enemy so operations were changed to the night time. Nighttime convoy operations continued through Operation Iraqi Freedom (OIF) and Operation Enduring Freedom (OEF, Afghanistan) because there were no civilians on the road due to a curfew being imposed on the civilian population. However this proved to be very problematic because during OIF and OEF improvised explosive devices (IEDs) were heavily employed due to the enemies ability to conceal them along the side of the road or burying them in the road.[16] From these two examples, it is evident that the security of the convoy is paramount. Therefore, convoy operations are conducted under very strict guidelines because Soldiers are being exposed to hazards that cannot be mitigated away. While there are many items that must be followed during convoy operations we will discuss two of them for this paper, security and convoy composition. A standard convoy composition is outlined below[22]:

- Head
  - Security Vehicle
  - Lead Vehicle Commander
- Main body
  - Additional security trucks
  - Logistics Trucks

  - Convoy Commander
  - Additional trucks for convoy personnel

- Trail
  - Medical Vehicle
  - Recovery Support
  - Security Trucks
  - Assistant Convoy Commander

The number of vehicles in a convoy varies depending on the mission. The commander of that geographic area will set the minimum/maximum number of trucks required for a convoy. Generally, the minimum number of trucks required to conduct convoy operations is 5 while the maximum number is 30 and the number of security trucks is based on the convoy size and is implemented on a $1:5$ ratio, being 1 security truck per 5 logistics trucks.[16] The convoy composition detailed above contains what we refer to as overhead. Overhead is the standard trucks that are required for every convoy, they are medical, recovery support and security trucks. These trucks are mainly located in the trail portion of the convoy while the security trucks are dispersed throughout the convoy.

CHAPTER 2

PROBLEM DESCRIPTION

2.1 PROBLEM INTRODUCTION

A constrained environment prevents a free flow of transportation from supplier to consumer and within the delivery network. When operating in this type of environment the objective is to minimize or eliminate the obstacles which prevents the network from operating smoothly. Within our delivery network the major obstacles our supplier have to contend with are distance, security, convoy size, product availability and road type. Some things to consider but will not greatly distress the network are inventory holding capacity at the consumer and supplier and vehicle capacity. Inventory capacity at the consumer will not be exceeded because the threshold quantity will control this variable. Vehicle capacity is not a problem because the requested amounts will never exceed the capacity of the lightest vehicle. Taken these factors into consideration the problem allows security to dictate the optimal route. Security is priority one because the enemy is always looking for ways to disrupt the movement of personnel and supplies around the battlefield. Knowing this, we have designed for each edge of the delivery network to have a security overhead denoted by o_{ij} associated with delivering supplies along that edge of the network. This will account for the different security requirements throughout the network. The following is how we classify different security requirements which can be adjusted as the situation changes:

- Normal: requires no additional vehicles
- Moderate: requires 4 additional security vehicles
- High: requires 8 additional security vehicles

Let us first discuss the classic problem and then we will address how we used this formulation and created our hybrid solution.

2.2 Problem definition

2.2.1 The Minimum-cost network-flow problem

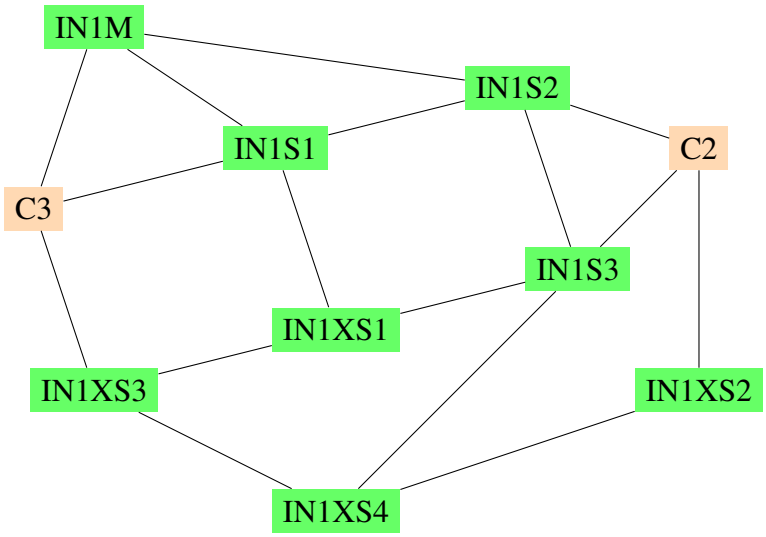

Figure 2.1: A Suppliers-Consumers Network

We considered the standard network flow optimization [1] as a first, natural choice for modeling our military supplier-consumer delivery problem. Figure 2.1 is an example of geographical placements and connections between a few suppliers ($C2$, $C3$) and consumers ($1M$, $1S1$, $1S2$, $1XS1$, $1XS2$, $1XS3$, $1XS4$) of the hierarchical distribution scheme in Figure 1.1. The problem is formulated as follows.

Definition 2.1 (Standard Network Flow Problem)**.** *Let $N(V, A)$ be a suppliers-consumer network, where $V = \{1, 2, \cdots, n\}$ is the set of vertices and $A = \{(i, j) \in V \times V : i \neq j\}$ is the set of edges (a directed graph, as given in Figure 2.1). Each node $i \in V$ has a divergence value b_i that represents the amount of commodity on stock (if $b_i > 0$, in which case i is a supply node) or needed (if $b_i < 0$, in which case i is a demand node) at node i. Each directed edge $(i, j) \in A$ has a transport cost coefficient c_{ij}, which represents the cost of transporting one unit of commodity, and a maximum capacity Q_{ij}, which represents*

the maximum amount of commodity that can be transported along the edge (i, j) at a given moment of time.

The minimum cost network flow problem for transporting quantity x of commodity across network N is formulated as follows:

$$\textit{Minimize } C = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1a}$$

subject to

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b_i, \quad i = 1, \cdots, n \tag{2.1b}$$

$$x_{ij} \leq Q_{ij}, \quad \forall (i,j) \in A \tag{2.1c}$$

$$x_{ij} \geq 0, \qquad \forall (i,j) \in A \tag{2.1d}$$

The standard network flow problem follows closely the topology of a network as in Figure 2.1, with suppliers ($C2$, $C3$) and consumers ($IN1M$, $IN1S1$, $IN1S2$, $IN1XS1$, $IN1XS2$, $IN1XS3$, $IN1XS4$) connected (or not connected) as in the real world. A decision variable x_{ij} is associated to each network edge (i, j) and the objective in (2.1) is to minimize the cost associated with transporting commodities along all edges (2.1a), subject to delivering requested quantities at each node (2.1b), edge capacities limits (2.1c), and positiveness of the quantities being transported (2.1d).

There are a few major shortcomings of the standard minimum-cost network-flow model in (2.1):

SC1 There is an overhead cost, o_{ij} associated to transporting a non-zero quantity of commodities along each edge (i, j) of the network. These overhead costs are a consequence of using standard detail trucks (medical, security and recovery) and can be considered constant parameters associated to each edge (essentially proportional to the length of the edge).

SC2 Extra-overhead costs may appear along an entire route passing through at least one edge associated with high security risk. These costs are a consequence of using extra security trucks for the whole route from a supplier to a consumer. Similar to the overhead costs, these costs can be considered constant for each edge, proportional to the length of the edge.

SC3 Additional constraints may be imposed to restrict traversal of edges with high security risk. For instance, a cap might be desired for the security costs, which would be equivalent to an upper limit for the security risk along any route.

SC4 Different commodities may have different transportation costs and demand-supply amounts (divergences b_i) at each node varies from commodity to commodity.

SC5 Suppliers have the necessary amount of commodities on hand to service their respective consumers (although not every commodity can be found at each supplier) and these commodities are transported by trucks organized in convoys. There are upper bound limitations on the number of trucks allowed along edges and routes (similar to the edge capacities above), but there are also lower bound limitations on the number of trucks.

We address these shortcomings in the subsequent subsections.

2.2.2 MININUM-COST NETWORK-FLOW UNDER EDGE OVERHEAD AND SECURITY CONSTRAINTS

Let us refine the standard problem of network flow optimization (2.1) to include the edge overhead and security costs. We will stick with the topology of the network to model the real world interconnection between suppliers and consumers and define the enhanced model on networks as in Figure 2.1. However, we will consider distinctively suppliers

and consumer nodes, rather than differentiating them by the sign of the "divergence" value b_i (the amount requested/provided). This distinction is necessary to remove the cap on quantities one supplier can deliver (and hence address the shortcoming SC5). Also, the quantities being transported (represented by variable x) represent the number of trucks.

Definition 2.2 (Network Flow Problem under Edge Overhead and Security Constrains)**.** *Let $N(V, A)$ be a suppliers-consumer network, where $V = \{1, 2, \cdots, n\}$ is the set of vertices and $A = \{(i, j) \mid i, j \in V\}$ is the set of edges (a directed graph, as given in Figure 2.1). Some of the vertices are designed as suppliers, the others are designed as consumers. Each consumer node $j \in V$ has a requested quantity value q_j^c for each commodity c, which represents the amount of commodity c needed at node j. Each directed edge $(i, j) \in A$ has: (i) a transport cost coefficient c_{ij}^c, which represents the cost of transporting one unit (truck) of commodity c; (ii) an overhead and security cost o_{ij} and s_{ij}, respectively; they are constants proportional to the length of the edge; (iii) and maximum and minimum capacities M_{ij} and m_{ij}, respectively; they represent the maximum/minum amount of commodity (number of trucks) that can be transported along the edge (i, j) at a given moment of time.*

The minimum cost network flow problem for transporting x^c trucks of commodity c

across network N is formulated as follows:

$$\textit{Minimize } C = \sum_{i,j} \left(\sum_{c} c_{ij}^{c} x_{ij}^{c} + \delta_{ij}(o_{ij} + s_{ij}) \right) \tag{2.2a}$$

subject to

$$\sum_{i} x_{ij}^{c} = q_{j}^{c}, \quad \textit{for all consumers } j, \textit{ commodity } c \tag{2.2b}$$

$$\sum_{c} x_{ij}^{c} \leq \delta_{ij} M_{ij}, \quad \forall (i,j) \in A \tag{2.2c}$$

$$\sum_{c} x_{ij}^{c} \geq \delta_{ij} m_{ij}, \quad \forall (i,j) \in A \tag{2.2d}$$

$$x_{ij}^{c} \geq 0, \qquad \forall (i,j) \in A, c \tag{2.2e}$$

$$\delta_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A \tag{2.2f}$$

The objective in (2.2) is to minimize the cost associated with transporting commodities along all routes (2.2a), including the overhead and security costs. The inclusion/exclusion binary variables δ_{ij} serve the purpose of including only the costs associated to edges that appear in a route:

$$\delta_{ij} = \begin{cases} 1, & \text{if } x_{ij} > 0 \\ 0, & \text{if } x_{ij} = 0 \end{cases}$$

.

The objective is subject to delivering requested quantities at each consumer node (2.2b), edge capacities limits (2.2c) and (2.2d), positiveness of the quantities being transported (2.2e), and the inclusion/exclusion binary variable (2.2f).

The model is also constructed on a network as in Figure 2.1, for instance, but in this formulation two decision variables are assigned to each edge (i,j): the quantity x_{ij} transported along the edge and the binary variable δ_{ij}.

While the current formulation (2.2) of the problem does incorporate both the overhead and security costs, in practice, it is not yet satisfactory. A subtle issue of shortcoming SC2 in Section 2.2.1 is that the security detail for a whole route from a supplier to a consumer

must be decided by the highest security risk edge in the route. That is, even if some edges need no additional security detail, additional security will be provided along these edges if they are part of a route that contains at least one edge needing additional security. Moreover, the shortcoming SC3 (which requires security limitations *along routes*) is not addressed at all and difficult to incorporate in formulation (2.2). Furthermore, the shortcoming SC5 is incorrectly addressed, as the bounds on the number of trucks (quantities transported) in constraints (2.2c) and (2.2d) act on the overall number of trucks for each edge (from all suppliers), rather than the number of trucks per convoy.

Next, we will address these limitations.

2.2.3 MININUM-COST NETWORK-FLOW UNDER ROUTES OVERHEAD AND SECURITY CONSTRAINTS

One of the major limitations of the model formulation (2.2) consists in its inability to incorporate security risk costs as the *maximum security needed along a whole route*. This limitation clearly affects the solution of the problem in a negative way. Say a route $R1$ has N edges with a single high security edge and many edges with no security restrictions. Another route $R2$ is longer than $R1$, with low security assignments along all edges. As the cost of the high security edge in $R1$ contributes only once in the overall cost (2.2a), the route $R1$ will likely win against route $R2$ (for which a cost of security is added for each edge of the longer route). However, the security overhead trucks are assigned to cover all needed security for the whole route. That is, with the high security needed along some edge of $R1$ the additional security will need to accompany the convoy along all edges of route $R1$. Summing the high security edge cost of $R1$ for all N edges in $R1$, the route $R2$ will become the righteous winner.

In order to correctly incorporate in the model the sensitive aspect of transportation security we remodel the problem as a set partitioning vehicle routing problem (SPVRP,

[18]) rather than a network flow problem. We partition the nodes of the network in two disjoint sets: the set of suppliers (U) and the set of consumers (V). We then formulate and solve the delivery problem under security constraints in two steps.

In step 1, for the original network, we compute the *optimal* route between every supplier i and every consumer j, in the sense that we compute the minimum overhead o_{ij} needed (that includes both the overhead and security costs between i and j) for traveling from supplier i to consumer j. For instance, the network in Figure 2.1 becomes a bipartite graph as in Figure 2.2. Finding the optimal route overhead (that includes both the convoy standard accompanying vehicles and the security detail) between supplier u and consumer v can be performed by iteratively using Dijktra's algorithm for finding the shortest path between u and v. Given that the overhead costs (including security) is proportional to the length of the route between two nodes and that security cost is determined by the highest security level along the route, one can slightly enhance Dijktra's algorithm (we call it *Enh-Dijkstra* in the description below) to return the edge of highest security level along the shortest route in addition to the length of the shortest route. The algorithm for finding the optimal route between nodes u and v is described in Algorithm 1.

The algorithm takes as input the suppliers-consumers network $N(V, E)$ and a pair (u, v) of a supplier and a consumer, respectively. The objective is to find (if possible) the optimal cost route between u and v. The algorithm's idea is quite simple: it uses Dijktra's algorithm to find the shortest path and the edge of highest security. It then computes the cost of route and stop if the cost at the current iteration is higher then the previously found cost. If not higher, the edge of highest security is removed from network and the algorithm proceeds to find again the shortest path and compute its cost. The algorithm also stops if no route is found (in which case the optimal cost is infinity and no route between supplier u and consumer v will be established). Clearly the algorithm iterates and uses Dijktra's at most $|E|$ times (at each iteration an edge is removed). It therefore runs in $O(|E| \cdot |V|^2)$

Algorithm 1 Optimal overhead cost route

procedure OPT-COST($N(V, E), (u, v)$) ▷ computes optimal cost
Input: $N(V, E)$, (u, v)
Output: $optcost(u, v)$
    $optcost \leftarrow \infty$
    $cost \leftarrow \infty$
    do
        $optcost \leftarrow cost$
        $(d, HSedge) \leftarrow Enh\text{-}Dijkstra(N, (u, v))$ ▷ shortest distance, highest security edge
        if no route **then**
            break
        end if
        $cost \leftarrow d \cdot cost(HSedge)$
        if $cost < optcost$ **then**
            remove $HSedge$ **from** $N(E, V)$
        end if
    while $optcost > cost$;
    return $optcost$ ▷ Returns the optimal cost
end procedure

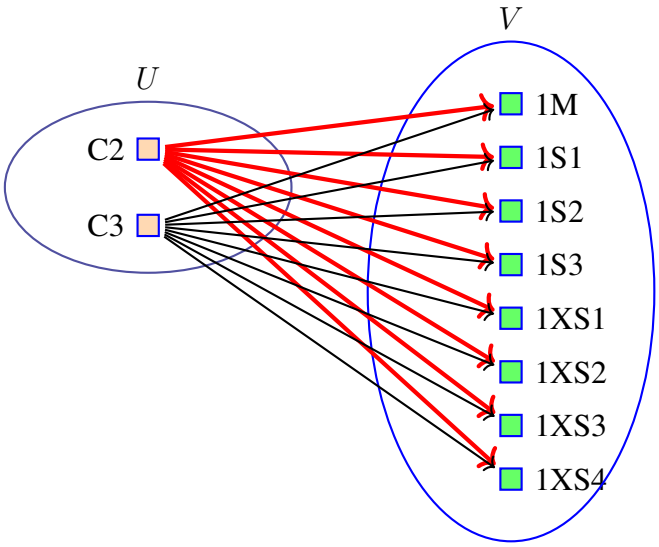

Figure 2.2: Suppliers-consumers under security constraints

time.

In step 2, we consider the bipartite graph of suppliers and consumers and all possible, pre-computed routes between them (see Figure 2.2). The delivery problem under overhead and security constraints is formulated as follows.

Definition 2.3 (Delivery Problem under Route Overhead and Security Constraints)**.** *Let $N(U \cup V, E)$ be a suppliers-consumer network, where $U \cup V = \{1, 2, \cdots, n\}$ is the set of vertices (disjoint union of suppliers and consumers) and $E = \{(i, j) \mid i, j \in V;\ i$ is a supplier; j is a consumer$\}$ is the set of routes from supplier i to consumer j (a bi-partite graph, as in Figure 2.2). Each consumer node $j \in V$ has a requested minimum quantity value q_j^c for each commodity c, which represents the amount of commodity c needed at consumer j. Each directed edge $(i, j) \in E$ has: (i) a transport cost coefficient c_{ij}^c, which represents the cost of transporting one unit (truck) of commodity c; (ii) an overhead and security cost o_{ij}, precomputed as constants incorporating both the length and maximum security along route (i, j); (iii) and maximum and minimum capacities M_{ij} and m_{ij}, respectively; they represent the maximum/minum amount of commodity (number of trucks) that can be transported along the route (i, j) at a given moment of time.*

The minimum cost network flow problem for transporting x^c trucks of commodity c across network N is formulated as follows:

$$\textit{Minimize } C = \sum_{i,j} \left(\sum_{c} c_{ij}^{c} x_{ij}^{c} + \delta_{ij} o_{ij} \right) \tag{2.3a}$$

subject to

$$\sum_{i} x_{ij}^{c} \geq q_{j}^{c}, \quad \textit{for all } j, c \tag{2.3b}$$

$$\sum_{j,c} x_{ij}^{c} \leq \delta_{ij} M_{ij}, \quad \textit{for all } i \tag{2.3c}$$

$$\sum_{j,c} x_{ij}^{c} \geq \delta_{ij} m_{ij}, \quad \textit{for all } i \tag{2.3d}$$

$$x_{ij}^{c} \geq 0, \qquad \forall (i,j) \in E, c \tag{2.3e}$$

$$\delta_{ij} \in \{0,1\}, \qquad \forall (i,j) \in E \tag{2.3f}$$

While formulation (2.3) looks very similar to (2.2), there is a big difference between them: an edge (i,j) in (2.3) represents a whole route starting at supplier i and ending at consumer j, whereas in (2.2) it simply represents a (topological) connection between two network nodes (which can be supplier-consumer or consumer-consumer). As such, the objective (2.3a) incorporates the correct overhead and security costs for the whole route (and hence the shortcoming SC2 is implicitly addressed). The shortcoming SC3 (caps on security costs) can be implicitly address as well by the model: by imposing maximum security costs along certain edges, these costs are included (in step 1) in the overhead cost for a whole route and hence incorporated in (2.3a). Finally, the constraints in (2.3c) and (2.3d) correctly address limitations of number of truck in a convoy sent from supplier i to consumer j. Hence shortcoming SC5 is also removed.

Another subtle difference is the interpretation of the binary variables δ_{ij}:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i \text{ delivers to } j \\ 0, & \text{if } i \text{ does not deliver to } j \end{cases}$$

. This decision variable practically indicates who delivers (no matter which commodity) to whom. In practice, this will give the human decision factors a clear image of delivery plans, which may help other subsequent decision (like combining convoys, etc.).

Notice that, in practice, there might be no routes between certain suppliers and some consumers (not every supplier delivers to every consumer) and hence the suppliers-consumers graph (as in Figure 2.2) does not necessary connect every supplier with every consumer.

2.2.4 Complexity of the Problem

Since this problem is a generalization of the TSP, it inherits the properties of the TSP as well. The TSP is classified as an NP-hard (non-deterministic polynomial-time hardness) problem and therefore the VRP is an NP-hard as well. This is an especially useful attribute to inherit because this means that if an algorithm was created to solve a TSP then it can also be used to solve a VRP. To calculate the computational complexity of a problem, we identified our parameters and constructed a constraints formula. This will provide us with the size of our problem.

Parameters:

- Number of suppliers: S
- Number of consumers: C
- Number of commodities: P

Size of the Model (upper bounds):

- Continuous Variables: $P \cdot S \cdot C$
- Binary Variables: $S \cdot C$
- Constraints: $2(PC + S) + SC(P + 1)$

For our problem: $S = 2$, $C = 8$, $P = 4$ would produce:

$$2(PC + S) + SC(P + 1) = 148 \text{ constraints}$$

This shows that for what appears to be a small problem is actually computationally intensive with 148 constraints.

CHAPTER 3

IMPLEMENTATION

We used Pyomo [13, 12], a Python language based optimization modeling language, for setting up and solving the optimization problems rising from our proposed solution.

Pyomo was specially developed to support intuitive modelling of optimization problems. It can be used to symbolically define optimization problem instances and then delegate the actual solving of these problems to specialized open-source or commercial solvers. Moreover, Pyomo offers support for using algebraic modelling languages, such as "A Mathematical Programming Language" (AMPL [10]), which can be used to separate the problem definition, parameters data, and the problem solver.

Our implemented solution consist of:

1. Pyomo code for defining the problem instance (for the case of the Delivery Problem under Route Overhead and Security Constraints as in Definition 2.3).

2. AMPL-based experiments data: we designed a few experimental data sets to illustrate a few case scenarios and their solutions for optimal delivery under security constraints.

3. Solving the model using two solvers: GLPK [11] and Gurobi [14].

In the following we will describe both the Pyomo formulation of the model (in Python) and the general concepts for designing an experimental data set using AMPL.

3.1 SOLVING THE OPTIMIZATION PROBLEM USING PYOMO

The complete Python code for the model formulation is included in Appendix C. We will explain in the following the main steps for the model formulation, which is the implemented analogue equivalent of the elements of the problem formulated in Definition 2.3.

- The required libraries:

```
import pyomo
import pyomo.opt
import pyomo.environ as pe
import os
```

- Creation of the abstract model of the LP problem in the Pyomo environment:

```
model = pe.AbstractModel(doc=PROBLEM_NAME)
```

- Creation of the parameters of the problem:
  - the number of suppliers and consumers:

```
model.NS = pe.Param(within=pe.PositiveIntegers)
model.NC = pe.Param(within=pe.PositiveIntegers)
```

  - the indexes over suppliers/consumers respectively:

```
model.I = pe.RangeSet(1, model.NS)
model.J = pe.RangeSet(1, model.NC)
```

  - the suppliers/consumers lists, respectively:

```
model.SS = pe.Set(doc='Suppliers')
model.CS = pe.Set(doc='Consummers')
```

  - the lower/upper bounds for quantities:

```
model.m = pe.Param()
model.M = pe.Param()
```

- cost (distance) along each route:

```
model.c = pe.Param(model.I, model.J)
```

- the overload cost (including security) parameters:

```
model.Ot = pe.Param()
model.Of = pe.Param()
def O_init(model, i, j):
    v = 0.0
    if model.c[i,j] >= model.Ot:
        v = model.c[i,j] * model.Of
    return v
model.o = pe.Param(model.I, model.J, initialize=O_init)
```

- the requested quantities for each consumer J:

```
model.qa = pe.Param(model.J)
model.Qa = pe.Param(model.J)
model.qc = pe.Param(model.J)
model.Qc = pe.Param(model.J)
model.qf = pe.Param(model.J)
model.Qf = pe.Param(model.J)
model.qw = pe.Param(model.J)
model.Qw = pe.Param(model.J)
```

- The problem variables:

  - the quantities (number of trucks) for each commodity

```
model.xa = pe.Var(model.I, model.J,
                  domain = pe.PositiveReals) #ammo
```

```
model.xc = pe.Var(model.I, model.J,
                  domain = pe.PositiveReals) #cargo
model.xf = pe.Var(model.I, model.J,
                  domain = pe.PositiveReals) #fuel
model.xw = pe.Var(model.I, model.J,
                  domain = pe.PositiveReals) #water
```

- the binary variables

```
model.d = pe.Var(model.I, model.J,
                  domain = pe.NonNegativeIntegers,
                  bounds = (0,1))
```

- The problem constraints:
  - the requested amounts:

```
def supply_ruleAl(model, j):
  return sum(model.xa[i,j] for i in model.I) >= model.qa[j]


def supply_ruleAu(model, j):
  return sum(model.xa[i,j] for i in model.I) <= model.Qa[j]


def supply_ruleCl(model, j):
  return sum(model.xc[i,j] for i in model.I) >= model.qc[j]


def supply_ruleCu(model, j):
  return sum(model.xc[i,j] for i in model.I) <= model.Qc[j]


def supply_ruleFl(model, j):
```

```
    return sum(model.xf[i,j] for i in model.I) >= model.qf[j]

  def supply_ruleFu(model, j):
    return sum(model.xf[i,j] for i in model.I) <= model.Qf[j]

  def supply_ruleWl(model, j):
    return sum(model.xw[i,j] for i in model.I) >= model.qw[j]

  def supply_ruleWu(model, j):
    return sum(model.xw[i,j] for i in model.I) <= model.Qw[j]
```

  - the for binary variables constraints:

```
  def supply_minq(model, i, j):
    return model.xa[i,j] + model.xc[i,j] + model.xf[i,j]
                  + model.xw[i,j] >= model.d[i,j]*model.m

  def supply_maxq(model, i, j):
    return model.xa[i,j] + model.xc[i,j] + model.xf[i,j]
                  + model.xw[i,j] <= model.d[i,j]*model.M
```

- The objective function definition, which is subsequently stored in the model:

```
def objective_rule(model):
  return sum(model.c[i,j]*(model.xa[i,j] + model.xc[i,j]
            + model.xf[i,j] + model.xw[i,j])
            + model.d[i,j]*model.o[i,j]
            for i in model.I for j in model.J)
```

```
model.objective = pe.Objective(rule=objective_rule,
                  sense=pe.minimize,
                  doc='The objective function')
```

- Populating the model with actual data from an AMPL file:

```
datafile = 'constants2a.dat'
instance = model.create_instance(datafile)
```

- Solving the model (with a choice between GLPK and/or Gurobi solvers) and displaying the results:

```
#solver = pyomo.opt.SolverFactory('gurobi')
solver = pyomo.opt.SolverFactory('glpk')
results = solver.solve(instance)
results.write()
```

We conclude the implementation description with an important remark. The implementation creates and abstract model of the problem that completely separates the model description from the actual model size and parameter values. That is, the model presented in this section can handle any problem size, with any number of parameters. The actual problem parameters are provided to the implemented model through an AMLP data file (the 'constants2a.dat' file in the description above). We run our experiments for different configurations (suppliers and consumers, distances, etc.) stored in a few AMPL data files but using the same model implementation.

A typical AMPL file content is described in the next section.

3.2 Experimental datasets using AMPL

A Mathematical Programming Language (AMPL [10]) is a modeling language to describe high-complexity problems for large-scale mathematical computing (and in particular large-scale optimization problems). One advantage of AMPL is the similarity of its syntax to the mathematical notation of optimization problems [34]. This allows for a very concise and readable definition of problems in the domain of optimization. According to Wikipedia, AMPL is the most popular format for representing mathematical programming problems.

We created three network configurations corresponding to each of the following scenarios:

1. All suppliers can deliver to all consumers and distances are comparable. However, no supplier has every commodity available on stock.

2. All suppliers can deliver to all consumers but one supplier is located significantly closer to the consumers.

3. Similar to above, but the closer supplier must overcome some higher security costs for delivering to some consumers.

Each configuration was described by a corresponding AMPL file. The results of the experiments are presented in Chapter 4. We will focus next on describing how one model configuration dataset is created using AMPL.

- The number of suppliers and consumers

```
param NS := 2;
param NC := 8;
```

- The set of suppliers and the set of consumers (as in Figure 2.2):

```
set SS := 'C2' 'C3';
set CS := 'IN1M' 'IN1S1' 'IN1S2' 'IN1S3' 'IN1XS1' 'IN1XS2' 'IN1XS3'
```

- The lower/upper number of trucks bounds

```
param m := 5;
param M := 30;
```

- The costs (distances) from each supplier to each consumer:

```
param c: 1 2 3 4 5 6 7 8 :=
1 120 143 183 163 158 195 193 172
2 93 116 156 136 131 168 166 145
;
```

- The overload threshold and factor: the distance above which an overload factor is applied to take into account security detail cost:

```
param Ot := 150;
param Of := 2;
```

- The requested quantities for: ammo, cargo, fuel, water (one value for each consumer in the list)

```
param Qa :=
1  5
2 10.5
3 6.5
4 9.5
5 2.5
```

```
6 7.5
7  9
8 8.5
;
param qa :=
1 0
2 4.5
3  2
4 0
5 0
6 3.5
7  0
8 4.5
;
param Qc :=
1 16
2 9
3 3
4 3.5
5 5
6 8
7 4.5
8 8
;
param qc :=
1 4.1
```

```
2 2
3 2.4
4 2
5 2.5
6 4.4
7 0
8 0
;
param Qf :=
1 88
2 11.2
3 1.5
4 4
5 3
6 3.3
7 3.3
8 3.2
;
param qf :=
1 0
2 3
3 .5
4 0
5 3
6 3
7 3
```

```
8 3
;
param Qw :=
1 159
2 16
3 5
4 26.1
5 3.2
6 1
7 4.4
8 5
;
param qw :=
1 5.1
2 0
3 0
4 13.18
5 1
6 0
7 1
8 0
;
```

We created three configuration files, one for each experimental scenario listed at the beginning of this section. The experimental results for each scenario are described in the next chapter.

CHAPTER 4

EXPERIMENTAL RESULTS

We tested the model using three networks configurations:

1. All suppliers can deliver to all consumers and distances are comparable. However, no supplier has every commodity available on stock. The goal was to test the model solution for a relatively small network.

2. All suppliers can deliver to all consumers but one supplier is located significantly closer to the consumers. We then compare the solution of this experiment with the previous, with the obvious expectation that the closer supplier would deliver the commodities.

3. Similar to above, but the closer supplier must overcome some higher security costs for delivering to some consumers. We wanted to observe how the security constrains factor in and overcome geographical distance costs, so the farther supplier becomes an optimal choice (at least for some commodities).

The experimental results are described in the following.

4.1 EXPERIMENT RESULTS 1

Based on the model we created a computer program using the python language to test whether the constraints we imposed will produce the results we were expecting to receive. For this experiment we ran our scenario for ten days and used a portion of those results to test the viability of the program. In examining Figure 4.1 you can see C2 and C3 are both responsible for resupplying all 8 consumers. However, notice that C2 only delivers to 4 of the consumers. This is primarily due to the fact that the amount requested exceeds the amount of on hand quantity located at C3.

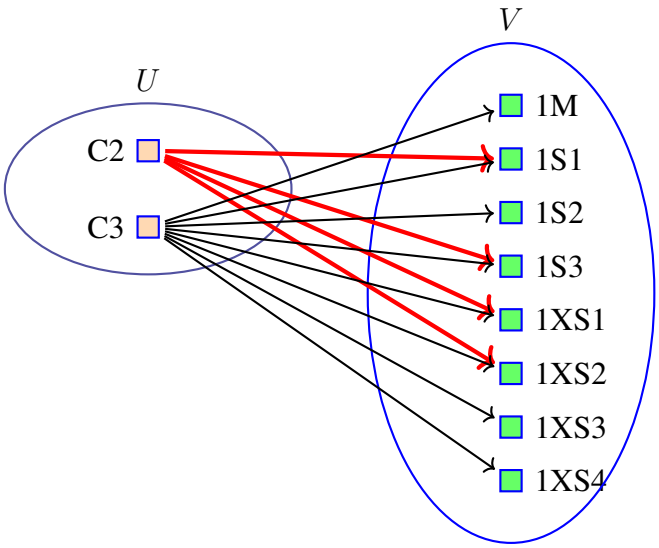

Figure 4.1: Suppliers-consumers network under security constraints (Experiment 1)

Table 4.4 provides a breakdown detailing the number of trucks each supplier will deliver to their respective consumer. In comparing 4.4 and 4.3 you will notice that on certain commodities the suppliers are sharing the delivery duties. For example, look at table 4.7 and 4.8 consumer 4 is being resupplied by both suppliers because neither supplier has enough on hand quantity to fulfill this particular commodity request.

	Supplier C2	
1M	(1, 1)	0.0
1S1	(1, 2)	1.0
1S2	(1, 3)	0.0
1S3	(1, 4)	1.0
1XS1	(1, 5)	1.0
1XS2	(1, 6)	1.0
1XS3	(1, 7)	0.0
1XS4	(1, 8)	0.0

Table 4.1: C2 Supply Delivery Experiment 1

	Supplier C3	
1M	(2, 1)	1.0
1S1	(2, 2)	1.0
1S2	(2, 3)	1.0
1S3	(2, 4)	1.0
1XS1	(2, 5)	1.0
1XS2	(2, 6)	1.0
1XS3	(2, 7)	1.0
1XS4	(2, 8)	1.0

Table 4.2: C3 Supply Delivery Experiment 1

	IN1M	**IN1S1**	**IN1S2**	**IN1S3**	**IN1XS1**	**IN1XS2**	**IN1XS3**	**IN1XS4**
Ammo	22	0	18	28	16	4	0	4
Cargo	8	22	0	18	28	16	4	0
Fuel	8	22	0	18	28	16	4	8
Water	8	22	0	18	28	16	4	5

Table 4.3: Consumer Commodity Request Experiment 1

Table 4.4: Supplier-Consumer Commodity Delivery Experiment 1

Ammo	**Supplier C2**		**Supplier C3**	
1M	(1, 1)	0.0	(2, 1)	22.0
1S1	(1, 2)	0.0	(2, 2)	0.0
1S2	(1, 3)	0.0	(2, 3)	18.0
1S3	(1, 4)	0.0	(2, 4)	28.0
1XS1	(1, 5)	0.0	(2, 5)	16.0
1XS2	(1, 6)	0.0	(2, 6)	4.0
1XS3	(1, 7)	0.0	(2, 7)	0.0
1XS4	(1, 8)	0.0	(2, 8)	4.0

Table 4.5: Ammo Delivery1

Cargo	**Supplier C2**		**Supplier C3**	
1M	(1, 1)	0.0	(2, 1)	8.0
1S1	(1, 2)	0.0	(2, 2)	22.0
1S2	(1, 3)	0.0	(2, 3)	0.0
1S3	(1, 4)	0.0	(2, 4)	18.0
1XS1	(1, 5)	22.0	(2, 5)	6.0
1XS2	(1, 6)	0.0	(2, 6)	16.0
1XS3	(1, 7)	0.0	(2, 7)	4.0
1XS4	(1, 8)	0.0	(2, 8)	40.0

Table 4.6: Cargo Delivery1

Fuel	**Supplier C2**		**Supplier C3**	
1M	(1, 1)	0.0	(2, 1)	8.0
1S1	(1, 2)	0.0	(2, 2)	22.0
1S2	(1, 3)	0.0	(2, 3)	0.0
1S3	(1, 4)	14.0	(2, 4)	4.0
1XS1	(1, 5)	28.0	(2, 5)	0.0
1XS2	(1, 6)	0.0	(2, 6)	16.0
1XS3	(1, 7)	0.0	(2, 7)	4.0
1XS4	(1, 8)	0.0	(2, 8)	8.0

Table 4.7: Fuel Delivery1

Water	**Supplier C2**		**Supplier C3**	
1M	(1, 1)	0.0	(2, 1)	8.0
1S1	(1, 2)	16.0	(2, 2)	6.0
1S2	(1, 3)	0.0	(2, 3)	0.0
1S3	(1, 4)	18.0	(2, 4)	0.0
1XS1	(1, 5)	0.0	(2, 5)	28.0
1XS2	(1, 6)	10.0	(2, 6)	6.0
1XS3	(1, 7)	0.0	(2, 7)	4.0
1XS4	(1, 8)	0.0	(2, 8)	5.0

Table 4.8: Water Delivery1

4.2 Experimental Results 2

As you saw in the previous experiment distance and requested amount affected which supplier would deliver to which consumer. So, for this experiment we wanted to see what would happen if we adjusted the amount of commodity being requested and not change the distance between supplier and consumer. The way the program is designed, there is a penalty imposed if the distance is over 150 miles, to discourage long delivery routes. Thereby providing another mechanism to reduce the amount of time Soldiers are exposed to environment. In examining Figure 4.2 you can see C3 is the only supplier responsible for resupplying all 8 consumers. So, by adjusting the amount requested we can eliminate the need of having both suppliers delivering to all consumers. This is accomplished due to two factors: first, the distance between C2 and the consumers and second, the amount requested does not exceed the amount of commodity C3 has on hand.

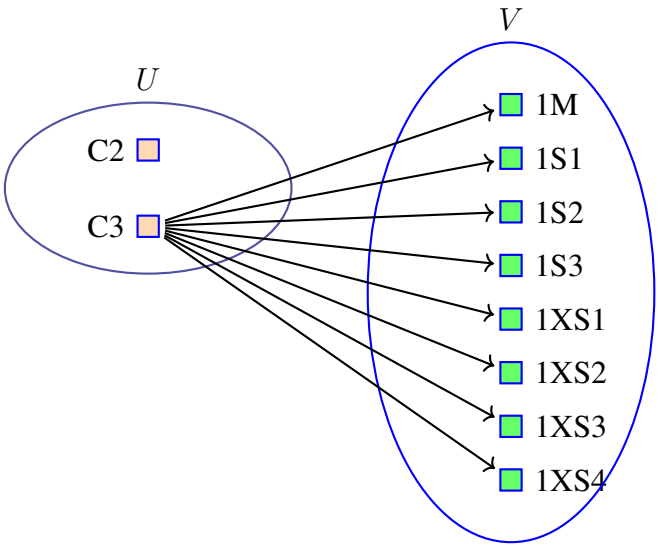

Figure 4.2: Suppliers-consumers network under security constraints (Experiment 2)

	Supplier C2	
1M	(1, 1)	0.0
1S1	(1, 2)	0.0
1S2	(1, 3)	0.0
1S3	(1, 4)	0.0
1XS1	(1, 5)	0.0
1XS2	(1, 6)	0.0
1XS3	(1, 7)	0.0
1XS4	(1, 8)	0.0

Table 4.9: C2 Supply Delivery Experiment 2

	Supplier C3	
1M	(2, 1)	1.0
1S1	(2, 2)	1.0
1S2	(2, 3)	1.0
1S3	(2, 4)	1.0
1XS1	(2, 5)	1.0
1XS2	(2, 6)	1.0
1XS3	(2, 7)	1.0
1XS4	(2, 8)	1.0

Table 4.10: C3 Supply Delivery Experiment 2

Table 4.12 provides a breakdown detailing the number of trucks each supplier will deliver to their respective consumer. Unlike experiment 1 due to the quantities being requested, as laid in table 4.11 supplier C3 is responsible for all deliveries.

	IN1M	**IN1S1**	**IN1S2**	**IN1S3**	**IN1XS1**	**IN1XS2**	**IN1XS3**	**IN1XS4**
Ammo	0	4.5	2	0	0	3.5	0	4.5
Cargo	4.1	2	2.4	2	2.5	4.4	0	0
Fuel	0	3	.5	0	3	3	3	3
Water	5.1	0	0	13.18	1	0	1	0

Table 4.11: Consumer Commodity Request Experiment 2

Table 4.12: Supplier-Consumer Commodity Delivery Experiment 2

Ammo	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	0.0
1S1	(1, 2)	0.0	(2, 2)	4.5
1S2	(1, 3)	0.0	(2, 3)	2.0
1S3	(1, 4)	0.0	(2, 4)	0.0
1XS1	(1, 5)	0.0	(2, 5)	0.0
1XS2	(1, 6)	0.0	(2, 6)	3.5
1XS3	(1, 7)	0.0	(2, 7)	0.0
1XS4	(1, 8)	0.0	(2, 8)	4.5

Table 4.13: Ammo Delivery2

Cargo	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	4.1
1S1	(1, 2)	0.0	(2, 2)	2.0
1S2	(1, 3)	0.0	(2, 3)	2.4
1S3	(1, 4)	0.0	(2, 4)	2.0
1XS1	(1, 5)	0.0	(2, 5)	2.5
1XS2	(1, 6)	0.0	(2, 6)	4.4
1XS3	(1, 7)	0.0	(2, 7)	0.0
1XS4	(1, 8)	0.0	(2, 8)	0.0

Table 4.14: Cargo Delivery2

Fuel	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	0.0
1S1	(1, 2)	0.0	(2, 2)	3.0
1S2	(1, 3)	0.0	(2, 3)	0.5
1S3	(1, 4)	0.0	(2, 4)	4.0
1XS1	(1, 5)	0.0	(2, 5)	3.0
1XS2	(1, 6)	0.0	(2, 6)	3.0
1XS3	(1, 7)	0.0	(2, 7)	3.0
1XS4	(1, 8)	0.0	(2, 8)	3.0

Table 4.15: Fuel Delivery2

Water	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	5.1
1S1	(1, 2)	0.0	(2, 2)	0.0
1S2	(1, 3)	0.0	(2, 3)	0.0
1S3	(1, 4)	0.0	(2, 4)	13.18
1XS1	(1, 5)	0.0	(2, 5)	1.0
1XS2	(1, 6)	0.0	(2, 6)	6.0
1XS3	(1, 7)	0.0	(2, 7)	1.0
1XS4	(1, 8)	0.0	(2, 8)	0.0

Table 4.16: Water Delivery2

4.3 Experimental Results 3

So far, we have seen how distance and quantity can dictate how consumers will be re-supplied. However, we have not shown the impact that security can have on the delivery

routes. This experiment will use the same base information from experiment 2, the one key difference will be changing the security along some of the routes and see how this change will impact which supplier gets used.

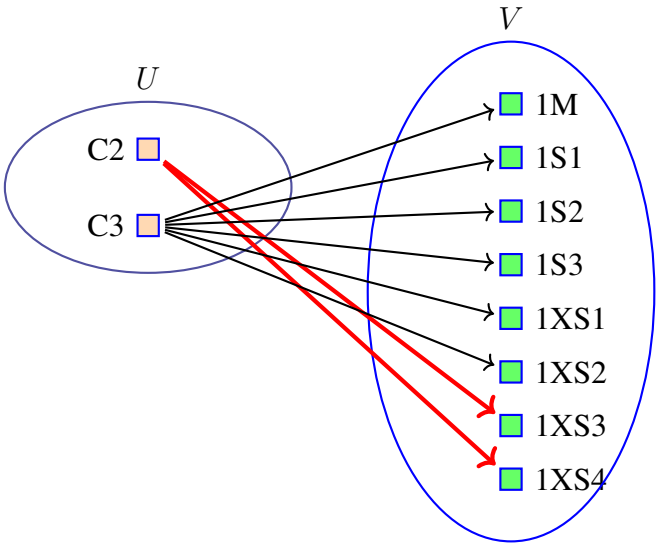

Figure 4.3: Suppliers-consumers network under security constraints (Experiment 3)

Table 4.19 shows the related security cost in terms of distance. The security cost is a factor applied to the distance of the route. For this scenario we changed the threat assessment along the edges connecting C3 to 7 and C3 to 8 from normal to moderate. Therefore a factor of 4 is applied to this edge. By doing this it increases the transportation cost along this route and therefore this supplier-consumer route is eliminated from the equation resulting in supplier C2 being responsible for these deliveries.

	Supplier C2	
1M	(1, 1)	0.0
1S1	(1, 2)	0.0
1S2	(1, 3)	0.0
1S3	(1, 4)	0.0
1XS1	(1, 5)	0.0
1XS2	(1, 6)	0.0
1XS3	(1, 7)	1.0
1XS4	(1, 8)	1.0

Table 4.17: C2 Supply Delivery Experiment 3

	Supplier C3	
1M	(2, 1)	1.0
1S1	(2, 2)	1.0
1S2	(2, 3)	1.0
1S3	(2, 4)	1.0
1XS1	(2, 5)	1.0
1XS2	(2, 6)	1.0
1XS3	(2, 7)	0.0
1XS4	(2, 8)	0.0

Table 4.18: C3 Supply Delivery Experiment 3

	IN1M	**IN1S1**	**IN1S2**	**IN1S3**	**IN1XS1**	**IN1XS2**	**IN1XS3**	**IN1XS4**
C2	120	143	183	163	158	195	193	172
C3	93	116	156	136	131	168	664	580

Table 4.19: Security Cost

Table 4.20: Supplier-Consumer Commodity Delivery Experiment 3

Ammo	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	0.0
1S1	(1, 2)	0.0	(2, 2)	4.5
1S2	(1, 3)	0.0	(2, 3)	2.0
1S3	(1, 4)	0.0	(2, 4)	0.0
1XS1	(1, 5)	0.0	(2, 5)	0.0
1XS2	(1, 6)	0.0	(2, 6)	3.5
1XS3	(1, 7)	0.0	(2, 7)	0.0
1XS4	(1, 8)	4.5	(2, 8)	0.0

Table 4.21: Ammo Delivery3

Cargo	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	4.1
1S1	(1, 2)	0.0	(2, 2)	2.0
1S2	(1, 3)	0.0	(2, 3)	2.4
1S3	(1, 4)	0.0	(2, 4)	2.0
1XS1	(1, 5)	0.0	(2, 5)	2.5
1XS2	(1, 6)	0.0	(2, 6)	4.4
1XS3	(1, 7)	0.0	(2, 7)	0.0
1XS4	(1, 8)	0.0	(2, 8)	0.0

Table 4.22: Cargo Delivery3

Fuel	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	0.0
1S1	(1, 2)	0.0	(2, 2)	3.0
1S2	(1, 3)	0.0	(2, 3)	0.5
1S3	(1, 4)	0.0	(2, 4)	4.0
1XS1	(1, 5)	0.0	(2, 5)	3.0
1XS2	(1, 6)	0.0	(2, 6)	3.0
1XS3	(1, 7)	3.0	(2, 7)	0.0
1XS4	(1, 8)	3.0	(2, 8)	0.0

Table 4.23: Fuel Delivery3

Water	Supplier C2		Supplier C3	
1M	(1, 1)	0.0	(2, 1)	5.1
1S1	(1, 2)	0.0	(2, 2)	0.0
1S2	(1, 3)	0.0	(2, 3)	0.0
1S3	(1, 4)	0.0	(2, 4)	13.18
1XS1	(1, 5)	0.0	(2, 5)	1.0
1XS2	(1, 6)	0.0	(2, 6)	6.0
1XS3	(1, 7)	1.0	(2, 7)	0.0
1XS4	(1, 8)	0.0	(2, 8)	0.0

Table 4.24: Water Delivery3

CHAPTER 5

CONCLUSION

Due to the increased interest in vehicle routing problems over the past 80 years several variations has arisen to address issues associated with the varying industries in which they are employed. The capacitated vehicle routing problem (CVRP) and set partitioning vehicle routing problem (SPVRP) are two of those variations and the basis upon which we built our military vehicle routing problem. By using the model for CVRP we were able to model our problem which would compute route optimality from supplier to consumer. This model was plagued by 5 crucial shortcomings as laid out in detail in chapter 2. To mitigate these shortcomings, first we reclassified the problem as a SPVRP and secondly, created a 2-step process to calculate the routes. This allowed us to incorporate security as part of the solution and partition our suppliers into one set and our consumers into another set. Effectively changing our model from an undirected graph to a bipartite graph. Due to these changes in the model we were able to address our multiple supplier scenario and additional factors. In our particular multiple supplier scenario, our suppliers do not carry all of the required commodities. This was done to address the possibility of a commodity shortage at one supplier but still needing the capability to support all consumers with the requested amount of commodities. Through experiment 1 you were able to see how the suppliers ($C2$, $C3$) shared the responsibility of delivering supplies to their 8 consumers. This was because distance and on hand quantity forced both suppliers to be utilized. However, in experiment 2 you were able to see only one supplier was utilized this is primarily due to distance from consumers. We imposed a penalty on distances greater than 150 miles to discourage long deliveries routes. Since, supplier $C3$ was able to handle the quantities requested there was no reason to include supplier $C2$.

5.1 Future Research

The application of this model is to be used in a vendor managed inventory (VMI) environment. This thesis only addressed the vehicle routing aspect of this environment. In Chapter 1 we reviewed the Inventory Routing Problem (IRP) to provide a full understanding of the VMI environment as a whole. So, future work would include modeling the IRP and providing a solution. The IRP will address how often, how much and when will the resupply take place.

Another possibility for future research would be in a military context. This current model could potentially be used to identify unit augmentations to logistics companies. Currently, security (outside the normal requirements) is not an aspect that logistics companies have to worry about along delivery routes. Because the battlespace owner secures everything within their span of control however as in our problem this was not the case. Therefore, forcing the supplier to supply their own security assets. In practice this cannot be sustained because the suppliers are not equipped with trained personnel to handle security as a full-time job. So, if this becomes the norm it would be essential to augment logistics companies with a security element. By utilizing this model it will help identify how the manning requirements for that security element.

REFERENCES

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network flows*, Prentice-Hall, New Jersey, 1993.

[2] Steven F. Baker, David P. Morton, Richard E. Rosenthal, and Laura Melody Williams, *Optimizing military airlift*, Operations Research **50** (2002), no. 4, 582–602.

[3] Moshe Dror; Michael Ball, *Inventory/routing: Reduction from an annual to a short-period problem*, Naval Research Logistics (NRL) **34** (1987).

[4] M. Grazia Bertazzi, Luca; Speranza, *Inventory routing problems: an introduction*, EURO Journal of Transportation and Logistics **1** (2012).

[5] Shoshana Anily; Julien Bramel, *A probabilistic analysis of a fixed partition policy for the inventory-routing problem*, Naval Research Logistics (NRL) **51** (2004).

[6] G.G. Brown, W.M. Carlyle, R.F. Dell, and J.W. Brau, *Optimizing intratheater military airlift in iraq and afghanistan*, Military Operations Research **18** (2013), no. 3, 35–52.

[7] Partha Chakroborty, *Optimal Routing and Scheduling in Transportation: Using Genetic Algorithm to Solve Difficult Optimization Problems*, Direction IITK Newsletter **6** (2004), 29–40.

[8] Jianxiang Li; Haoxun Chen; Feng Chu, *Performance evaluation of distribution strategies for the inventory routing problem*, European Journal of Operational Research **202** (2010).

[9] Awi Federgruen, *[handbooks in operations research and management science] network routing volume 8 —— chapter 4 analysis of vehicle routing and inventory-routing problems*, 1995.

[10] Robert Fourer and Brian W Kernighan, *Ampl: A modeling language for mathematical programming*, Duxbury Press, 2002.

[11] GNU, *GNU Linear Programming Kit, Version 4.65*, 2018, http://www.gnu.org/software/glpk/glpk.html.

[12] William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola, *Pyomo–optimization modeling in python*, second ed., vol. 67, Springer Science & Business Media, 2017.

[13] William E Hart, Jean-Paul Watson, and David L Woodruff, *Pyomo: modeling and solving mathematical programs in python*, Mathematical Programming Computation **3** (2011), no. 3, 219–260.

[14] Gurobi Optimization Inc, *Gurobi Optimizer Reference Manual*, 2014, http://www.gurobi.com.

[15] Dr. Gerald G. Brown Dr. E. Matthew Carlyle John W. Brau, Jr and Dr. Robert F. Dell, *Optimizing Intra theater Military Airlift in Iraq and Afghanistan*, Military Operations Research **18** (2013), no. 3, 35—-52.

[16] Richard E. Killblane, *Convoy ambush case studies*, **Volume II: Iraq and Afghanistan** (2015), 274–276.

[17] CPT McCormack, Ian M., *The Military Inventory Routing Problem with Direct Delivery*, March 2014.

[18] P. Munari, T. Dollevoet, and R. Spliet, *A generalized formulation for vehicle routing problems*, ArXiv e-prints (2016).

[19] Department of the Army, *Technical Manual 9-2320-366-10-1 M1083 Series 5 ton 6x6, Medium Tactical Vehicles (MTV)*, 15 September 1998.

[20] _____, *Technical Manual 9-2320-365-10 M1078 Series 2.5 ton 4x4, Light Medium Tactical Vehicles (LMTV)*, 20 August 2005.

[21] _____, *Technical Manual 9-2330-326-14p Flat Bed Semitrailer, Break Bulk*, 20 May 2003.

[22] _____, *Field Manual 4-01.45(05) Tactical Convoy Operations*, 24 March 2005.

[23] _____, *Technical Manual 9-2330-386-14p Flat Bed Semitrailer, Break Bulk*, 28 September 1990.

[24] _____, *Army Techniques Publication 4-93 The Sustainment Brigade*, 9 August 2013.

[25] _____, *Technical Manual 9-2330-359-14p Flat Bed Semitrailer, Break Bulk*, August 1991.

[26] _____, *Technical Manual 9-2320-364-10 Truck, Tractor, M1074 and M1075 Palletized Load System (PLS)*, August 1999.

[27] _____, *Technical Manual 9-2330-385-14 Palletized Load System (PLS) Trailer*, August 1999.

[28] _____, *Technical Manual 9-2330-358-14p Flat Bed Semitrailer, Break Bulk*, December 1987.

[29] _____, *Technical Manual 9-2320-279-10-1 M977 Series 8x8, Heavy Expanded Mobility Tactical Trucks (HEMTT)*, November 1986.

[30] _____, *Technical Manual 9-2320-356-14 Semitrailer, Tank, 5000 Gallon, Bulk Haul*, October 1990.

[31] Martin Savelsbergh; Jin-Hwa Song, *An optimization algorithm for the inventory routing problem with continuous moves*, Computers & Operations Research **35** (2008).

[32] Daniele Toth, Paolo; Vigo, *The vehicle routing problem —— 12. inventory routing in practice*, vol. 10.1137/1.9780898718515, 2002.

[33] Paolo Toth and Daniele Vigo, *Vehicle routing:problems, methods, and applications*, 2nd ed., Society for Industrial and Applied Mathematics and the Mathematical Optimization Society, 2014.

[34] Wikipedia, *AMPL*, 2018, https://en.wikipedia.org/wiki/AMPL.

[35] Lei Yang, Xianfeng; Feng, *Inventory routing problem*, Transportation Research Record Journal of the Transportation Research Board **2378** (2013).

Appendix A

REORDER INFORMATION

Tables in this appendix were utilized in determining the quantities to be used in our experiments. While the tables represents the needs of the consumer after 10 days have lapsed you can easily adjust these figures to represent a different time-frame. These tables are presented to give an understanding of the consumers consumption over any planning horizon.

Table A.1 gives the threshold limits by commodity and unit. There is a minimum and maximum threshold limit. This is what sets the minimum and maximum quantity amounts a unit receive.

Unit	**Threshold Limits**							
	Cargo		**Water**		**Fuel**		**Ammo**	
	Min	Max	Min	Max	Min	Max	Min	Max
IN1M	0.60	0.90	0.69	0.99	0.65	0.95	0.60	0.90
IN1S1	0.50	0.75	0.65	0.95	0.60	0.90	0.50	0.75
IN1S2	0.50	0.75	0.63	0.93	0.60	0.90	0.50	0.75
IN1S3	0.50	0.75	0.66	0.96	0.60	0.90	0.50	0.75
IN1XS1	0.50	0.75	0.60	0.90	0.60	0.90	0.50	0.75
IN1XS2	0.50	0.75	0.63	0.93	0.60	0.90	0.50	0.75
IN1XS3	0.50	0.75	0.61	0.91	0.60	0.90	0.50	0.75
IN1XS4	0.50	0.75	0.59	0.89	0.60	0.90	0.50	0.75

Table A.1: Threshold Limits

Table A.2 provides a breakdown of the minimum and maximum truck requirement based on the amount goods a unit request. The quantity requested is translated into truck requirements.

Unit	Truck Equivalent for Reorder*							
	Cargo		Water		Fuel		Ammo	
	Min	Max	Min	Max	Min	Max	Min	Max
IN1M	16.32132	4.080329	158.9888653	5.128673075	87.61767008	12.51681001	0.83324169	0.208310423
IN1S1	1.678358	0.839179	15.54010763	2.220015375	11.19385244	2.798463111	0.04712906	0.023564528
IN1S2	2.890612	1.445306	26.30988878	4.977546525	1.469855677	0.367463919	0.12567748	0.06283874
IN1S3	3.304878	1.652439	26.13386455	3.0745723	15.457488	3.864372	0.14531459	0.072657293
IN1XS1	0.479923	0.239961	3.170023	0.79250575	2.663128889	0.665782222	0.03770324	0.018851622
IN1XS2	0.64872	0.32436	4.172048775	0.789306525	3.362195556	0.840548889	0.00942581	0.004712906
IN1XS3	0.64872	0.32436	4.397564925	1.014822675	3.362195556	0.840548889	0.00942581	0.004712906
IN1XS4	0.64872	0.32436	4.623081075	1.240338825	3.148862222	0.787215556	0.00942581	0.004712906

Table A.2: Min-Max Truck Requirement

To determine the number of trucks required to deliver the reorder point figures:

- Cargo was calculated with a truck that has a capacity of 49,500lbs
- Fuel was calculated with a truck that has a capacity of 4,500 gallons
- Ammo was calculated with a truck that has a capacity of 12,731lbs
- Water was calculated with a truck that has a capacity of 4,000 gallons

Formula:

$$\frac{(storagecapacity - reorderpoint)}{truckcapacity}$$

Table A.3 gives the quantity which will trigger a resupply. Just as in the previous table there is a minimum and maximum requirement as well. This information will be used in the IRP solution.

Unit	Reorder Point							
	Cargo		Water		Fuel		Ammo	
	Min	Max	Min	Max	Min	Max	Min	Max
IN1M	1,211,857.79	1,817,786.68	1,415,513.77	2,030,954.54	732,233.39	1,070,187.26	15,912.00	23,868.00
IN1S1	83,078.70	124,618.05	115,440.80	168,721.17	75,558.50	113,337.76	600.00	900.00
IN1S2	143,085.30	214,627.95	179,191.67	264,521.04	109,136.78	163,705.18	1,600.00	2,400.00
IN1S3	163,591.45	245,387.17	202,921.77	295,158.94	104,338.04	156,507.07	1,850.00	2,775.00
IN1XS1	23,756.18	35,634.27	19,020.14	28,530.21	17,976.12	26,964.18	480.00	720.00
IN1XS2	32,111.66	48,167.49	28,415.03	41,946.00	22,694.82	34,042.23	120.00	180.00
IN1XS3	32,111.66	48,167.49	27,512.97	41,043.94	22,694.82	34,042.23	120.00	180.00
IN1XS4	32,111.66	48,167.49	26,610.91	40,141.87	21,254.82	31,882.23	120.00	180.00

Table A.3: Reorder Point

Table A.4 provides a time frame for how often a units reorder point is reached. This will help gauge when a resupply operation should occur (forecasting). This information will be used in the IRP solution. The numbers in the table represent days, to get the hours

Unit	How often Reorder Point is reached?*							
	Cargo		Water		Fuel		Ammo	
	Min	Max	Min	Max	Min	Max	Min	Max
IN1M	4.7084627	1.17711567	4.29464739	0.138537	10.17115	1.453022	4.494916	1.123729
IN1S1	5.67633765	2.83816882	5.40189998	0.7717	10.82058	2.705145	7.500069	3.750034
IN1S2	6.15021242	3.07510621	5.95758249	1.1271102	11.93345	2.983362	5.333333	2.666667
IN1S3	5.44537831	2.72268916	4.63552058	0.5453554	8.837226	2.209307	5.138889	2.569444
IN1XS1	6.00014316	3.00007158	4.85492403	1.213731	9.270687	2.317672	6.000041	3.000021
IN1XS2	6.00012519	3.0000626	4.72437129	0.8938	10.06953	2.517382	6.000046	3.000023
IN1XS3	6.00012494	3.00006247	4.97977039	1.1491778	9.578748	2.394687	6.00006	3.00003
IN1XS4	6.0001216	3.0000608	5.23511413	1.4045428	9.043116	2.260779	6.00005	3.000025

Table A.4: Delivery Frequency

equivalent multiply by 24. Formula:

$$\frac{(capacity - reorderpoint)}{consumption}$$

.

Table A.5 provides the hourly consumption rate by commodity. This information was already pre-determined and provided in our data set.

Unit	Hourly Consumption by Good			
	Cargo	**Water**	**Ammo**	**Fuel**
IN1M	7,149.41	6,496.49	98.33	2,655.38
IN1S1	609.83	560.42	3.33	322.66
IN1S2	969.38	838.19	12.5	422.45
IN1S3	1251.76	1,067.52	15	534.03
IN1XS1	164.97	123.94	3.33	94.52
IN1XS2	222.99	166.68	.83	112.50
IN1XS3	222.99	166.68	.83	116.38
IN1XS4	222.99	166.68	.83	123.05

Table A.5: Hourly Consumption

Table A.6 provides the consumption by commodity per 10 days. This was calculated by multiplying the hourly consumption rate by 24.

Unit	10 Day Consumption by Good			
	Cargo	**Water**	**Ammo**	**Fuel**
IN1M	171,585.77	148,080.95	2,360.00	38,764.49
IN1S1	14,635.97	11,507.14	80.00	4,655.23
IN1S2	23,265.10	17,664.81	300.00	6,096.97
IN1S3	30,042.26	22,550.96	360.00	7,871.10
IN1XS1	3,959.27	2,611.80	80.00	1,292.69
IN1XS2	5,351.83	3,532.36	20.00	1,502.54
IN1XS3	5,351.83	3,532.34	20.00	1,579.53
IN1XS4	5,351.83	3,532.36	20.00	1,566.92

Table A.6: 10 Day Consumption

Appendix B

TRUCK CHARACTERISTICS

Table B.1 provides the characteristics of the logistics in the fleet.

Nomenclature	*Model*	*Fuel Capacity (gallons)*	*Length (inches)*	*Width (inches)*	*Pallets (40”x48”)*	*Maximum Load (Cargo/Fuel)*
HEMTT Truck	M977/985	155	401	102	8	11.00 ST
MTV	M1083	155	168	88	6	11.00 ST
LMTV	M1078	155	147	88	6	11.00 ST
HEMTT Fuel Tanker 2.5K	M978	155	401	96	NA	2250 GAL
HEMTT Fuel Tanker 5K	M969	155	401	96	NA	4750 GAL
Palletized Loading System w/Flatrack	M1074 M1075	155	431	102	10	11.00 ST
Loading Handling System w/Flatrack	M1120	155	419	96	20	11.00 ST
22.5-ton Trailer	M871A1	NA	358	96	14	22.50 ST
	M871A2		374	96	13	22.50 ST
	M871A3		495	96	13	22.50 ST
34-ton Trailer	M872	NA	490	96	18	34.00 ST
16.5-ton Trailer PLS w/Flatrack	M1076	NA	309	96	18	16.50 ST

Table B.1: Truck Capacity By Type [29][20][26][30]

Appendix C

PYTHON CODE

C.1 EXPERIMENT 1

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 31 05:48:31 2017

@author: T.Fletcher
"""

# Import
import pyomo
import pyomo.opt
import pyomo.environ as pe
import os

#problem name
PROBLEM_NAME = "Minimal cost supplies delivery"



# Creation of an Abstract Model
model = pe.AbstractModel(doc=PROBLEM_NAME)

##########################
#general parameters: all read from a data file
##########################
#number of suppliers and consumers
model.NS = pe.Param(within=pe.PositiveIntegers)
model.NC = pe.Param(within=pe.PositiveIntegers)
```

```
#indexes over suppliers/consumers respectively
model.I = pe.RangeSet(1, model.NS)
model.J = pe.RangeSet(1, model.NC)

## Define sets ##
#  Sets
#       model.SS   Suppliers set
#       model.CS   Consummers set
model.SS = pe.Set(doc='Suppliers')
model.CS = pe.Set(doc='Consummers')

#lower/upper bounds for quantities
model.m = pe.Param()
model.M = pe.Param()

#cost (distance) along each route
model.c = pe.Param(model.I, model.J)

#overload cost (security) parameters
model.Ot = pe.Param()
model.Of = pe.Param()
def O_init(model, i, j):
    #
    # Create the value of model.o[i,j]
    #
    v = 0.0
    if model.c[i,j] >= model.Ot:
        v = model.c[i,j] * model.Of
    return v
model.o = pe.Param(model.I, model.J, initialize=O_init)
```

```
#requested quantities
model.qa = pe.Param(model.J)
model.qc = pe.Param(model.J)
model.qf = pe.Param(model.J)
model.qw = pe.Param(model.J)


###################
#  Variables
################
#quantities (number of trucks) for each commodity
model.xa = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers) #ammo
model.xc = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers) #cargo
model.xf = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers) #fuel
model.xw = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers) #water

#binary variables
model.d = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers,
                 bounds = (0,1))


############################
# Constraints
############################
def supply_ruleA(model, j):
  return sum(model.xa[i,j] for i in model.I) >= model.qa[j]
model.supplyA = pe.Constraint(model.J, rule=supply_ruleA,
                          doc='Supply requested ammo quantities for j')

def supply_ruleC(model, j):
  return sum(model.xc[i,j] for i in model.I) >= model.qc[j]
```

```
model.supplyC = pe.Constraint(model.J, rule=supply_ruleC,
                          doc='Supply requested cargo quantities for j')

def supply_ruleF(model, j):
  return sum(model.xf[i,j] for i in model.I) >= model.qf[j]
model.supplyF = pe.Constraint(model.J, rule=supply_ruleF,
                          doc='Supply requested fuel quantities for j')

def supply_ruleW(model, j):
  return sum(model.xw[i,j] for i in model.I) >= model.qw[j]
model.supplyW = pe.Constraint(model.J, rule=supply_ruleW,
                          doc='Supply requested water quantities for j')


#constraints for binary variables
#quantities lower bound
def supply_minq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
                                              >= model.d[i,j]*model.m
model.supplymin = pe.Constraint(model.I, model.J, rule=supply_minq,
doc='Minimum quantity from supplier i to consumer j')

#quantities upper bound
def supply_maxq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
   <= model.d[i,j]*model.M
model.supplymax = pe.Constraint(model.I, model.J, rule=supply_maxq,
doc='Maximum quantity from supplier i to consumer j')


################################################
```

```
## Define Objective function ##
#  minimize cost   C =  sum((i,j), c(i,j)*(xa(i,j) + xc(i,j) + xf(i,j)
                                          + xw(i,j)) + d(i,j)*o(i,j));
def objective_rule(model):
  return sum(model.c[i,j]*(model.xa[i,j] + model.xc[i,j]
             + model.xf[i,j] + model.xw[i,j])
             + model.d[i,j]*model.o[i,j]
             for i in model.I for j in model.J)

model.objective = pe.Objective(rule=objective_rule, sense=pe.minimize,
                               doc='The objective function')

##read these parameters from a data file (relative to working directory)
cwdir = os.getcwd() #the current working directory must be set to the script
location
datafile = os.path.join(os.sep, cwdir + '\\data\\constants1.dat')
instance = model.create_instance(datafile)
instance.pprint()



"""Solve the model."""
#solver = pyomo.opt.SolverFactory('gurobi')
#results = solver.solve(instance, tee=True, keepfiles=False,
options_string="mip_tolerances_integrality=1e-9 mip_tolerances_mipgap=0")
solver = pyomo.opt.SolverFactory('glpk')
results = solver.solve(instance)
results.write()
instance.solutions.load_from(results)

for v in instance.component_objects(pe.Var, active=True):
```

```
    print ("Variable",str(v))
    varobject = getattr(instance, str(v))
    for index in varobject:
        print ("   ",index, varobject[index].value)
```

C.2 EXPERIMENT 1 DATA FILE

```
##
## Data set for Scenario 1
##

#the number of suppliers and consumers
param NS := 2;
param NC := 8;

#the set of suppliers
set SS := 'C2' 'C3';
#the set of consummers
set CS := 'IN1M' 'IN1S1' 'IN1S2' 'IN1S3' 'IN1XS1' 'IN1XS2' 'IN1XS3' 'IN1XS4';

#lower/upper number of trucks bounds
param m := 10;
param M := 50;


#costs (distances) from each supplier to each consummer
param c: 1 2 3 4 5 6 7 8 :=
1 120 143 183 163 158 195 193 172
2 93 116 156 136 131 168 166 145
;
```

```
#overload threshold: the distance above which an overload factor is applied
param Ot := 150;
#the overload factor: the distance-multiplier when overload is applied
param Of := 2;

#requested quantities for: ammo, cargo, fuel, water
#(one value for each consumer in the list)
param qa :=
1 22
2 0
3 18
4 28
5 16
6 4
7 0
8 4
;
param qc :=
1 8
2 22
3 0
4 18
5 28
6 16
7 4
8 0
;
param qf :=
1 8
2 22
3 0
```

```
4 18
5 28
6 16
7 4
8 8
;
param qw :=
1 8
2 22
3 0
4 18
5 28
6 16
7 4
8 5
;
```

C.3 RESULTS EXPERIMENT 1

```
11 Set Declarations
    CS : Consummers
        Dim=0, Dimen=1, Size=8, Domain=None, Ordered=False, Bounds=None
        ['IN1M', 'IN1S1', 'IN1S2', 'IN1S3', 'IN1XS1', 'IN1XS2', 'IN1XS3', 'IN1XS4']
    SS : Suppliers
        Dim=0, Dimen=1, Size=2, Domain=None, Ordered=False, Bounds=None
        ['C2', 'C3']
    c_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    d_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    o_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
```

```
    supplymax_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    supplymin_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xa_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xc_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xf_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xw_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual

2 RangeSet Declarations
    I : Dim=0, Dimen=1, Size=2, Domain=Integers, Ordered=True, Bounds=(1, 2)
        Virtual
    J : Dim=0, Dimen=1, Size=8, Domain=Integers, Ordered=True, Bounds=(1, 8)
        Virtual

12 Param Declarations
    M : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
        Key  : Value
        None :    50
    NC : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
        Key  : Value
        None :     8
    NS : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
        Key  : Value
        None :     2
    Of : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
        Key  : Value
```

```
    None :     2
Ot : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :   150
c : Size=16, Index=c_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   120
    (1, 2) :   143
    (1, 3) :   183
    (1, 4) :   163
    (1, 5) :   158
    (1, 6) :   195
    (1, 7) :   193
    (1, 8) :   172
    (2, 1) :    93
    (2, 2) :   116
    (2, 3) :   156
    (2, 4) :   136
    (2, 5) :   131
    (2, 6) :   168
    (2, 7) :   166
    (2, 8) :   145
m : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :    10
o : Size=16, Index=o_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   0.0
    (1, 2) :   0.0
    (1, 3) :   366
    (1, 4) :   326
```

```
    (1, 5) :   316
    (1, 6) :   390
    (1, 7) :   386
    (1, 8) :   344
    (2, 1) :   0.0
    (2, 2) :   0.0
    (2, 3) :   312
    (2, 4) :   0.0
    (2, 5) :   0.0
    (2, 6) :   336
    (2, 7) :   332
    (2, 8) :   0.0
qa : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    22
      2 :     0
      3 :    18
      4 :    28
      5 :    16
      6 :     4
      7 :     0
      8 :     4
qc : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :     8
      2 :    22
      3 :     0
      4 :    18
      5 :    28
      6 :    16
      7 :     4
```

```
      8 :     0
    qf : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
        Key : Value
          1 :     8
          2 :    22
          3 :     0
          4 :    18
          5 :    28
          6 :    16
          7 :     4
          8 :     8
    qw : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
        Key : Value
          1 :     8
          2 :    22
          3 :     0
          4 :    18
          5 :    28
          6 :    16
          7 :     4
          8 :     5

5 Var Declarations
    d : Size=16, Index=d_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
```

```
    (1, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (1, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    (2, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
xa : Size=16, Index=xa_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
xc : Size=16, Index=xc_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
```

```
    (1, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
xf : Size=16, Index=xf_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (1, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
    (2, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
```

```
        (2, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
    xw : Size=16, Index=xw_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (1, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 1) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 2) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 3) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 4) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 5) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 6) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 7) :     0 :  None :  None : False :  True : NonNegativeIntegers
        (2, 8) :     0 :  None :  None : False :  True : NonNegativeIntegers

1 Objective Declarations
    objective : The objective function
        Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : minimize : 120*(xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1])
        + 143*(xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2]) + 183*(xa[1,3] + xc[1,3]
        + xf[1,3] + xw[1,3]) + 366*d[1,3] + 163*(xa[1,4] + xc[1,4] + xf[1,4] +
        xw[1,4]) + 326*d[1,4] + 158*(xa[1,5] + xc[1,5] + xf[1,5] + xw[1,5])
        + 316*d[1,5] + 195*(xa[1,6] + xc[1,6] + xf[1,6] + xw[1,6]) + 390*d[1,6]
        + 193*(xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7]) + 386*d[1,7] + 172*( xa[1,8]
```

```
        + xc[1,8] + xf[1,8] + xw[1,8]) + 344*d[1,8] + 93*(xa[2,1] + xc[2,1]
        + xf[2,1] + xw[2,1]) + 116*(xa[2,2] + xc[2,2] + xf[2,2] + xw[2,2])
        + 156*(xa[2,3] + xc[2,3] + xf[2,3] + xw[2,3]) + 312*d[2,3] + 136*(xa[2,4]
        + xc[2,4] + xf[2,4] + xw[2,4]) + 131*(xa[2,5] + xc[2,5] + xf[2,5] + xw[2,5])
        + 168*(xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6]) + 336*d[2,6] + 166*(xa[2,7]
        + xc[2,7] + xf[2,7] + xw[2,7]) + 332*d[2,7] + 145*(xa[2,8] + xc[2,8] + xf[2,8]
        + xw[2,8])

6 Constraint Declarations
    supplyA : Supply requested ammo quantities for j
        Size=8, Index=J, Active=True
        Key : Lower : Body              : Upper : Active
          1 :  22.0 : xa[1,1] + xa[2,1] :  +Inf :   True
          2 :   0.0 : xa[1,2] + xa[2,2] :  +Inf :   True
          3 :  18.0 : xa[1,3] + xa[2,3] :  +Inf :   True
          4 :  28.0 : xa[1,4] + xa[2,4] :  +Inf :   True
          5 :  16.0 : xa[1,5] + xa[2,5] :  +Inf :   True
          6 :   4.0 : xa[1,6] + xa[2,6] :  +Inf :   True
          7 :   0.0 : xa[1,7] + xa[2,7] :  +Inf :   True
          8 :   4.0 : xa[1,8] + xa[2,8] :  +Inf :   True
    supplyC : Supply requested cargo quantities for j
        Size=8, Index=J, Active=True
        Key : Lower : Body              : Upper : Active
          1 :   8.0 : xc[1,1] + xc[2,1] :  +Inf :   True
          2 :  22.0 : xc[1,2] + xc[2,2] :  +Inf :   True
          3 :   0.0 : xc[1,3] + xc[2,3] :  +Inf :   True
          4 :  18.0 : xc[1,4] + xc[2,4] :  +Inf :   True
          5 :  28.0 : xc[1,5] + xc[2,5] :  +Inf :   True
          6 :  16.0 : xc[1,6] + xc[2,6] :  +Inf :   True
          7 :   4.0 : xc[1,7] + xc[2,7] :  +Inf :   True
          8 :   0.0 : xc[1,8] + xc[2,8] :  +Inf :   True
```

```
supplyF : Supply requested fuel quantities for j
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   8.0 : xf[1,1] + xf[2,1] :  +Inf :   True
      2 :  22.0 : xf[1,2] + xf[2,2] :  +Inf :   True
      3 :   0.0 : xf[1,3] + xf[2,3] :  +Inf :   True
      4 :  18.0 : xf[1,4] + xf[2,4] :  +Inf :   True
      5 :  28.0 : xf[1,5] + xf[2,5] :  +Inf :   True
      6 :  16.0 : xf[1,6] + xf[2,6] :  +Inf :   True
      7 :   4.0 : xf[1,7] + xf[2,7] :  +Inf :   True
      8 :   8.0 : xf[1,8] + xf[2,8] :  +Inf :   True
supplyW : Supply requested water quantities for j
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   8.0 : xw[1,1] + xw[2,1] :  +Inf :   True
      2 :  22.0 : xw[1,2] + xw[2,2] :  +Inf :   True
      3 :   0.0 : xw[1,3] + xw[2,3] :  +Inf :   True
      4 :  18.0 : xw[1,4] + xw[2,4] :  +Inf :   True
      5 :  28.0 : xw[1,5] + xw[2,5] :  +Inf :   True
      6 :  16.0 : xw[1,6] + xw[2,6] :  +Inf :   True
      7 :   4.0 : xw[1,7] + xw[2,7] :  +Inf :   True
      8 :   5.0 : xw[1,8] + xw[2,8] :  +Inf :   True
supplymax : Maximum quantity from supplier i to consumer j
    Size=16, Index=supplymax_index, Active=True
    Key : Lower : Body : Upper : Active
    (1, 1) : -Inf : xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1] - 50*d[1,1] : 0.0 : True
    (1, 2) : -Inf : xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2] - 50*d[1,2] : 0.0 : True
    (1, 3) : -Inf : xa[1,3] + xc[1,3] + xf[1,3] + xw[1,3] - 50*d[1,3] : 0.0 : True
    (1, 4) : -Inf : xa[1,4] + xc[1,4] + xf[1,4] + xw[1,4] - 50*d[1,4] : 0.0 : True
    (1, 5) : -Inf : xa[1,5] + xc[1,5] + xf[1,5] + xw[1,5] - 50*d[1,5] : 0.0 : True
    (1, 6) : -Inf : xa[1,6] + xc[1,6] + xf[1,6] + xw[1,6] - 50*d[1,6] : 0.0 : True
```

```
        (1, 7) : -Inf : xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7] - 50*d[1,7] : 0.0 : True
        (1, 8) : -Inf : xa[1,8] + xc[1,8] + xf[1,8] + xw[1,8] - 50*d[1,8] : 0.0 : True
        (2, 1) : -Inf : xa[2,1] + xc[2,1] + xf[2,1] + xw[2,1] - 50*d[2,1] : 0.0 : True
        (2, 2) : -Inf : xa[2,2] + xc[2,2] + xf[2,2] + xw[2,2] - 50*d[2,2] : 0.0 : True
        (2, 3) : -Inf : xa[2,3] + xc[2,3] + xf[2,3] + xw[2,3] - 50*d[2,3] : 0.0 : True
        (2, 4) : -Inf : xa[2,4] + xc[2,4] + xf[2,4] + xw[2,4] - 50*d[2,4] : 0.0 : True
        (2, 5) : -Inf : xa[2,5] + xc[2,5] + xf[2,5] + xw[2,5] - 50*d[2,5] : 0.0 : True
        (2, 6) : -Inf : xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6] - 50*d[2,6] : 0.0 : True
        (2, 7) : -Inf : xa[2,7] + xc[2,7] + xf[2,7] + xw[2,7] - 50*d[2,7] : 0.0 : True
        (2, 8) : -Inf : xa[2,8] + xc[2,8] + xf[2,8] + xw[2,8] - 50*d[2,8] : 0.0 : True
    supplymin : Minimum quantity from supplier i to consumer j
        Size=16, Index=supplymin_index, Active=True
        Key    : Lower : Body                                  :Upper:Active
        (1, 1):-Inf:10*d[1,1]-xa[1,1]-xc[1,1]-xf[1,1]-xw[1,1]:0.0:True
        (1, 2):-Inf:10*d[1,2]-xa[1,2]-xc[1,2]-xf[1,2]-xw[1,2]:0.0:True
        (1, 3):-Inf:10*d[1,3]-xa[1,3]-xc[1,3]-xf[1,3]-xw[1,3]:0.0:True
        (1, 4):-Inf:10*d[1,4]-xa[1,4]-xc[1,4]-xf[1,4]-xw[1,4]:0.0:True
        (1, 5):-Inf:10*d[1,5]-xa[1,5]-xc[1,5]-xf[1,5]-xw[1,5]:0.0:True
        (1, 6):-Inf:10*d[1,6]-xa[1,6]-xc[1,6]-xf[1,6]-xw[1,6]:0.0:True
        (1, 7):-Inf:10*d[1,7]-xa[1,7]-xc[1,7]-xf[1,7]-xw[1,7]:0.0:True
        (1, 8):-Inf:10*d[1,8]-xa[1,8]-xc[1,8]-xf[1,8]-xw[1,8]:0.0:True
        (2, 1):-Inf:10*d[2,1]-xa[2,1]-xc[2,1]-xf[2,1]-xw[2,1]:0.0:True
        (2, 2):-Inf:10*d[2,2]-xa[2,2]-xc[2,2]-xf[2,2]-xw[2,2]:0.0:True
        (2, 3):-Inf:10*d[2,3]-xa[2,3]-xc[2,3]-xf[2,3]-xw[2,3]:0.0:True
        (2, 4):-Inf:10*d[2,4]-xa[2,4]-xc[2,4]-xf[2,4]-xw[2,4]:0.0:True
        (2, 5):-Inf:10*d[2,5]-xa[2,5]-xc[2,5]-xf[2,5]-xw[2,5]:0.0 True
        (2, 6):-Inf:10*d[2,6]-xa[2,6]-xc[2,6]-xf[2,6]-xw[2,6]:0.0:True
        (2, 7):-Inf:10*d[2,7]-xa[2,7]-xc[2,7]-xf[2,7]-xw[2,7]:0.0:True
        (2, 8):-Inf:10*d[2,8]-xa[2,8]-xc[2,8]-xf[2,8]-xw[2,8]:0.0:True

37 Declarations: NS NC I J SS CS m M c_index c Ot Of o_index o qa qc qf
```

```
qw xa_index xa xc_index xc xf_index xf xw_index xw d_index d supplyA
supplyC supplyF supplyW supplymin_index supplymin
supplymax_index supplymax objective
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 57115.0
  Upper bound: 57115.0
  Number of objectives: 1
  Number of constraints: 65
  Number of variables: 81
  Number of nonzeros: 225
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 11
      Number of created subproblems: 11
  Error rc: 0
  Time: 0.14010190963745117
# ----------------------------------------------------------
```

```
#   Solution Information
# ---------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
Variable xa
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 22.0
    (2, 2) 0.0
    (2, 3) 18.0
    (2, 4) 28.0
    (2, 5) 16.0
    (2, 6) 4.0
    (2, 7) 0.0
    (2, 8) 4.0
Variable xc
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 14.0
    (1, 5) 28.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
```

```
    (2, 1) 8.0
    (2, 2) 22.0
    (2, 3) 0.0
    (2, 4) 4.0
    (2, 5) 0.0
    (2, 6) 16.0
    (2, 7) 4.0
    (2, 8) 0.0
Variable xf
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 18.0
    (1, 5) 22.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 8.0
    (2, 2) 22.0
    (2, 3) 0.0
    (2, 4) 0.0
    (2, 5) 6.0
    (2, 6) 16.0
    (2, 7) 4.0
    (2, 8) 8.0
Variable xw
    (1, 1) 0.0
    (1, 2) 16.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
```

```
    (1, 6) 10.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 8.0
    (2, 2) 6.0
    (2, 3) 0.0
    (2, 4) 18.0
    (2, 5) 28.0
    (2, 6) 6.0
    (2, 7) 4.0
    (2, 8) 5.0
Variable d
    (1, 1) 0.0
    (1, 2) 1.0
    (1, 3) 0.0
    (1, 4) 1.0
    (1, 5) 1.0
    (1, 6) 1.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 1.0
    (2, 2) 1.0
    (2, 3) 1.0
    (2, 4) 1.0
    (2, 5) 1.0
    (2, 6) 1.0
    (2, 7) 1.0
    (2, 8) 1.0
```

C.4 EXPERIMENT 2

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Tue Oct 31 05:48:31 2017

@author: T.Fletcher

Variation of transport2.py, with no integer constraints for quantities.
"""

# Import
import pyomo
import pyomo.opt
import pyomo.environ as pe
import os

#problem name
PROBLEM_NAME = "Minimal cost supplies delivery"


# Creation of an Abstract Model
model = pe.AbstractModel(doc=PROBLEM_NAME)

##########################
#general parameters: all read from a data file
##########################
#number of suppliers and consumers
model.NS = pe.Param(within=pe.PositiveIntegers)
model.NC = pe.Param(within=pe.PositiveIntegers)

#indexes over suppliers/consumers respectively
model.I = pe.RangeSet(1, model.NS)
model.J = pe.RangeSet(1, model.NC)
```

```
## Define sets ##
#  Sets
#       model.SS   Suppliers set
#       model.CS   Consummers set
model.SS = pe.Set(doc='Suppliers')
model.CS = pe.Set(doc='Consummers')

#lower/upper bounds for quantities
model.m = pe.Param()
model.M = pe.Param()

#cost (distance) along each route
model.c = pe.Param(model.I, model.J)

#overload cost (security) parameters
model.Ot = pe.Param()
model.Of = pe.Param()
def O_init(model, i, j):
    #
    # Create the value of model.o[i,j]
    #
    v = 0.0
    if model.c[i,j] >= model.Ot:
        v = model.c[i,j] * model.Of
    return v
model.o = pe.Param(model.I, model.J, initialize=O_init)

#requested quantities
model.qa = pe.Param(model.J)
model.Qa = pe.Param(model.J)
```

```
model.qc = pe.Param(model.J)
model.Qc = pe.Param(model.J)
model.qf = pe.Param(model.J)
model.Qf = pe.Param(model.J)
model.qw = pe.Param(model.J)
model.Qw = pe.Param(model.J)


###################
#  Variables
################
#quantities (number of trucks) for each commodity
model.xa = pe.Var(model.I, model.J, domain = pe.PositiveReals) #ammo
model.xc = pe.Var(model.I, model.J, domain = pe.PositiveReals) #cargo
model.xf = pe.Var(model.I, model.J, domain = pe.PositiveReals) #fuel
model.xw = pe.Var(model.I, model.J, domain = pe.PositiveReals) #water

#binary variables
model.d = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers,
                 bounds = (0,1))


############################
# Constraints
############################
def supply_ruleAl(model, j):
  return sum(model.xa[i,j] for i in model.I) >= model.qa[j]
model.supplyAl = pe.Constraint(model.J, rule=supply_ruleAl,
                          doc='Supply requested ammo quantities for j (lower bound)')
def supply_ruleAu(model, j):
  return sum(model.xa[i,j] for i in model.I) <= model.Qa[j]
```

```
model.supplyAu = pe.Constraint(model.J, rule=supply_ruleAu,
                          doc='Supply requested ammo quantities for j (upper bound)')

def supply_ruleCl(model, j):
  return sum(model.xc[i,j] for i in model.I) >= model.qc[j]
model.supplyCl = pe.Constraint(model.J, rule=supply_ruleCl,
                          doc='Supply requested cargo quantities for j (lower bound)')
def supply_ruleCu(model, j):
  return sum(model.xc[i,j] for i in model.I) <= model.Qc[j]
model.supplyCu = pe.Constraint(model.J, rule=supply_ruleCu,
                          doc='Supply requested cargo quantities for j (upper bound)')

def supply_ruleFl(model, j):
  return sum(model.xf[i,j] for i in model.I) >= model.qf[j]
model.supplyFl = pe.Constraint(model.J, rule=supply_ruleFl,
                          doc='Supply requested fuel quantities for j (lower bound)')
def supply_ruleFu(model, j):
  return sum(model.xf[i,j] for i in model.I) <= model.Qf[j]
model.supplyFu = pe.Constraint(model.J, rule=supply_ruleFu,
                          doc='Supply requested fuel quantities for j (upper bound)')

def supply_ruleWl(model, j):
  return sum(model.xw[i,j] for i in model.I) >= model.qw[j]
model.supplyWl = pe.Constraint(model.J, rule=supply_ruleWl,
                          doc='Supply requested water quantities for j (lower bound)')
def supply_ruleWu(model, j):
  return sum(model.xw[i,j] for i in model.I) <= model.Qw[j]
model.supplyWu = pe.Constraint(model.J, rule=supply_ruleWu,
                          doc='Supply requested water quantities for j (upper bound)')
```

```
#constraints for binary variables
#quantities lower bound
def supply_minq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
                                                >= model.d[i,j]*model.m
model.supplymin = pe.Constraint(model.I, model.J, rule=supply_minq,
                          doc='Minimum quantity from supplier i to consumer j')

#quantities upper bound
def supply_maxq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
                                               <= model.d[i,j]*model.M
model.supplymax = pe.Constraint(model.I, model.J, rule=supply_maxq,
                          doc='Maximum quantity from supplier i to consumer j')


################################################
## Define Objective function ##
#  minimize cost   C =  sum((i,j), c(i,j)*(xa(i,j) + xc(i,j) + xf(i,j) + xw(i,j))
                                                              + d(i,j)*o(i,j)) ;
def objective_rule(model):
  return sum(model.c[i,j]*(model.xa[i,j] + model.xc[i,j]
             + model.xf[i,j] + model.xw[i,j])
             + model.d[i,j]*model.o[i,j]
             for i in model.I for j in model.J)

model.objective = pe.Objective(rule=objective_rule, sense=pe.minimize,
                            doc='The objective function')

##read these parameters from a data file (relative to working directory)
cwdir = os.getcwd() #the current working directory must be set to the script location
```

```
datafile = os.path.join(os.sep, cwdir + '\\data\\constants2a.dat')
instance = model.create_instance(datafile)
instance.pprint()



"""Solve the model."""
#solver = pyomo.opt.SolverFactory('gurobi')
#results = solver.solve(instance, tee=True, keepfiles=False,
options_string="mip_tolerances_integrality=1e-9 mip_tolerances_mipgap=0")
solver = pyomo.opt.SolverFactory('glpk')
results = solver.solve(instance)
results.write()
instance.solutions.load_from(results)

for v in instance.component_objects(pe.Var, active=True):
    print ("Variable",str(v))
    varobject = getattr(instance, str(v))
    for index in varobject:
        print ("   ",index, varobject[index].value)
```

C.5 EXPERIMENT 2 DATA FILE

```
##
## Data set for Scenario 2
##

#the number of suppliers and consumers
param NS := 2;
param NC := 8;

#the set of suppliers
```

```
set SS := 'C2' 'C3';
#the set of consummers
set CS := 'IN1M' 'IN1S1' 'IN1S2' 'IN1S3' 'IN1XS1' 'IN1XS2' 'IN1XS3' 'IN1XS4';

#lower/upper number of trucks bounds
param m := 5;
param M := 30;


#costs (distances) from each supplier to each consummer
param c: 1 2 3 4 5 6 7 8 :=
1 120 143 183 163 158 195 193 172
2 93 116 156 136 131 168 166 145
;


#overload threshold: the distance above which an overload factor is applied
param Ot := 150;
#the overload factor: the distance-multiplier when overload is applied
param Of := 2;

#requested quantities for: ammo, cargo, fuel, water
#(one value for each consumer in the list)
param Qa :=
1  5
2 10.5
3 6.5
4 9.5
5 2.5
6 7.5
7  9
```

```
8 8.5
;
param qa :=
1 0
2 4.5
3  2
4 0
5 0
6 3.5
7  0
8 4.5
;
param Qc :=
1 16
2 9
3 3
4 3.5
5 5
6 8
7 4.5
8 8
;
param qc :=
1 4.1
2 2
3 2.4
4 2
5 2.5
6 4.4
7 0
8 0
```

```
;
param Qf :=
1 88
2 11.2
3 1.5
4 4
5 3
6 3.3
7 3.3
8 3.2
;
param qf :=
1 0
2 3
3 .5
4 0
5 3
6 3
7 3
8 3
;
param Qw :=
1 159
2 16
3 5
4 26.1
5 3.2
6 1
7 4.4
8 5
;
```

```
param qw :=
1 5.1
2 0
3 0
4 13.18
5 1
6 0
7 1
8 0
;
```

C.6 RESULTS EXPERIMENT 2

```
   supplymin_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xa_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xc_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xf_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xw_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual

2 RangeSet Declarations
    I : Dim=0, Dimen=1, Size=2, Domain=Integers, Ordered=True, Bounds=(1, 2)
        Virtual
    J : Dim=0, Dimen=1, Size=8, Domain=Integers, Ordered=True, Bounds=(1, 8)
        Virtual

16 Param Declarations
    M : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
```

```
    Key  : Value
    None :    30
NC : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
    Key  : Value
    None :     8
NS : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
    Key  : Value
    None :     2
Of : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :     2
Ot : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :   150
Qa : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :     5
      2 :  10.5
      3 :   6.5
      4 :   9.5
      5 :   2.5
      6 :   7.5
      7 :     9
      8 :   8.5
Qc : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    16
      2 :     9
      3 :     3
      4 :   3.5
      5 :     5
```

```
      6 :     8
      7 :   4.5
      8 :     8
Qf : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    88
      2 :  11.2
      3 :   1.5
      4 :     4
      5 :     3
      6 :   3.3
      7 :   3.3
      8 :   3.2
Qw : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   159
      2 :    16
      3 :     5
      4 :  26.1
      5 :   3.2
      6 :     1
      7 :   4.4
      8 :     5
c : Size=16, Index=c_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   120
    (1, 2) :   143
    (1, 3) :   183
    (1, 4) :   163
    (1, 5) :   158
    (1, 6) :   195
```

```
    (1, 7) :   193
    (1, 8) :   172
    (2, 1) :    93
    (2, 2) :   116
    (2, 3) :   156
    (2, 4) :   136
    (2, 5) :   131
    (2, 6) :   168
    (2, 7) :   166
    (2, 8) :   145
m : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :     1
o : Size=16, Index=o_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   0.0
    (1, 2) :   0.0
    (1, 3) :   366
    (1, 4) :   326
    (1, 5) :   316
    (1, 6) :   390
    (1, 7) :   386
    (1, 8) :   344
    (2, 1) :   0.0
    (2, 2) :   0.0
    (2, 3) :   312
    (2, 4) :   0.0
    (2, 5) :   0.0
    (2, 6) :   336
    (2, 7) :   332
    (2, 8) :   0.0
```

```
qa : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :     0
      2 :   4.5
      3 :     2
      4 :     0
      5 :     0
      6 :   3.5
      7 :     0
      8 :   4.5
qc : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   4.1
      2 :     2
      3 :   2.4
      4 :     2
      5 :   2.5
      6 :   4.4
      7 :     0
      8 :     0
qf : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :     0
      2 :     3
      3 :   0.5
      4 :     0
      5 :     3
      6 :     3
      7 :     3
      8 :     3
qw : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
```

```
        Key : Value
          1 :   5.1
          2 :     0
          3 :     0
          4 : 13.18
          5 :     1
          6 :     0
          7 :     1
          8 :     0

5 Var Declarations
    d : Size=16, Index=d_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    xa : Size=16, Index=xa_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
```

```
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xc : Size=16, Index=xc_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
```

```
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xf : Size=16, Index=xf_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xw : Size=16, Index=xw_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
```

```
        (2, 1) :     0 :  None :  None : False :  True : PositiveReals
        (2, 2) :     0 :  None :  None : False :  True : PositiveReals
        (2, 3) :     0 :  None :  None : False :  True : PositiveReals
        (2, 4) :     0 :  None :  None : False :  True : PositiveReals
        (2, 5) :     0 :  None :  None : False :  True : PositiveReals
        (2, 6) :     0 :  None :  None : False :  True : PositiveReals
        (2, 7) :     0 :  None :  None : False :  True : PositiveReals
        (2, 8) :     0 :  None :  None : False :  True : PositiveReals

1 Objective Declarations
    objective : The objective function
        Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None : True : minimize : 120*( xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1])
        + 143*( xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2] ) + 183*( xa[1,3]
        + xc[1,3] + xf[1,3] + xw[1,3] ) + 366*d[1,3] + 163*( xa[1,4] + xc[1,4]
        + xf[1,4] + xw[1,4] ) + 326*d[1,4] + 158*( xa[1,5] + xc[1,5] + xf[1,5]
        + xw[1,5] ) + 316*d[1,5] + 195*( xa[1,6] + xc[1,6] + xf[1,6] + xw[1,6])
        + 390*d[1,6] + 193*(xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7]) + 386*d[1,7]
        + 172*(xa[1,8] + xc[1,8] + xf[1,8] + xw[1,8]) + 344*d[1,8] + 93*(xa[2,1]
        + xc[2,1] + xf[2,1] + xw[2,1]) + 116*( xa[2,2] + xc[2,2] + xf[2,2]
        + xw[2,2] ) + 156*( xa[2,3] + xc[2,3] + xf[2,3] + xw[2,3]) + 312*d[2,3]
        + 136*( xa[2,4] + xc[2,4] + xf[2,4] + xw[2,4]) + 131*(xa[2,5] + xc[2,5]
        + xf[2,5] + xw[2,5] ) + 168*( xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6] )
        + 336*d[2,6] + 166*(xa[2,7] + xc[2,7] + xf[2,7] + xw[2,7]) + 332*d[2,7]
        + 145*(xa[2,8] + xc[2,8] + xf[2,8] + xw[2,8])

10 Constraint Declarations
    supplyAl : Supply requested ammo quantities for j (lower bound)
        Size=8, Index=J, Active=True
        Key : Lower : Body              : Upper : Active
```

```
      1 :   0.0 : xa[1,1] + xa[2,1] :  +Inf :   True
      2 :   4.5 : xa[1,2] + xa[2,2] :  +Inf :   True
      3 :   2.0 : xa[1,3] + xa[2,3] :  +Inf :   True
      4 :   0.0 : xa[1,4] + xa[2,4] :  +Inf :   True
      5 :   0.0 : xa[1,5] + xa[2,5] :  +Inf :   True
      6 :   3.5 : xa[1,6] + xa[2,6] :  +Inf :   True
      7 :   0.0 : xa[1,7] + xa[2,7] :  +Inf :   True
      8 :   4.5 : xa[1,8] + xa[2,8] :  +Inf :   True
supplyAu : Supply requested ammo quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xa[1,1] + xa[2,1] :   5.0 :   True
      2 :  -Inf : xa[1,2] + xa[2,2] :  10.5 :   True
      3 :  -Inf : xa[1,3] + xa[2,3] :   6.5 :   True
      4 :  -Inf : xa[1,4] + xa[2,4] :   9.5 :   True
      5 :  -Inf : xa[1,5] + xa[2,5] :   2.5 :   True
      6 :  -Inf : xa[1,6] + xa[2,6] :   7.5 :   True
      7 :  -Inf : xa[1,7] + xa[2,7] :   9.0 :   True
      8 :  -Inf : xa[1,8] + xa[2,8] :   8.5 :   True
supplyCl : Supply requested cargo quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   4.1 : xc[1,1] + xc[2,1] :  +Inf :   True
      2 :   2.0 : xc[1,2] + xc[2,2] :  +Inf :   True
      3 :   2.4 : xc[1,3] + xc[2,3] :  +Inf :   True
      4 :   2.0 : xc[1,4] + xc[2,4] :  +Inf :   True
      5 :   2.5 : xc[1,5] + xc[2,5] :  +Inf :   True
      6 :   4.4 : xc[1,6] + xc[2,6] :  +Inf :   True
      7 :   0.0 : xc[1,7] + xc[2,7] :  +Inf :   True
      8 :   0.0 : xc[1,8] + xc[2,8] :  +Inf :   True
supplyCu : Supply requested cargo quantities for j (upper bound)
```

```
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xc[1,1] + xc[2,1] :  16.0 :   True
      2 :  -Inf : xc[1,2] + xc[2,2] :   9.0 :   True
      3 :  -Inf : xc[1,3] + xc[2,3] :   3.0 :   True
      4 :  -Inf : xc[1,4] + xc[2,4] :   3.5 :   True
      5 :  -Inf : xc[1,5] + xc[2,5] :   5.0 :   True
      6 :  -Inf : xc[1,6] + xc[2,6] :   8.0 :   True
      7 :  -Inf : xc[1,7] + xc[2,7] :   4.5 :   True
      8 :  -Inf : xc[1,8] + xc[2,8] :   8.0 :   True
supplyFl : Supply requested fuel quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   0.0 : xf[1,1] + xf[2,1] :  +Inf :   True
      2 :   3.0 : xf[1,2] + xf[2,2] :  +Inf :   True
      3 :   0.5 : xf[1,3] + xf[2,3] :  +Inf :   True
      4 :   0.0 : xf[1,4] + xf[2,4] :  +Inf :   True
      5 :   3.0 : xf[1,5] + xf[2,5] :  +Inf :   True
      6 :   3.0 : xf[1,6] + xf[2,6] :  +Inf :   True
      7 :   3.0 : xf[1,7] + xf[2,7] :  +Inf :   True
      8 :   3.0 : xf[1,8] + xf[2,8] :  +Inf :   True
supplyFu : Supply requested fuel quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xf[1,1] + xf[2,1] :  88.0 :   True
      2 :  -Inf : xf[1,2] + xf[2,2] :  11.2 :   True
      3 :  -Inf : xf[1,3] + xf[2,3] :   1.5 :   True
      4 :  -Inf : xf[1,4] + xf[2,4] :   4.0 :   True
      5 :  -Inf : xf[1,5] + xf[2,5] :   3.0 :   True
      6 :  -Inf : xf[1,6] + xf[2,6] :   3.3 :   True
      7 :  -Inf : xf[1,7] + xf[2,7] :   3.3 :   True
```

```
      8 :  -Inf : xf[1,8] + xf[2,8] :   3.2 :   True
supplyWl : Supply requested water quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   5.1 : xw[1,1] + xw[2,1] :  +Inf :   True
      2 :   0.0 : xw[1,2] + xw[2,2] :  +Inf :   True
      3 :   0.0 : xw[1,3] + xw[2,3] :  +Inf :   True
      4 : 13.18 : xw[1,4] + xw[2,4] :  +Inf :   True
      5 :   1.0 : xw[1,5] + xw[2,5] :  +Inf :   True
      6 :   0.0 : xw[1,6] + xw[2,6] :  +Inf :   True
      7 :   1.0 : xw[1,7] + xw[2,7] :  +Inf :   True
      8 :   0.0 : xw[1,8] + xw[2,8] :  +Inf :   True
supplyWu : Supply requested water quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xw[1,1] + xw[2,1] : 159.0 :   True
      2 :  -Inf : xw[1,2] + xw[2,2] :  16.0 :   True
      3 :  -Inf : xw[1,3] + xw[2,3] :   5.0 :   True
      4 :  -Inf : xw[1,4] + xw[2,4] :  26.1 :   True
      5 :  -Inf : xw[1,5] + xw[2,5] :   3.2 :   True
      6 :  -Inf : xw[1,6] + xw[2,6] :   1.0 :   True
      7 :  -Inf : xw[1,7] + xw[2,7] :   4.4 :   True
      8 :  -Inf : xw[1,8] + xw[2,8] :   5.0 :   True
supplymax : Maximum quantity from supplier i to consumer j
    Size=16, Index=supplymax_index, Active=True
    Key : Lower : Body : Upper : Active
    (1, 1) : -Inf : xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1] - 30*d[1,1] : 0.0 : True
    (1, 2) : -Inf : xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2] - 30*d[1,2] : 0.0 : True
    (1, 3) : -Inf : xa[1,3] + xc[1,3] + xf[1,3] + xw[1,3] - 30*d[1,3] : 0.0 : True
    (1, 4) : -Inf : xa[1,4] + xc[1,4] + xf[1,4] + xw[1,4] - 30*d[1,4] : 0.0 : True
    (1, 5) : -Inf : xa[1,5] + xc[1,5] + xf[1,5] + xw[1,5] - 30*d[1,5] : 0.0 : True
```

```
    (1, 6) : -Inf : xa[1,6] + xc[1,6] + xf[1,6] + xw[1,6] - 30*d[1,6] : 0.0 : True
    (1, 7) : -Inf : xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7] - 30*d[1,7] : 0.0 : True
    (1, 8) : -Inf : xa[1,8] + xc[1,8] + xf[1,8] + xw[1,8] - 30*d[1,8] : 0.0 : True
    (2, 1) : -Inf : xa[2,1] + xc[2,1] + xf[2,1] + xw[2,1] - 30*d[2,1] : 0.0 : True
    (2, 2) : -Inf : xa[2,2] + xc[2,2] + xf[2,2] + xw[2,2] - 30*d[2,2] : 0.0 : True
    (2, 3) : -Inf : xa[2,3] + xc[2,3] + xf[2,3] + xw[2,3] - 30*d[2,3] : 0.0 : True
    (2, 4) : -Inf : xa[2,4] + xc[2,4] + xf[2,4] + xw[2,4] - 30*d[2,4] : 0.0 : True
    (2, 5) : -Inf : xa[2,5] + xc[2,5] + xf[2,5] + xw[2,5] - 30*d[2,5] : 0.0 : True
    (2, 6) : -Inf : xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6] - 30*d[2,6] : 0.0 : True
    (2, 7) : -Inf : xa[2,7] + xc[2,7] + xf[2,7] + xw[2,7] - 30*d[2,7] : 0.0 : True
    (2, 8) : -Inf : xa[2,8] + xc[2,8] + xf[2,8] + xw[2,8] - 30*d[2,8] : 0.0 : True
supplymin : Minimum quantity from supplier i to consumer j
    Size=16, Index=supplymin_index, Active=True
    Key : Lower : Body : Upper : Active
    (1, 1) : -Inf : d[1,1] - xa[1,1] - xc[1,1] - xf[1,1] - xw[1,1] : 0.0 : True
    (1, 2) : -Inf : d[1,2] - xa[1,2] - xc[1,2] - xf[1,2] - xw[1,2] : 0.0 : True
    (1, 3) : -Inf : d[1,3] - xa[1,3] - xc[1,3] - xf[1,3] - xw[1,3] : 0.0 : True
    (1, 4) : -Inf : d[1,4] - xa[1,4] - xc[1,4] - xf[1,4] - xw[1,4] : 0.0 : True
    (1, 5) : -Inf : d[1,5] - xa[1,5] - xc[1,5] - xf[1,5] - xw[1,5] : 0.0 : True
    (1, 6) : -Inf : d[1,6] - xa[1,6] - xc[1,6] - xf[1,6] - xw[1,6] : 0.0 : True
    (1, 7) : -Inf : d[1,7] - xa[1,7] - xc[1,7] - xf[1,7] - xw[1,7] : 0.0 : True
    (1, 8) : -Inf : d[1,8] - xa[1,8] - xc[1,8] - xf[1,8] - xw[1,8] : 0.0 : True
    (2, 1) : -Inf : d[2,1] - xa[2,1] - xc[2,1] - xf[2,1] - xw[2,1] : 0.0 : True
    (2, 2) : -Inf : d[2,2] - xa[2,2] - xc[2,2] - xf[2,2] - xw[2,2] : 0.0 : True
    (2, 3) : -Inf : d[2,3] - xa[2,3] - xc[2,3] - xf[2,3] - xw[2,3] : 0.0 : True
    (2, 4) : -Inf : d[2,4] - xa[2,4] - xc[2,4] - xf[2,4] - xw[2,4] : 0.0 : True
    (2, 5) : -Inf : d[2,5] - xa[2,5] - xc[2,5] - xf[2,5] - xw[2,5] : 0.0 : True
    (2, 6) : -Inf : d[2,6] - xa[2,6] - xc[2,6] - xf[2,6] - xw[2,6] : 0.0 : True
    (2, 7) : -Inf : d[2,7] - xa[2,7] - xc[2,7] - xf[2,7] - xw[2,7] : 0.0 : True
    (2, 8) : -Inf : d[2,8] - xa[2,8] - xc[2,8] - xf[2,8] - xw[2,8] : 0.0 : True
```

```
45 Declarations: NS NC I J SS CS m M c_index c Ot Of o_index o qa Qa qc Qc
 qf Qf qw Qw xa_index xa xc_index xc xf_index
xf xw_index xw d_index d supplyAl supplyAu supplyCl supplyCu supplyFl
supplyFu supplyWl supplyWu supplymin_index
supplymin supplymax_index supplymax objective
# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 10200.68
  Upper bound: 10200.68
  Number of objectives: 1
  Number of constraints: 97
  Number of variables: 81
  Number of nonzeros: 289
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 33
      Number of created subproblems: 33
  Error rc: 0
```

```
  Time: 0.18099379539489746
# ------------------------------------------------------------
#   Solution Information
# ------------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
Variable xa
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 0.0
    (2, 2) 4.5
    (2, 3) 2.0
    (2, 4) 0.0
    (2, 5) 0.0
    (2, 6) 3.5
    (2, 7) 0.0
    (2, 8) 4.5
Variable xc
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
```

```
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 4.1
    (2, 2) 2.0
    (2, 3) 2.4
    (2, 4) 2.0
    (2, 5) 2.5
    (2, 6) 4.4
    (2, 7) 0.0
    (2, 8) 0.0
Variable xf
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 0.0
    (2, 2) 3.0
    (2, 3) 0.5
    (2, 4) 0.0
    (2, 5) 3.0
    (2, 6) 3.0
    (2, 7) 3.0
    (2, 8) 3.0
Variable xw
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
```

```
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 5.1
    (2, 2) 0.0
    (2, 3) 0.0
    (2, 4) 13.18
    (2, 5) 1.0
    (2, 6) 0.0
    (2, 7) 1.0
    (2, 8) 0.0
Variable d
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.0
    (2, 1) 1.0
    (2, 2) 1.0
    (2, 3) 1.0
    (2, 4) 1.0
    (2, 5) 1.0
    (2, 6) 1.0
    (2, 7) 1.0
    (2, 8) 1.0
```

C.7 EXPERIMENT 3

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct 31 05:48:31 2017

@author: T.Fletcher

Variation of transport2.py, with no integer constraints for quantities.
"""

# Import
import pyomo
import pyomo.opt
import pyomo.environ as pe
import os

#problem name
PROBLEM_NAME = "Minimal cost supplies delivery"


# Creation of an Abstract Model
model = pe.AbstractModel(doc=PROBLEM_NAME)

##########################
#general parameters: all read from a data file
##########################
#number of suppliers and consumers
model.NS = pe.Param(within=pe.PositiveIntegers)
model.NC = pe.Param(within=pe.PositiveIntegers)

#indexes over suppliers/consumers respectively
```

```
model.I = pe.RangeSet(1, model.NS)
model.J = pe.RangeSet(1, model.NC)

## Define sets ##
#  Sets
#       model.SS   Suppliers set
#       model.CS   Consummers set
model.SS = pe.Set(doc='Suppliers')
model.CS = pe.Set(doc='Consummers')

#lower/upper bounds for quantities
model.m = pe.Param()
model.M = pe.Param()

#cost (distance) along each route
model.c = pe.Param(model.I, model.J)

#overload cost (security) parameters
model.o = pe.Param(model.I, model.J)

#requested quantities
model.qa = pe.Param(model.J)
model.Qa = pe.Param(model.J)
model.qc = pe.Param(model.J)
model.Qc = pe.Param(model.J)
model.qf = pe.Param(model.J)
model.Qf = pe.Param(model.J)
model.qw = pe.Param(model.J)
model.Qw = pe.Param(model.J)
```

```
###################
#  Variables
################
#quantities (number of trucks) for each commodity
model.xa = pe.Var(model.I, model.J, domain = pe.PositiveReals) #ammo
model.xc = pe.Var(model.I, model.J, domain = pe.PositiveReals) #cargo
model.xf = pe.Var(model.I, model.J, domain = pe.PositiveReals) #fuel
model.xw = pe.Var(model.I, model.J, domain = pe.PositiveReals) #water

#binary variables
model.d = pe.Var(model.I, model.J, domain = pe.NonNegativeIntegers,
                 bounds = (0,1))



############################
# Constraints
############################
def supply_ruleAl(model, j):
  return sum(model.xa[i,j] for i in model.I) >= model.qa[j]
model.supplyAl = pe.Constraint(model.J, rule=supply_ruleAl,
                          doc='Supply requested ammo quantities for j (lower bound)')
def supply_ruleAu(model, j):
  return sum(model.xa[i,j] for i in model.I) <= model.Qa[j]
model.supplyAu = pe.Constraint(model.J, rule=supply_ruleAu,
                          doc='Supply requested ammo quantities for j (upper bound)')

def supply_ruleCl(model, j):
  return sum(model.xc[i,j] for i in model.I) >= model.qc[j]
model.supplyCl = pe.Constraint(model.J, rule=supply_ruleCl,
                          doc='Supply requested cargo quantities for j (lower bound)')
def supply_ruleCu(model, j):
```

```
  return sum(model.xc[i,j] for i in model.I) <= model.Qc[j]
model.supplyCu = pe.Constraint(model.J, rule=supply_ruleCu,
                          doc='Supply requested cargo quantities for j (upper bound)')

def supply_ruleFl(model, j):
  return sum(model.xf[i,j] for i in model.I) >= model.qf[j]
model.supplyFl = pe.Constraint(model.J, rule=supply_ruleFl,
                          doc='Supply requested fuel quantities for j (lower bound)')
def supply_ruleFu(model, j):
  return sum(model.xf[i,j] for i in model.I) <= model.Qf[j]
model.supplyFu = pe.Constraint(model.J, rule=supply_ruleFu,
                          doc='Supply requested fuel quantities for j (upper bound)')

def supply_ruleWl(model, j):
  return sum(model.xw[i,j] for i in model.I) >= model.qw[j]
model.supplyWl = pe.Constraint(model.J, rule=supply_ruleWl,
                          doc='Supply requested water quantities for j (lower bound)')
def supply_ruleWu(model, j):
  return sum(model.xw[i,j] for i in model.I) <= model.Qw[j]
model.supplyWu = pe.Constraint(model.J, rule=supply_ruleWu,
                          doc='Supply requested water quantities for j (upper bound)')


#constraints for binary variables
#quantities lower bound
def supply_minq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
  >= model.d[i,j]*model.m
model.supplymin = pe.Constraint(model.I, model.J, rule=supply_minq,
                          doc='Minimum quantity from supplier i to consumer j')
```

```
#quantities upper bound
def supply_maxq(model, i, j):
  return model.xa[i,j] + model.xc[i,j] + model.xf[i,j] + model.xw[i,j]
  <= model.d[i,j]*model.M
model.supplymax = pe.Constraint(model.I, model.J, rule=supply_maxq,
                          doc='Maximum quantity from supplier i to consumer j')


##############################################
## Define Objective function ##
#  minimize cost   C =  sum((i,j), c(i,j)*(xa(i,j) + xc(i,j) + xf(i,j) + xw(i,j))
+ d(i,j)*o(i,j)) ;
def objective_rule(model):
  return sum(model.c[i,j]*(model.xa[i,j] + model.xc[i,j]
             + model.xf[i,j] + model.xw[i,j])
             + model.d[i,j]*model.o[i,j]
             for i in model.I for j in model.J)

model.objective = pe.Objective(rule=objective_rule, sense=pe.minimize,
                           doc='The objective function')

##read these parameters from a data file (relative to working directory)
cwdir = os.getcwd() #the current working directory must be set to the script location
datafile = os.path.join(os.sep, cwdir + '\\data\\constants2b.dat')
instance = model.create_instance(datafile)
instance.pprint()



"""Solve the model."""
#solver = pyomo.opt.SolverFactory('gurobi')
```

```
#results = solver.solve(instance, tee=True, keepfiles=False, options_string=
"mip_tolerances_integrality=1e-9 mip_tolerances_mipgap=0")
solver = pyomo.opt.SolverFactory('glpk')
results = solver.solve(instance)
results.write()
instance.solutions.load_from(results)

for v in instance.component_objects(pe.Var, active=True):
    print ("Variable",str(v))
    varobject = getattr(instance, str(v))
    for index in varobject:
        print ("   ",index, varobject[index].value)
```

C.8 Experiment 3 Data File

```
##
## Data set for Scenario 3
##

#the number of suppliers and consumers
param NS := 2;
param NC := 8;

#the set of suppliers
set SS := 'C2' 'C3';
#the set of consummers
set CS := 'IN1M' 'IN1S1' 'IN1S2' 'IN1S3' 'IN1XS1' 'IN1XS2' 'IN1XS3' 'IN1XS4';

#lower/upper number of trucks bounds
param m := 5;
param M := 30;
```

```
#costs (distances) from each supplier to each consummer
param c: 1 2 3 4 5 6 7 8 :=
1 120 143 183 163 158 195 193 172
2 93 116 156 136 131 168 166 145
;

#security costs from each supplier to each consummer
param o: 1 2 3 4 5 6 7 8 :=
1 120 143 183 163 158 195 193 172
2 93 116 156 136 131 168 664 580
;

#requested quantities for: ammo, cargo, fuel, water
#(one value for each consumer in the list)
param Qa :=
1  5
2 10.5
3 6.5
4 9.5
5 2.5
6 7.5
7  9
8 8.5
;
param qa :=
1 0
2 4.5
3  2
4 0
5 0
```

```
6 3.5
7  0
8 4.5
;
param Qc :=
1 16
2 9
3 3
4 3.5
5 5
6 8
7 4.5
8 8
;
param qc :=
1 4.1
2 2
3 2.4
4 2
5 2.5
6 4.4
7 0
8 0
;
param Qf :=
1 88
2 11.2
3 1.5
4 4
5 3
6 3.3
```

```
7 3.3
8 3.2
;
param qf :=
1 0
2 3
3 .5
4 0
5 3
6 3
7 3
8 3
;
param Qw :=
1 159
2 16
3 5
4 26.1
5 3.2
6 1
7 4.4
8 5
;
param qw :=
1 5.1
2 0
3 0
4 13.18
5 1
6 0
7 1
```

```
8 0
;
```

C.9 RESULTS EXPERIMENT 3

```
d_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    o_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    supplymax_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    supplymin_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xa_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xc_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xf_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual
    xw_index : Dim=0, Dimen=2, Size=16, Domain=None, Ordered=True, Bounds=None
        Virtual

2 RangeSet Declarations
    I : Dim=0, Dimen=1, Size=2, Domain=Integers, Ordered=True, Bounds=(1, 2)
        Virtual
    J : Dim=0, Dimen=1, Size=8, Domain=Integers, Ordered=True, Bounds=(1, 8)
        Virtual

14 Param Declarations
    M : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
        Key  : Value
        None :    30
```

```
NC : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
    Key  : Value
    None :     8
NS : Size=1, Index=None, Domain=PositiveIntegers, Default=None, Mutable=False
    Key  : Value
    None :     2
Qa : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :     1
      2 :   0.5
      3 :   0.5
      4 :   0.5
      5 :   0.5
      6 :   0.5
      7 :     0
      8 :   0.5
Qc : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    16
      2 :     2
      3 :     3
      4 :   3.5
      5 :     1
      6 :     1
      7 :   0.5
      8 :     1
Qf : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    88
      2 :  11.2
      3 :   1.5
```

```
      4 :     4
      5 :     3
      6 :   3.3
      7 :   3.3
      8 :   3.2
Qw : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   159
      2 :    16
      3 :     5
      4 :  26.1
      5 :   3.2
      6 :     1
      7 :   4.4
      8 :     5
c : Size=16, Index=c_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   120
    (1, 2) :   143
    (1, 3) :   183
    (1, 4) :   163
    (1, 5) :   158
    (1, 6) :   195
    (1, 7) :   193
    (1, 8) :   172
    (2, 1) :    93
    (2, 2) :   116
    (2, 3) :   156
    (2, 4) :   136
    (2, 5) :   131
    (2, 6) :   168
```

```
    (2, 7) :   166
    (2, 8) :   145
m : Size=1, Index=None, Domain=Any, Default=None, Mutable=False
    Key  : Value
    None :     1
o : Size=16, Index=o_index, Domain=Any, Default=None, Mutable=False
    Key    : Value
    (1, 1) :   120
    (1, 2) :   143
    (1, 3) :   183
    (1, 4) :   163
    (1, 5) :   158
    (1, 6) :   195
    (1, 7) :   193
    (1, 8) :   172
    (2, 1) :    93
    (2, 2) :   116
    (2, 3) :   156
    (2, 4) :   136
    (2, 5) :   131
    (2, 6) :   168
    (2, 7) :   266
    (2, 8) :   445
qa : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   0.5
      2 :   0.5
      3 :     0
      4 :   0.5
      5 :     0
      6 :   0.5
```

```
      7 :     0
      8 :   0.5
qc : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   4.1
      2 :     1
      3 :   1.4
      4 :     2
      5 :   0.5
      6 :     0
      7 :     0
      8 :   0.5
qf : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :    13
      2 :     3
      3 :   0.5
      4 :     0
      5 :     1
      6 :     1
      7 :     1
      8 :     1
qw : Size=8, Index=J, Domain=Any, Default=None, Mutable=False
    Key : Value
      1 :   5.1
      2 :     0
      3 :     0
      4 : 13.18
      5 :     1
      6 :     0
      7 :     1
```

```
          8 :     0

5 Var Declarations
    d : Size=16, Index=d_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (1, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 1) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 2) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 3) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 4) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 5) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 6) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 7) :     0 :  None :     1 : False :  True : NonNegativeIntegers
        (2, 8) :     0 :  None :     1 : False :  True : NonNegativeIntegers
    xa : Size=16, Index=xa_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :  None :  None : False :  True : PositiveReals
        (1, 2) :     0 :  None :  None : False :  True : PositiveReals
        (1, 3) :     0 :  None :  None : False :  True : PositiveReals
        (1, 4) :     0 :  None :  None : False :  True : PositiveReals
        (1, 5) :     0 :  None :  None : False :  True : PositiveReals
        (1, 6) :     0 :  None :  None : False :  True : PositiveReals
        (1, 7) :     0 :  None :  None : False :  True : PositiveReals
        (1, 8) :     0 :  None :  None : False :  True : PositiveReals
```

```
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xc : Size=16, Index=xc_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xf : Size=16, Index=xf_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
```

```
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
xw : Size=16, Index=xw_index
    Key    : Lower : Value : Upper : Fixed : Stale : Domain
    (1, 1) :     0 :  None :  None : False :  True : PositiveReals
    (1, 2) :     0 :  None :  None : False :  True : PositiveReals
    (1, 3) :     0 :  None :  None : False :  True : PositiveReals
    (1, 4) :     0 :  None :  None : False :  True : PositiveReals
    (1, 5) :     0 :  None :  None : False :  True : PositiveReals
    (1, 6) :     0 :  None :  None : False :  True : PositiveReals
    (1, 7) :     0 :  None :  None : False :  True : PositiveReals
    (1, 8) :     0 :  None :  None : False :  True : PositiveReals
    (2, 1) :     0 :  None :  None : False :  True : PositiveReals
    (2, 2) :     0 :  None :  None : False :  True : PositiveReals
    (2, 3) :     0 :  None :  None : False :  True : PositiveReals
    (2, 4) :     0 :  None :  None : False :  True : PositiveReals
    (2, 5) :     0 :  None :  None : False :  True : PositiveReals
    (2, 6) :     0 :  None :  None : False :  True : PositiveReals
    (2, 7) :     0 :  None :  None : False :  True : PositiveReals
    (2, 8) :     0 :  None :  None : False :  True : PositiveReals
```

```
1 Objective Declarations
    objective : The objective function
        Size=1, Index=None, Active=True
        Key  : Active : Sense    : Expression
        None :   True : minimize : 120*(xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1]) +
         120*d[1,1] + 143*(xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2]) + 143*d[1,2]
         + 183*(xa[1,3] + xc[1,3] + xf[1,3] + xw[1,3]) + 183*d[1,3] + 163*(xa[1,4]
         + xc[1,4] + xf[1,4] + xw[1,4]) + 163*d[1,4] + 158*(xa[1,5] + xc[1,5]
         + xf[1,5] + xw[1,5]) + 158*d[1,5] + 195*(xa[1,6] + xc[1,6] + xf[1,6]
         + xw[1,6]) + 195*d[1,6] + 193*(xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7])
         + 193*d[1,7] + 172*(xa[1,8] + xc[1,8] + xf[1,8] + xw[1,8]) + 172*d[1,8]
         + 93*(xa[2,1] + xc[2,1] + xf[2,1] + xw[2,1]) + 93*d[2,1] + 116*(xa[2,2]
         + xc[2,2] + xf[2,2] + xw[2,2]) + 116*d[2,2] + 156*(xa[2,3] + xc[2,3]
         + xf[2,3] + xw[2,3]) + 156*d[2,3] + 136*(xa[2,4] + xc[2,4] + xf[2,4]+xw[2,4])
          + 136*d[2,4] + 131*(xa[2,5] + xc[2,5] + xf[2,5] + xw[2,5]) + 131*d[2,5]
          + 168*(xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6]) + 168*d[2,6] + 166*(xa[2,7]
          + xc[2,7] + xf[2,7] + xw[2,7]) + 266*d[2,7] + 145*(xa[2,8] + xc[2,8]
          + xf[2,8] + xw[2,8]) + 445*d[2,8]

10 Constraint Declarations
    supplyAl : Supply requested ammo quantities for j (lower bound)
        Size=8, Index=J, Active=True
        Key : Lower : Body              : Upper : Active
          1 :   0.5 : xa[1,1] + xa[2,1] :  +Inf :   True
          2 :   0.5 : xa[1,2] + xa[2,2] :  +Inf :   True
          3 :   0.0 : xa[1,3] + xa[2,3] :  +Inf :   True
          4 :   0.5 : xa[1,4] + xa[2,4] :  +Inf :   True
          5 :   0.0 : xa[1,5] + xa[2,5] :  +Inf :   True
          6 :   0.5 : xa[1,6] + xa[2,6] :  +Inf :   True
          7 :   0.0 : xa[1,7] + xa[2,7] :  +Inf :   True
```

```
      8 :   0.5 : xa[1,8] + xa[2,8] :  +Inf :   True
supplyAu : Supply requested ammo quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xa[1,1] + xa[2,1] :   1.0 :   True
      2 :  -Inf : xa[1,2] + xa[2,2] :   0.5 :   True
      3 :  -Inf : xa[1,3] + xa[2,3] :   0.5 :   True
      4 :  -Inf : xa[1,4] + xa[2,4] :   0.5 :   True
      5 :  -Inf : xa[1,5] + xa[2,5] :   0.5 :   True
      6 :  -Inf : xa[1,6] + xa[2,6] :   0.5 :   True
      7 :  -Inf : xa[1,7] + xa[2,7] :   0.0 :   True
      8 :  -Inf : xa[1,8] + xa[2,8] :   0.5 :   True
supplyCl : Supply requested cargo quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   4.1 : xc[1,1] + xc[2,1] :  +Inf :   True
      2 :   1.0 : xc[1,2] + xc[2,2] :  +Inf :   True
      3 :   1.4 : xc[1,3] + xc[2,3] :  +Inf :   True
      4 :   2.0 : xc[1,4] + xc[2,4] :  +Inf :   True
      5 :   0.5 : xc[1,5] + xc[2,5] :  +Inf :   True
      6 :   0.0 : xc[1,6] + xc[2,6] :  +Inf :   True
      7 :   0.0 : xc[1,7] + xc[2,7] :  +Inf :   True
      8 :   0.5 : xc[1,8] + xc[2,8] :  +Inf :   True
supplyCu : Supply requested cargo quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xc[1,1] + xc[2,1] :  16.0 :   True
      2 :  -Inf : xc[1,2] + xc[2,2] :   2.0 :   True
      3 :  -Inf : xc[1,3] + xc[2,3] :   3.0 :   True
      4 :  -Inf : xc[1,4] + xc[2,4] :   3.5 :   True
      5 :  -Inf : xc[1,5] + xc[2,5] :   1.0 :   True
```

```
      6 :  -Inf : xc[1,6] + xc[2,6] :   1.0 :   True
      7 :  -Inf : xc[1,7] + xc[2,7] :   0.5 :   True
      8 :  -Inf : xc[1,8] + xc[2,8] :   1.0 :   True
supplyFl : Supply requested fuel quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  13.0 : xf[1,1] + xf[2,1] :  +Inf :   True
      2 :   3.0 : xf[1,2] + xf[2,2] :  +Inf :   True
      3 :   0.5 : xf[1,3] + xf[2,3] :  +Inf :   True
      4 :   0.0 : xf[1,4] + xf[2,4] :  +Inf :   True
      5 :   1.0 : xf[1,5] + xf[2,5] :  +Inf :   True
      6 :   1.0 : xf[1,6] + xf[2,6] :  +Inf :   True
      7 :   1.0 : xf[1,7] + xf[2,7] :  +Inf :   True
      8 :   1.0 : xf[1,8] + xf[2,8] :  +Inf :   True
supplyFu : Supply requested fuel quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xf[1,1] + xf[2,1] :  88.0 :   True
      2 :  -Inf : xf[1,2] + xf[2,2] :  11.2 :   True
      3 :  -Inf : xf[1,3] + xf[2,3] :   1.5 :   True
      4 :  -Inf : xf[1,4] + xf[2,4] :   4.0 :   True
      5 :  -Inf : xf[1,5] + xf[2,5] :   3.0 :   True
      6 :  -Inf : xf[1,6] + xf[2,6] :   3.3 :   True
      7 :  -Inf : xf[1,7] + xf[2,7] :   3.3 :   True
      8 :  -Inf : xf[1,8] + xf[2,8] :   3.2 :   True
supplyWl : Supply requested water quantities for j (lower bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :   5.1 : xw[1,1] + xw[2,1] :  +Inf :   True
      2 :   0.0 : xw[1,2] + xw[2,2] :  +Inf :   True
      3 :   0.0 : xw[1,3] + xw[2,3] :  +Inf :   True
```

```
      4 : 13.18 : xw[1,4] + xw[2,4] :  +Inf :   True
      5 :   1.0 : xw[1,5] + xw[2,5] :  +Inf :   True
      6 :   0.0 : xw[1,6] + xw[2,6] :  +Inf :   True
      7 :   1.0 : xw[1,7] + xw[2,7] :  +Inf :   True
      8 :   0.0 : xw[1,8] + xw[2,8] :  +Inf :   True
supplyWu : Supply requested water quantities for j (upper bound)
    Size=8, Index=J, Active=True
    Key : Lower : Body              : Upper : Active
      1 :  -Inf : xw[1,1] + xw[2,1] : 159.0 :   True
      2 :  -Inf : xw[1,2] + xw[2,2] :  16.0 :   True
      3 :  -Inf : xw[1,3] + xw[2,3] :   5.0 :   True
      4 :  -Inf : xw[1,4] + xw[2,4] :  26.1 :   True
      5 :  -Inf : xw[1,5] + xw[2,5] :   3.2 :   True
      6 :  -Inf : xw[1,6] + xw[2,6] :   1.0 :   True
      7 :  -Inf : xw[1,7] + xw[2,7] :   4.4 :   True
      8 :  -Inf : xw[1,8] + xw[2,8] :   5.0 :   True
supplymax : Maximum quantity from supplier i to consumer j
    Size=16, Index=supplymax_index, Active=True
    Key : Lower : Body : Upper : Active
    (1, 1) : -Inf : xa[1,1] + xc[1,1] + xf[1,1] + xw[1,1] - 30*d[1,1] : 0.0 : True
    (1, 2) : -Inf : xa[1,2] + xc[1,2] + xf[1,2] + xw[1,2] - 30*d[1,2] : 0.0 : True
    (1, 3) : -Inf : xa[1,3] + xc[1,3] + xf[1,3] + xw[1,3] - 30*d[1,3] : 0.0 : True
    (1, 4) : -Inf : xa[1,4] + xc[1,4] + xf[1,4] + xw[1,4] - 30*d[1,4] : 0.0 : True
    (1, 5) : -Inf : xa[1,5] + xc[1,5] + xf[1,5] + xw[1,5] - 30*d[1,5] : 0.0 : True
    (1, 6) : -Inf : xa[1,6] + xc[1,6] + xf[1,6] + xw[1,6] - 30*d[1,6] : 0.0 : True
    (1, 7) : -Inf : xa[1,7] + xc[1,7] + xf[1,7] + xw[1,7] - 30*d[1,7] : 0.0 : True
    (1, 8) : -Inf : xa[1,8] + xc[1,8] + xf[1,8] + xw[1,8] - 30*d[1,8] : 0.0 : True
    (2, 1) : -Inf : xa[2,1] + xc[2,1] + xf[2,1] + xw[2,1] - 30*d[2,1] : 0.0 : True
    (2, 2) : -Inf : xa[2,2] + xc[2,2] + xf[2,2] + xw[2,2] - 30*d[2,2] : 0.0 : True
    (2, 3) : -Inf : xa[2,3] + xc[2,3] + xf[2,3] + xw[2,3] - 30*d[2,3] : 0.0 : True
    (2, 4) : -Inf : xa[2,4] + xc[2,4] + xf[2,4] + xw[2,4] - 30*d[2,4] : 0.0 : True
```

```
        (2, 5) : -Inf : xa[2,5] + xc[2,5] + xf[2,5] + xw[2,5] - 30*d[2,5] : 0.0 : True
        (2, 6) : -Inf : xa[2,6] + xc[2,6] + xf[2,6] + xw[2,6] - 30*d[2,6] : 0.0 : True
        (2, 7) : -Inf : xa[2,7] + xc[2,7] + xf[2,7] + xw[2,7] - 30*d[2,7] : 0.0 : True
        (2, 8) :  -Inf : xa[2,8] + xc[2,8] + xf[2,8] + xw[2,8] - 30*d[2,8] :   0.0 :
    supplymin : Minimum quantity from supplier i to consumer j
        Size=16, Index=supplymin_index, Active=True
        Key : Lower : Body : Upper : Active
        (1, 1) : -Inf : d[1,1] - xa[1,1] - xc[1,1] - xf[1,1] - xw[1,1] : 0.0 : True
        (1, 2) : -Inf : d[1,2] - xa[1,2] - xc[1,2] - xf[1,2] - xw[1,2] : 0.0 : True
        (1, 3) : -Inf : d[1,3] - xa[1,3] - xc[1,3] - xf[1,3] - xw[1,3] : 0.0 : True
        (1, 4) : -Inf : d[1,4] - xa[1,4] - xc[1,4] - xf[1,4] - xw[1,4] : 0.0 : True
        (1, 5) : -Inf : d[1,5] - xa[1,5] - xc[1,5] - xf[1,5] - xw[1,5] : 0.0 : True
        (1, 6) : -Inf : d[1,6] - xa[1,6] - xc[1,6] - xf[1,6] - xw[1,6] : 0.0 : True
        (1, 7) : -Inf : d[1,7] - xa[1,7] - xc[1,7] - xf[1,7] - xw[1,7] : 0.0 : True
        (1, 8) : -Inf : d[1,8] - xa[1,8] - xc[1,8] - xf[1,8] - xw[1,8] : 0.0 : True
        (2, 1) : -Inf : d[2,1] - xa[2,1] - xc[2,1] - xf[2,1] - xw[2,1] : 0.0 : True
        (2, 2) : -Inf : d[2,2] - xa[2,2] - xc[2,2] - xf[2,2] - xw[2,2] : 0.0 : True
        (2, 3) : -Inf : d[2,3] - xa[2,3] - xc[2,3] - xf[2,3] - xw[2,3] : 0.0 : True
        (2, 4) : -Inf : d[2,4] - xa[2,4] - xc[2,4] - xf[2,4] - xw[2,4] : 0.0 : True
        (2, 5) : -Inf : d[2,5] - xa[2,5] - xc[2,5] - xf[2,5] - xw[2,5] : 0.0 : True
        (2, 6) : -Inf : d[2,6] - xa[2,6] - xc[2,6] - xf[2,6] - xw[2,6] : 0.0 : True
        (2, 7) : -Inf : d[2,7] - xa[2,7] - xc[2,7] - xf[2,7] - xw[2,7] : 0.0 : True
        (2, 8) : -Inf : d[2,8] - xa[2,8] - xc[2,8] - xf[2,8] - xw[2,8] : 0.0 : True

43 Declarations: NS NC I J SS CS m M c_index c o_index o qa Qa qc Qc qf Qf qw Qw
xa_index xa xc_index xc xf_index xf xw_index xw d_index d supplyAl supplyAu
supplyCl supplyCu supplyFl supplyFu supplyWl supplyWu supplymin_index supplymin
supplymax_index supplymax objective
# ==========================================================
# = Solver Results                                         =
# ==========================================================
```

```
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 7536.48
  Upper bound: 7536.48
  Number of objectives: 1
  Number of constraints: 97
  Number of variables: 81
  Number of nonzeros: 289
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 83
      Number of created subproblems: 83
  Error rc: 0
  Time: 0.1275949478149414
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
Variable xa
```

```
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.5
    (2, 1) 0.5
    (2, 2) 0.5
    (2, 3) 0.0
    (2, 4) 0.5
    (2, 5) 0.0
    (2, 6) 0.5
    (2, 7) 0.0
    (2, 8) 0.0
Variable xc
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 0.0
    (1, 8) 0.5
    (2, 1) 4.1
    (2, 2) 1.0
    (2, 3) 1.4
    (2, 4) 2.0
    (2, 5) 0.5
    (2, 6) 0.0
```

```
    (2, 7) 0.0
    (2, 8) 0.0
Variable xf
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 1.0
    (1, 8) 1.0
    (2, 1) 13.0
    (2, 2) 3.0
    (2, 3) 0.5
    (2, 4) 0.0
    (2, 5) 1.0
    (2, 6) 1.0
    (2, 7) 0.0
    (2, 8) 0.0
Variable xw
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 1.0
    (1, 8) 0.0
    (2, 1) 5.1
    (2, 2) 0.0
    (2, 3) 0.0
```

```
    (2, 4) 13.18
    (2, 5) 1.0
    (2, 6) 0.0
    (2, 7) 0.0
    (2, 8) 0.0
Variable d
    (1, 1) 0.0
    (1, 2) 0.0
    (1, 3) 0.0
    (1, 4) 0.0
    (1, 5) 0.0
    (1, 6) 0.0
    (1, 7) 1.0
    (1, 8) 1.0
    (2, 1) 1.0
    (2, 2) 1.0
    (2, 3) 1.0
    (2, 4) 1.0
    (2, 5) 1.0
    (2, 6) 1.0
    (2, 7) 0.0
    (2, 8) 0.0
```