UCLouvain

NHH

LOUVAIN
School of Management

Louvain School of Management

and Norwegian School of Economics

LNG Inventory Routing Problem Under Uncertain Weather

Research Master's Thesis submitted by
Xiaozhi Feng

With a view of getting the degrees:
Master of Science in Economics and Business Administration
Master 120 credits in Business Engineering, Supply Chain Management

Supervisor :
Stein W. Wallace

Academic Year [*2019-2020*]

Acknowledgment

In 2019, I learned about the topic of natural gas transportation for the first time on the Land Use course and became very interested in it. Coincidentally, after assessing many topics related to natural gas transportation, I found a topic similar to the one I wanted to explore in the Shipping and Logistics Research Center in NHH. After discussing with my supervisor, Prof. Stein Wallace, and Dr. Xin Wang of Tieto Company, we finally settled the research content on the establishment and application of the stochastic model in LNG inventory routing.

The research process was exciting and creative at the beginning. However, due to the outbreak of the epidemic and various unexpected things that happened to me, the writing process of this master thesis became extremely tortuous. Here, I have to thank my supervisor, Prof. Stein W. Wallace, for helping me during every difficult time along the way. He was the one who gave me confidence and convinced me that I was capable of doing my best to finish this paper when I was about to give up. I would also like to thank Dr. Xin Wang for the useful information he has provided, which has given me a better understanding of the operation mode of the industry. Finally, I would like to thank the doctors and nurses who provided me with all conveniences during my quarantine days to ensure that I could attend classes and finish my studies. It was an unforgettable and unique experience for me.

Xiaozhi Feng

Xiaozhi FENG

Abstract

The inventory routing problem (IRP) in Liquefied Natural Gas (LNG) is one of the representative maritime IRP. In this problem, how to hedge the risk of uncertain voyage time brought by uncertain weather has long been a challenging issue for LNG suppliers. Given high costs in LNG delivery operations and storage, efficient inventory management and scheduling can yield substantial savings. This paper first introduces the LNG supply chain, describing how the uncertain weather conditions influence the shipment and then establishes two stochastic models to find the optimal solution.

The first one focuses on reducing the uncertain influence through speed and ship schedule adjustments when the weather condition is realized in the second stage. In contrast, the second one extends the first model, adding a path choice so that ships can bypass the area with rough weather rather than go through it. A deterministic model is generated at the same time and works as a reference to compare with the two stochastic models. Finally, a real-world case computation is conducted to evaluate the models.

From the computational results, we see that both stochastic models result in higher costs than the deterministic one. However, when adapting the deterministic solution to stochastic settings, the result is quite different. Although the original schedule cost of the deterministic solution is still the lowest, the value of the stochastic part (expected value of extra costs of changing the original schedule under different possible scenarios) is significantly higher than the two stochastic models' solutions, leading to higher total costs. It means that the schedule chosen by the deterministic model is not the most economical one facing the uncertain weather influence.

When comparing the results of the two stochastic models, we find that when the rough weather days are scattered and short, there's nearly no difference between the two models. If the stormy days are concentrated and continuous, the model with path choice has better performance because it can avoid all the negative influences brought by the bad weather. However, when the long detour which can bypass the rough area is too long, this choice becomes meaningless. At this time, the solutions of these two stochastic models become almost consistent.

Table of Contents

Figure List

Table List

List of Abbreviations

Abbreviations in order of appearance in the thesis:

NG	Natural Gas
LNG	Liquefied Natural Gas
BOG	Boil-off Gas
ADP	Annual Delivery Program
SIRP	Stochastic LNG Inventory Routing Problem
IRP	Inventory Routing Problem
ADNOC	Abu Dhabi National Oil Company
MIP	Mixed-Integer Programming
DES	Delivered Ex Ship
SA	Simulated Annealing Algorithm
TEPCO	Tokyo Electric Power Company
SDS	Shipping Delivery Schedule
UAE	United Arab Emirates
LPG	Liquefied Petroleum Gas
FOB	Free on Board

This page intentionally left blank

1. Introduction

Natural gas (NG) is the second-largest energy source in world power generation, "representing 22% of generated power globally and the only fossil fuel whose share of primary energy consumption is projected to grow" (World Energy Council, 2016). In 2018, both global consumption and production increased by over 5%, one of the most substantial growth rates in either gas demand or output for more than three decades (BP Department, 2018). Moreover, the growth on the demand side is mainly concentrated in Asian countries such as China and Japan.

Pipelines are the conventional way of transporting natural gas and are cost-efficient for long-distance transportation. However, to connect suppliers and consumers in different continents and fulfill the trans-ocean demand, for example, from suppliers in North America to customers in China and Japan, shipping NG in the form of liquefied natural gas (LNG) has become a much more convenient and efficient method. By 2018, LNG accounted for 45.7% of the natural gas trade, compared to 30.8% in 2008 (BP Department, 2018). This situation leads to an increased complexity of the LNG supply chain.

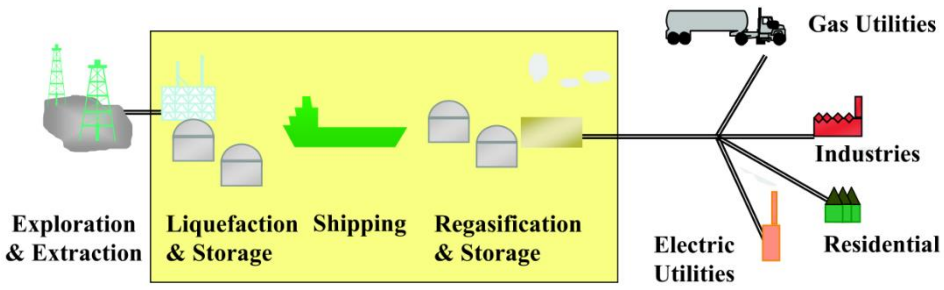

Fig. 1 The LNG Supply Chain

(Source: Roar Grønhaug, 2008)

Fig.1 shows the complete LNG supply chain. Once NG is produced, it is stored in a storage tank in the liquid form at a temperature of -160°C. The volume of NG in the liquefied state is around 1/600 of the volume of NG in the gaseous state. Then, LNG is transported from the production plant to a consumer port by the LNG tanker. During maritime transportation, a certain amount of LNG is vaporized. This boil-off gas (BOG) cannot be delivered to customers and is always considered as a loss. After that, when the LNG tanker arrives at the customer

port, LNG is transformed back to the gaseous state for road-based transportation (Thomas and Dawe, 2003).

In light of these special requirements, shipping NG requires significant capital investments and operating expenses. The capital cost of a liquefaction plant can be as high as 600 dollars per ton per annum production while the capital cost of an LNG vessel ranges between 200 and 400 million dollars. Operational costs are also significant; the cost of daily vessel charters average around 48,800 dollars (Argus, 2020). Thus, the LNG market has been traditionally characterized by relatively rigid and long-term contracts and operations, which always fulfilled with the Annual Delivery Program (ADP).

Typically, these contracts will specify a delivery date and a grace period, as well as the amount of LNG needed each time. During the implementation of agreements, for suppliers, the major concern is whether the ship can deliver goods on time. Due to poor scheduling and uncertainties, such as bad weather, late deliveries can happen from time to time. It will not only increase port costs and operation costs but also affect the subsequent delivery plan. Thus, there is a need for more efficient and advanced planning tools to manage the inventory level and adjust the shipping schedule in time.

In this paper, focusing on tactical planning, we study the process from liquefaction to regasification of the LNG supply chain (highlighted part in Fig.1). The aim is to optimize the LNG inventory and shipping schedule, minimizing the sum of all operating costs and penalty costs from uncertainties, and finding the optimal speed for each vessel in every voyage at the same time. This problem is traditionally classified as a stochastic LNG inventory routing problem (SIRP).

To solve this problem, we first establish a basic IRP model without considering random events to improve the shipping schedule. After that, a stochastic model and its extended version, which take weather influence into account, are presented and compared to the results of the previous one.

The rest of the paper is organized as follows: The literature review and contributions of this thesis will be presented in Section 2. In Section 3, the LNG-SIRP problem is first introduced, then the relevant mathematical formulations and solution methods are given in 3.2. After that, all models will be tested on a real case from Abu Dhabi National Oil Company (ADNOC) in

the computational study in Section 4. Finally, in Section 5, we present some concluding remarks, as well as limitations.

2. Literature Review

The inventory-routing problem (IRP) integrates inventory management, vehicle routing, and delivery scheduling decisions. The origin of this study is rooted in the seminal paper of Bell et al. (1983) published around 35 years ago, which formulated the first IRP as a mixed-integer program to manage industrial gases at customer locations. After decades of development, it has become a relatively well-established research field, and there are several reviews on this issue.

Andersson, Hoff, Christiansen, Hasle, and Løkketangen (2010) distinguish the IRP by road-based or maritime-based first and then classify all literature by planning horizon, demand, topology, routing, inventory, vehicle fleet, and solution approach. Coelho et al. (2014) concentrate more on structural variants and the availability of information on customer demand, focusing on methodologies. Roldan et al. (2017) emphasize on stochastic problems, such as demand and lead times, reviewing solutions to multi-depot IRPs. In this article, maritime-based IRP is the focal point.

Oil, gas, and other chemical goods are the main study objects in maritime IRP problems because of their maritime shipping environment. As one of the representative maritime IRPs, the number of studies on LNG businesses shows an increasing trend from 2009, when the LNG demand increased dramatically. Grønhaug and Christiansen (2009) conduct one of the earliest LNG-IRP studies, setting a mixed-integer programming (MIP) model and formulating it in both arc-flow and path-flow models. In their article, inventories at liquefaction and regasification plants, as well as routing, are considered. Later, Shao et al. (2015), Mutlut et al. (2015), Halvorsen-Weare and Fagerholt (2013), Rakke et al. (2011) and Stålhane et al. (2012) attempt to develop cost-efficient ADPs. Shao, Furman, Goel, and Hoda (2015) develop a hybrid heuristic strategy to improve LNG-IRP solutions proposed by Goel et al. (2012).

From these papers, we can see the significant progress in LNG-IRP in the past three decades. However, these models still have limitations. First, they assume that the shipping speed is fixed, which means that each route has a given number of days to travel. In practice, an LNG tanker can often adjust its speed to shorten or lengthen shipping days to match production and inventory levels at suppliers and reduce additional costs due to untimely arrival. Siti et al. (2015) and Wang et al. (2019) both present deterministic IRP models that can optimize the speed. The former provides a shipping schedule that considers port dwelling time, while the latter analyzes the fuel consumption change caused by the speed change and its impact on the

environment, and the option of speed changing affords the ship operator more operational flexibility. After taking the shipping speed as a variable, LNG IRP becomes a mixed-Integer non-linear program, and the solution procedures become complicated and diversified.

Second, most of these models are deterministic, not considering uncertainties. Yet, in reality, the LNG-IRPs are influenced by many uncertain factors. Stochastic demand, travel time, boil-off rate and fluctuating market price, etc. can challenge the accuracy of established models, weakening the application of models in real industries. Using average numbers is the most straightforward and simple way to deal with uncertainties (Bell et al., 1983; Federgruen and Zipkin, 1984). In recent years, more advanced methods have been developed. A three-stage stochastic programming technique was proposed by Zhang et al. (2017) to design the LNG supply system. They mainly optimize the infrastructure development and inventory-routing decisions, and the proposed methodology was evaluated in a real case study in China. Cho et al. (2018) introduce an inventory-routing problem (IRP) to design the LNG supply system under uncertain weather conditions. They applied a stochastic mixed-integer programming (MIP) model to maximize the total expected revenue and concurrently minimize the total disruption cost caused by the uncertain effects of the dust storm.

Finally, the clear trend changing from long-term to short-term contracts puts forward new requirements for planning. Reroute and reassignment have also become an increasingly important issue in the context of short-term and high variably contracts.

Hence in this paper, a two-stage stochastic model will be developed considering uncertain weather conditions and variable shipping speed. Besides, an extended model takes into account additional options for ships facing unpleasant weather. Assume that there is no longer only one path between two ports; instead, longer paths that would not frequently be used are among the ship's options. When the unpleasant weather is realized, reassignment and speed changes are not the only ways to optimize -- ships can choose to avoid rough areas as they do in real life. This problem is an NP-hard problem that is not easy to solve. Thus, we use the simulated annealing heuristic algorithm to solve the model. Finally, the computation of a real-world case from Abu Dhabi National Oil Company is conducted to do the evaluation.

3. Model

3.1 Problem Description

The problem considered in this paper is a tactical LNG IRP for an LNG producer who is in charge of a series of vessels. The trading mode used here is Delivered Ex Ship (DES), which means that besides LNG production and storage, the producer is also responsible for the transportation to customers that are located all over the world. In other words, it is the producer who pays for all transportation and insurance until the ship has arrived at the port of destination. The customer then takes over the goods and assumes all costs and risks afterward.

The producer usually controls a heterogeneous fleet of vessels to transport the LNG. Generally, however, the ship sizes in the same supplier do not differ much, so in this paper, fleets are considered homogeneous, meaning that all the vessels are non-dedicated and have the same capacity. At the start (end) of the planning horizon, each LNG vessel has an initial (ending) position that can be an artificial port or a position at sea. When the order is confirmed, the producer will arrange for a vessel to depart from its initial position to the producer port to load LNG. In this paper, we assume that all the ships are fully loaded.

However, the ship's departure date depends not only on the order date but also on the LNG inventory at the producer port. Because of the given capacity of storage tanks in the liquefication plant, there are upper bound and lower safety bound for inventory, which are usually treated as hard requirements that cannot be violated. As a result, ships can only carry out loading operations within stock limits. In practice, when the inventory cannot meet the loading demand, the ship needs to wait at the port until the shipment requirement is satisfied. What's more, if multiple ships arrive at the producer port at the same time, or if one ship enters the port while another is loading, all vessels must queue up to wait because of the berth constraint.

Demands of all customers in each month are known in advance according to their contracts, which also specify the cargo discharging time window in each port call. Beyond that, customers usually allow a grace period to each order in case the ship is delayed or arrives early due to an unexpected incident on the way. But the producer will have to pay the extra penalties

for the grace period outside the discharging time window. Fig.2 shows the simplified relationship between the grace period and penalty costs. Ships will not receive any extra charges if they arrive at the destination within the allowable discharging time window. However, when they arrive earlier or later than the time window but still in the grace period, the penalty will increase with the length of time.

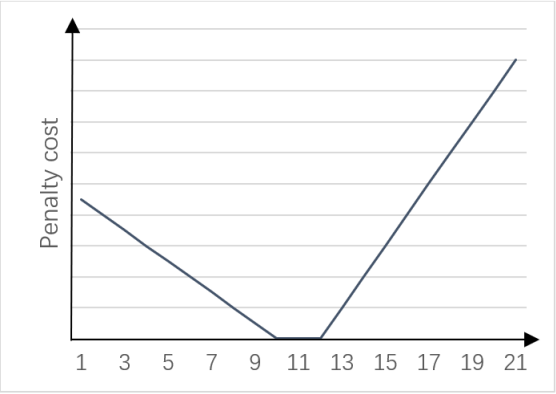

Fig. 2 Time Window and Penalty

Specially, many ports levy lower penalties on early arrivals than late arrivals. This may be because, compared with an earlier arrival, the delay has a more significant impact on terminals' follow-up scheduling, such as poor inventory management in customer port and delayed road-based transportation after regasification. In our model, arriving within the grace period can be deemed as a hard constraint, since in reality, if a ship cannot arrive within the grace period, the supplier would choose to buy a spot cargo than deliver it.

The seasonality of LNG demand is not significant in this context because the planning period is relatively short. Similarly, this is also the reason why the production rate is regarded as a constant in this paper. On the one hand, it is unlikely that a plant will adjust its productivity every day or every 12 hours since each operation has a time lag. On the other hand, frequent changes in overall factory productivity in the short term will increase unnecessary wastes of human resources.

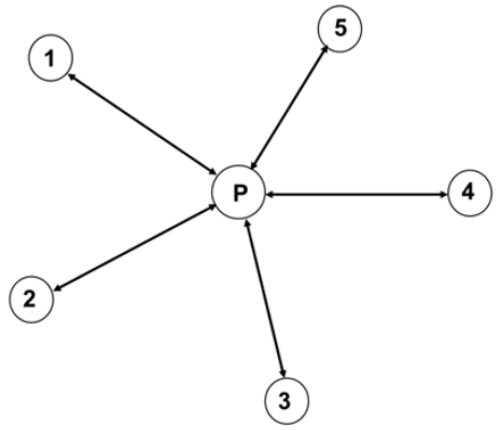

Fig. 3 Illustration for ship routes
(source: Halvorsen-Weare & Fagerholt, 2010)

For ships, within this problem, they only have two routes: An initial delivery route connects the liquefaction plant to a regasification plant, and a return route follows the reverse order (see Fig.3). The producer needs to assign the optimal speed to each ship's two routes before its departure.

Here, the speed selected for each ship will highly influence the transportation costs because of the relationship between speed and fuel consumption. In this paper, we use the empirical

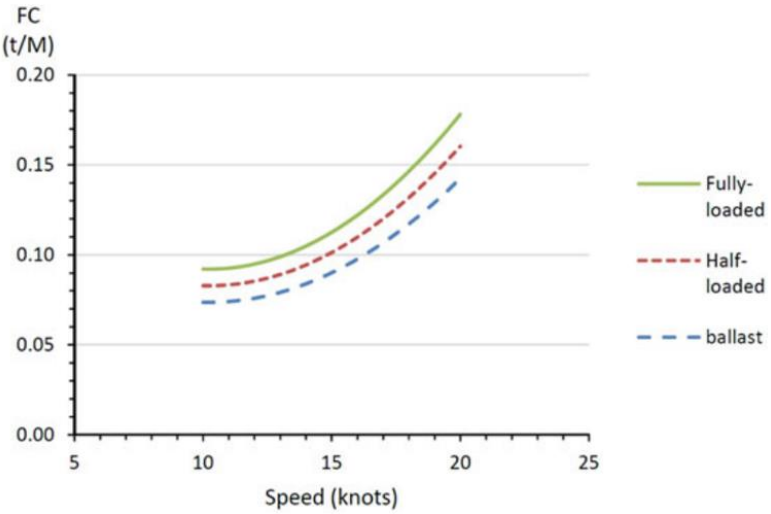

Fig. 4 Fuel Consumption Characteristics (Source: Wang et al.,2019)

function put up by Wang et al. (2019), which is $FC = (Ak^2 + Bk + C) \times (0.8 + 0.2\rho)$, where FC is the fuel consumption rate in tonnes (t) per traveled nautical mile, k is the shipping speed within the ship's feasible speed range, and ρ takes its value between 0% (ballast) and 100% (fully loaded). Fig.4 depicts this relationship.

Uncertain weather conditions usually have two ways to influence shipping. The first one is influencing the speed of vessels during a given number of days. For example, when a ship is traveling in days with high wind waves and strong winds, its speed will partially be offset (or increased if the wind and ship have the same direction) by this event, leading to shipping delay. In other words, when the ship is traveling at 15 knots, the rough weather may reduce the actual speed of the ship to only 11 knots or even less. The second one is suspending the shipping to avoid a storm or other extreme weather. This event seldom happens, and if it does happen, the producer may directly choose to buy spot cargoes or negotiate with customers for other solutions. These are outside the scope of speed and routing optimization problem. Thus, in this paper, we will not consider the second case.

The total costs in the planning period consist of transportation costs, inventory costs, reroute costs, and penalties that come from any other constraint violations. For the stochastic model, there are seven categories of decision variables involved: 1) the vessel assignment to each travel route, 2) the speed of each ship, 3) the changed speed due to wind and waves, 4) the inventory schedule per time period, 5) departure and arrival dates of an incoming and outgoing vessel at each port, 6) the number of unpunctual periods (include early arrive and delay), 7) The number of reassigned vessels.

3.2 Mathematical Model

3.2.1 Basic model – Model I

First, we formulate the IRP basic model without considering weather conditions. In the underlying network for the mathematical model, let V be the set of available ships. An origin node $o(v)$ at the beginning of the planning horizon can geographically represent an artificial point at sea, denoting the origin position of a given vessel, and the same to the destination node $d(v)$ at the end of the horizon. Each ship needs to start from the starting point $o(v)$ to the supplier to load the cargo, and then start the delivery task for the whole planning period. At the end of the planning horizon, ships still need to return to their endpoints $d(v)$ because the supplier ports cannot accommodate so many LNG ships.

Each port is represented by an index i, and the set of ports is given by P (including $o(v)$ and $d(v)$). Subsets P^s and $P^c \subseteq P$ consists of the supplier port and all customer ports, respectively. Each port can be visited several times during the planning horizon, and M_i is the set of possible calls at port i. The port call number is represented by an index m, and $|M_i|$ is the last possible call at port i.

The set of nodes in the flow network represents the set of port calls, and each port call is specified by $(i, m), i \in P, m \in M_i$. Finally, A_v contains all feasible arcs for ship v, which is a subset of $\{ i \in P, m \in M_i \} \times \{ i \in P, m \in M_i \}$. Set T, indexed by t, contains all periods starting from time 1 to time T. Other parameters and variables are listed as follows.

Parameters	
Q	Capacity of vessels
R	Production rate per time period
C^{inv}	Inventory cost per time period
C^{fuel}	Price of bunker fuel per ton
I_{min}, I_{max}	Minimum and Maximum inventory level in storage tanks
D_{ij}	Distance from port i to port j
C^{early}	Penalty for early arrive per time period
C^{delay}	Penalty for shipping delay per time period
T_{min}, T_{max}	Maximum number of days allowed in customer ports for early arrival/delay (grace period)

t_{im}^{E}	Expected arrival time
T^{s}	Service time (loading/ unloading) in each port
K_{min}, K_{max}	Minimum and Maximum traveling speed for a given vessel
$I^{initial}$	The initial inventory at the beginning of the planning period
M	A large enough positive number
Decision Variables	
K_{imjnv}	Traveling speed from the node (i,m) to (j,n) by ship v
I_t	LNG inventory in storage tanks on a given time period t, bounded by minimum and maximum value I_{min}, I_{max}
x_{imjnv}	Binary, equals 1 if ship v sails from the node (i,m) directly to the node (j,n), and 0 otherwise
t_{im}	time variable, the time at which service (loading/unloading) begins at node (i,m)
y_{im}, z_{im}	1 if the service start time at node (i,m) is earlier/later than inner time window respectively

Table. 1 Sets, Parameters, and Variables

The fuel consumption rate of each ship can be a specific function of its speed and payload. Let D_{ij} be the sailing distance from node i to node j. The variable K_{imjnv} defines the speed of travel from the node (i,m) to node (j,n) irrespective of ship chosen. Thus, the traveling time between a supplier port and a customer port can be computed by D_{ij}/K_{imjnv}. The non-linear function $C_v(K_{imjnv}, l)$, defined on the speed interval $[K_{min}, K_{max}]$, represents the variable transportation costs per unit of distance for a ship sailing at speed K_{imjnv} with load l on board. Because of our full load assumption, load l in this paper only has two choices, 100%, and 0%, corresponding to fuel consumption $C_v(K_{imjnv})$ and $80\% * C_v(K_{imjnv})$ according to the function mentioned in Section 3.1.

The mathematical problem formulation is defined as follows:

Objective Function:

$$\text{Min} \quad inventory\ costs + transportation\ costs + unpunctual\ penalty \tag{1}$$

$$inventory\ costs = \sum_{t \in T} I_t * C^{inv} \tag{2}$$

$$transportation\ costs = \sum_{v\in V}\sum_{(i,m,j,n)\in A_v} C_v\left(K_{imjnv}\right) * D_{ij} * x_{imjnv} \quad (3)$$

According to the relationship between the shipping speed and fuel consumption mentioned above, laden and ballast voyages bring different fuel consumption costs due to different loads. For a laden voyage, ships are in full-load, so:

$$C_v\left(K_{imjnv}\right) = C^{fuel} * \left(0.0019{K_{imjnv}}^2 + 0.045K_{imjnv} - 0.3739\right) \quad (4)$$

The relationship between ballast voyage, which corresponds to no-load, and bunker fuel consumption is shown below:

$$C_v\left(K_{imjnv}\right) = C^{fuel} * \left(0.0019{K_{imjnv}}^2 + 0.045K_{imjnv} - 0.3739\right) * 0.8 \quad (5)$$

$$unpunctual\ penalty = \sum_{i\in P^c}\sum_{m\in M_i}(t_{im}^E - t_{im}) * (y_{im} * C^{early} + C^{delay} * z_{im}) \quad (6)$$

Constraints:

$$\sum_{v\in V}\sum_{j\in P}\sum_{n\in M_j} x_{imjnv} = 1, (i,m,j,n) \in A_v, v \in V \quad (7)$$

$$\sum_{j\in P^s}\sum_{n\in M_j} x_{o(v)1jnv} = 1, v \in V \quad (8)$$

$$\sum_{i\in P^s}\sum_{m\in M_i} x_{imjnv} - \sum_{i\in P^s}\sum_{m\in M_i} x_{jnimv} = 0, j \in P^c, n \in M_i, v \in V \quad (9)$$

$$\sum_{i\in P^s}\sum_{m\in M_i} x_{imd(v)1v} = 1, v \in V \quad (10)$$

$$I_0 = I^{initial} \quad (11)$$

$$I_{t_{im}} = I_{t_{i(m-1)}} + R\left(t_{im} - t_{i(m-1)}\right) - Q, i \in P^s, m \in M_i \quad (12)$$

$$I_{max} - I_{t_{i(m-1)}} \geq R\left(t_{im} - t_{i(m-1)}\right), i \in P^s, m \in M_i \quad (13)$$

$$I_{min} \leq I_t \leq I_{max}, t \in T \quad (14)$$

$$x_{imjnv}\left(t_{im} + T^s + \frac{D_{ij}}{K_{imjnv}} - t_{jn}\right) = 0, (i,m,j,n) \in A_v, v \in V \quad (15)$$

$$t_{i(m-1)} + T^s \leq t_{im}, i \in P, m \in M_i \quad (16)$$

$$T_{min} \leq t_{im} - t_{im}^E \leq T_{max}, i \in P^c, m \in M_i \quad (17)$$

$$K_{min} \leq K_{imjnv} \leq K_{max}, (i,m,j,n) \in A_v, v \in V \quad (18)$$

$$x_{imjnv} \in \{0,1\} \quad (19)$$

$$t_{im} \geq 0, i \in P, m \in M_i \quad (20)$$

$$y_{im} \in \{0,1\} \quad (21)$$

$$z_{im} \in \{0,1\} \quad (22)$$

$$y_{im} + z_{im} = 1, i \in P^c, m \in M_i \quad (23)$$

$$t_{im}^E - t_{im} > M(1 - y_{im}), i \in P^c, m \in M_i \quad (24)$$

$$t_{im} - t_{im}^E > M(1 - z_{im}), i \in P^c, m \in M_i \quad (25)$$

The objective function (1)-(6) describes the total cost to be minimized, which consists of three parts: total inventory costs during the planning period, transportation costs, and penalty costs for early arrival or shipping delay. The non-linear function of fuel consumption $C_v(K_{imjnv})$ in equation (4) and (5) comes from Wang et al. (2019), in which they studied a vessel with 56800 dwt and came up with the formula. As for the penalty of early arrival and shipping delay, the common sense is that it is better to arrive early than late. Although in both situations, extra port charges should be paid, shipping delay may cause continuous delays in subsequent plans, leaving the demand gap at the customer port too large. In contrast, early arrival at the port will make the follow-up plan more flexible to some extent, and the penalty is only the inventory scheduling fee of the customer ports and the bunker fuel costs that may exist in waiting for unloading.

Constraint (7) ensures that we can make one port call at most once. Constraints (8)-(10) set the shipping flow, giving the initial and end position of a vessel, letting the ship return to the supplier port as soon as it unloads cargoes at the customer port and wait for the next delivery task. Constraint (11) gives the initial stock at the start of the planning period. Constraints (12)-(14) are inventory constraints. Inequation (12) keeps the balance of inventory between two continuous port call at the supplier. Here, the demand at each customer port is an integer multiple of the number of ships. It is not hypothetical to assume that; It is common to order by the number of vessels in actual transactions.

There is an unstable boil-off rate in the process of sailing, causing a part of the liquefied natural gas to volatilize, so it is unrealistic to ask for the exact quantity of the cargo in each order. Therefore, the port of the customer takes the number of ships as the order demand and then gives a minimum annual discharge quantity in the annual contract to ensure that their demands can be met.

Constraint (14) bounds the inventory quantity, and constraint (13) links the upper limit of inventory with the service start time. The scheduling of the route is taken into account in constraints (15) and (17). Constraint (16) considers the berth limits, and only allow one ship to (un)load in one port. The variables domains are given in constraints (18)-(22). Constraints (23)-(25) make sure that if the vessel cannot arrive on time, it will be fined for being late or early.

The basic model can improve the scheduling and give the optimal speed of each ship while minimizing the cost.

3.2.2 Stochastic Models

Basic Stochastic Model – Model II

In this section, a two-stage stochastic model that takes into account the effects of uncertain weather is established. Within the model, the first stage mainly works on the original delivery plan, and the second stage, when the impact of weather conditions is realized, will adjust the shipping speed and reschedule the ships if needed. The objective function under this situation includes several more parts about the stochastic influence. Except for minimizing the original schedule costs, minimizing the stochastic effects is also an important task, and we hope to find a shipping schedule that can tolerate different weather as much as possible.

The underlying assumption is that, based on the weather forecast available to the supplier, each ship will know the impact of the weather event on its way to port on the day of departure. Here, the set of weather disruption scenarios Ω indexed by w is assumed discrete and finite. Besides, the time constraint is relaxed to avoid no solution situations, which means that the arrival time could outside the grace period, but a very large penalty E which equals to the total value of the shipment will be proposed to motivate shipping to reschedule and reassignment when grace period limits are violated.

Additional parameters and random elements which are necessary are listed as follow, and the formulation will be changed correspondingly:

Parameters	
E	Penalty when ships cannot arrive within the grace period
Variables	
t_{im}^{w+}, t_{im}^{w-}	time variables, describe how much time the second stage rescheduled arrival time is earlier (later) at the node (i,m) under scenario w
u_{imjnv}^{w}	Binary, 1 if the origin ship assignment cannot reach the destination within the grace period and decide to change the ship used under scenario w, 0 otherwise
$k_{imjnvw}^{+}, k_{imjnvw}^{-}$	The speed change (increase/decrease) on route (i,m,j,n) after rescheduling under scenario w by ship v
I_t^{w+}, I_t^{w-}	Inventory change under scenario w in time t
y_{im}^{w}, z_{im}^{w}	Binary, 1 if the service start time at node (i,m) is earlier/later than inner time window under scenario w respectively
o_{im}^{w}	1 if the start time at node (i,m) is outside the grace period under scenario w
Random elements	
diff_{imjnv}^{w}	The impacts of a weather event on speed on the route (i,m,j,n) under scenario w of ship v
ξ_{imjnv}^{w}	The length of the event time that ship v experiences on the route (i,m,j,n) under scenario w

Table. 2 Parameters and Variables in Basic Stochastic Model

Objective Function:

$$\text{Min} \quad I_t * C^{inv} + \sum_{w\in\Omega} p_w \sum_{v\in V} \sum_{(i,m,j,n)\in A_v} C_v\left(K_{imjnv} + k_{imjnvw}^{+} + k_{imjnvw}^{-}\right) * D_{ij} * \left(x_{imjnv} + u_{imjnw}^{v\prime}\right) + \sum_{w\in\Omega} p_w \sum_{i\in P^c} \sum_{m\in M_i} \left(t_{im}^{E} - \left(t_{im} + t_{im}^{w+} + t_{im}^{w-}\right)\right) * \left(y_{im}^{w} * C^{early} + C^{delay} * z_{im}^{w}\right) + \sum_{w\in\Omega} p_w \left(\sum_{i\in P^c} \sum_{m\in M_i} o_{im}^{w} * \text{E} + \left(I_t^{w+} + I_t^{w-}\right) * C^{inv}\right) \tag{1.1}$$

Constraints:

$$\sum_{v,v\prime\in V} \sum_{j\in P} \sum_{n\in M_j} \left(x_{imjnv} + u_{imjnw}^{v\prime}\right) = 1, (i,m,j,n) \in A_v, v \in V \tag{7.1}$$

$$I_{t_{im}} + I_{t_{im}}^{w+} + I_{t_{im}}^{w-} = I_{t_{i(m-1)}} + I_{t_{i(m-1)}}^{w+} + I_{t_{i(m-1)}}^{w-} + R\left((t_{im} + t_{im}^{w+} + t_{im}^{w-}) - (t_{i(m-1)}^{w} + t_{i(m-1)}^{w+} + t_{i(m-1)}^{w-})\right) - Q, i \in P^s, m \in M_i, w \in \Omega \quad (12.1)$$

$$I_{max} - I_{t_{i(m-1)}} + I_{t_{i(m-1)}}^{w+} + I_{t_{i(m-1)}}^{w-} \geq R\left((t_{im} + t_{im}^{w+} + t_{im}^{w-}) - (t_{i(m-1)} + t_{i(m-1)}^{w+} + t_{i(m-1)}^{w-})\right), i \in P^s, m \in M_i,\ w \in \Omega \quad (13.1)$$

$$(x_{imjnv} + u_{imjnw}^{v\prime}) * \left(t_{im} + t_{im}^{w+} + t_{im}^{w-} + T^s + \frac{D_{ij} - \xi_{imjnv}^{w}\left(K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-} - diff_{imjnv}^{w}\right)}{K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-}} + \xi_{imjnv}^{w} - (t_{jn} + t_{jn}^{w+} + t_{jn}^{w-})\right) = 0,\ (i,m,j,n) \in A_v, v \in V, w \in \Omega \quad (15.1)$$

$$t_{im}^{w} + T^s + \frac{D_{ij} - \xi_{imjnv}^{w}\left(K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-} - diff_{imjnv}^{w}\right)}{K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-}} + \xi_{imjnv}^{w} - \left(t_{jn}^{E} + T_{max}\right) > M\left(1 - u_{imjnw}^{v\prime}\right),\ (i,m,j,n) \in A_v, v \in V, w \in \Omega \quad (26)$$

$$x_{imjnv} + u_{imjnw}^{v\prime} = 1, (i,m,j,n) \in A_v, v, v' \in V, w \in \Omega \quad (27)$$

$$t_{i(m-1)} + t_{i(m-1)}^{w+} + t_{i(m-1)}^{w-} + T^s \leq t_{im} + t_{im}^{w+} + t_{im}^{w-}, i \in P, m \in M_i, w \in \Omega \quad (16.1)$$

$$T_{min} - (t_{im} + t_{im}^{w+} + t_{im}^{w-}) - t_{im}^{E} > M(1 - o_{im}^{w}), i \in P^c, m \in M_i, w \in \Omega \quad (28)$$

$$(t_{im} + t_{im}^{w+} + t_{im}^{w-}) - t_{im}^{E} - T_{max} > M(1 - o_{im}^{w}), i \in P^c, m \in M_i, w \in \Omega \quad (29)$$

$$y_{im}^{w} + z_{im}^{w} = 1, i \in P^c, m \in M_i \quad (23.1)$$

$$t_{im}^{E} - (t_{im} + t_{im}^{w+} + t_{im}^{w-}) > M(1 - y_{im}^{w}), i \in P^c, m \in M_i \quad (24.1)$$

$$(t_{im} + t_{im}^{w+} + t_{im}^{w-}) - t_{im}^{E} > M(1 - z_{im}^{w}), i \in P^c, m \in M_i \quad (25.1)$$

The objective function changes from equation (1) to (1.1), adding the outside grace period penalty, and changing the unpunctual costs according to different scenarios. The second term of this function gives the new transportation costs if rescheduling is required. Constraints

which contain scenario-related variables are updated based on the deterministic model, such as constraints (12.1) and (13.1). When coming across strong wind waves, there will be a difference between actual speed and ship speed. The random variable diff_{imjn}^{w} gives this difference. Within the event area, the actual shipping speed is then $(K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-} - \text{diff}_{imjnv}^{w})$ and the total length of time the ship experienced in a rough weather event is ξ_{imjnv}^{w}. Thus, the actual sailing time on route (i, m, j, n) consists of two parts: sailing time in the unaffected area $\frac{D_{ij} - \xi_{imjnv}^{w}\left(K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-} - diff_{imjnv}^{w}\right)}{K_{imjnv} + k_{imjnv}^{w+} + k_{imjnv}^{w-}}$ and the time length of rough weather event ξ_{imjnv}^{w}. Constraint (15.1) reflect this relationship with the new speed. Constraints (26)-(27) illustrate that ship reassignment is triggered only if the origin scheduling fails to meet the time window limits. Constraints (28) and (29) define the binary o_{im}^{w}, giving the conditions when penalty E should be counted. The remaining constraints in the deterministic model will be followed.

Extended Stochastic Model – Model III

In the two models above, the supplier is determined to have only one path to each port. Nevertheless, in practice, ships can take long detours to avoid unpleasant areas. So, we are going to extend model II to consider the decisions ships make in the face of the rough weather event when there are two different lengths of paths between the supplier and the customers, among which the longer one can avoid rough weather effectively. R is defined as the set of paths within the route (i, m, j, n).

Parameters	
E	Penalty when ships cannot arrive within the grace period
Variables	
K_{imjnvr}	The speed on route (i, m, j, n) by path r
$k_{imjnvr}^{w+}, k_{imjnvr}^{w-}$	The speed change (increase/decrease) on route (i, m, j, n) path r after rescheduling under scenario w
x_{imjnr}^{v}	Binary, equals 1 if ship v sails on route (i, m, j, n) by path r
$u_{imjnr'}^{v'w}$	Binary, 1 if the origin ship assignment cannot reach the destination within the grace period and require to reschedule the path or ship under scenario w, 0 otherwise
Random elements	

diff_{imjnvr}^{w}	The speed change on the path r in route (i,m,j,n) if there are rough weather days under scenario w
ξ_{imjnr}^{vw}	The length of the rough event that ship v experiences on the path r in route (i,m,j,n) under scenario w

Table. 3 Parameters and Variables in the Extended Model

Objective Function:

$$\text{Min} \quad I_t * C^{inv} + \sum_{w\in\Omega} p_w \sum_{v\in V} \sum_{(i,m,j,n)\in A_v} C_v\left(K_{imjnvr} + k_{imjnvr}^{w+} + k_{imjnvr}^{w-}\right) * D_{ij} * \left(x_{imjnr}^{v} + u_{imjnr}^{v'w}\right) + \sum_{w\in\Omega} p_w \sum_{i\in P^c} \sum_{m\in M_i} \left(t_{im}^{E} - \left(t_{im} + t_{im}^{w+} + t_{im}^{w-}\right)\right) * \left(y_{im}^{w} * C^{early} + C^{delay} * z_{im}^{w}\right) + \sum_{w\in\Omega} p_w \left(\sum_{i\in P^c} \sum_{m\in M_i} o_{im}^{w} * \text{E} + \left(I_t^{w+} + I_t^{w-}\right) * C^{inv}\right) \tag{1.2}$$

All the constraints are the same as in model II, except that the path index is added to the variable described in the Table. 3.

3.2.3 Solution Method

In this section, we use the simulated annealing (SA) algorithm to solve the NP-hard non-linear problem. The SA algorithm starts from a given random initial state, and on each iteration, generates a new neighbor state. If the new state is better, then accept that state as a new solution. If it is worse, then the algorithm will use a probability function to decide whether to accept that solution or not. It is this characteristic that makes the SA algorithm occasionally accepts worse states, enabling the algorithm to avoid being limited to local optimality.

The probability of choosing a worse solution is controlled by the temperature parameter, which starts large but decreases over time. It is analogous to temperature in an annealing system. When the temperature is high, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely (Teukolsky, Vetterling, and Flannery, 1992). The pseudo-code for a SA algorithm is shown below:

Algorithm 1 SA algorithm structure
Define a high temperature T
Define a cooling schedule T(it), e.g. T=alpha T
Define an energy function S

```
Define current_model initial state
    While (not converged)
        new_model = random
        Delta_S = S(new_model)-S(current_model)
        If (Delta_S < 0) current_model = new_model
    Else with probability P = e^(-Delta_S/T) : current_model = new_model
T=alpha T
```

The Logic for Scheduling

For model I, the deterministic model, the energy function is the objective function (1), and it searches for a better solution globally until the temperature reaches its lower bound. However, the solution finally obtained is not necessarily the optimal one because of the attributes of heuristic. After setting the temperature parameter, the most important part is defining the search scheme, considering the shipping flow and all the constraints. Fig.5 shows the logic of scheduling each order.

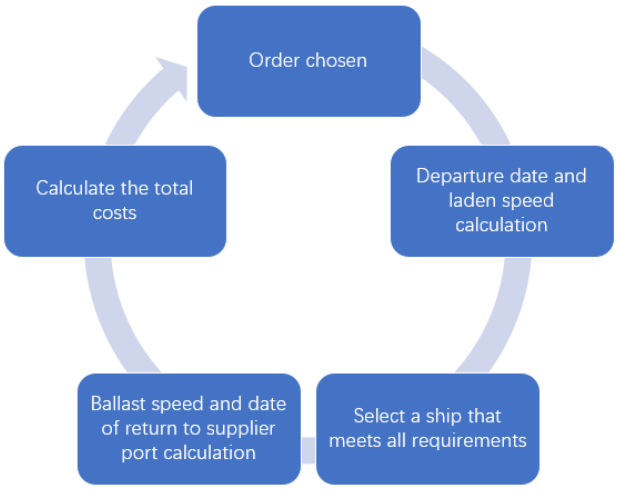

Fig. 5 Scheduling Logic for each order

In general, the order to serve is firstly selected according to the length of the lead time, then the feasible departure date and the corresponding laden speed will be calculated and assigned to it.

Normally, the departure date of the laden voyage is constrained by the intersection of the following two date sets: the first one is the LNG inventory available days. A date is deemed to be available only if the LNG volume on that day is larger than the sum of the ship capacity and storage tank's lower limit, and does not exceed the inventory ceiling. Therefore, under this constraint, the set of available dates ranges from the first day when the storage at the supplier port meets the loading requirements to the day when the LNG storage reaches the upper limit. The second set is days within the permitted time window. For a supplier, if the ship can arrive within the discharge time window, the unpunctual penalty can be avoided. So, we will first control the time limit tightly, and then calculate the departure time range if the ship wants to

arrive at the customer port on time. The earliest (latest) departure date can be obtained by subtracting the longest (shortest) sailing time and loading time from the earliest (latest) date in the discharging time window.

If the intersection of these two sets is not empty, it means that there are dates that meet these constraints, then a day is randomly selected from these dates as the departure date. When the intersection is empty, we should relax the time limit, recalculating the departure time range based on the grace period. Again, after two new date sets are obtained, a day is randomly selected from the date intersection. If the intersection is still empty, it is considered a planning failure.

After deciding the laden attributes, we need to find a suitable ship for this trip. If the supplier has a ship available on the date of departure, then directly use that ship, and if there are no idle ships near the supplier port, then estimate the fastest return dates for all ships assigned for unfulfilled orders. Then, randomly choose a ship that can complete its voyage before the departure date of the new order.

Each random combination choice of departure dates and ships results in different schedules and total costs. The SA algorithm will almost find the best one within the given iteration times.

To note that the departure date of the ballast voyage depends on the arrival time of the laden voyage, so there is no need for extra estimation. The only thing to determine is the ballast speed, which will influence the date of return to the supplier port. During the planning period, a good ballast speed for a ship should make the ship possible to return to the supplier port before the next voyage, and minimizing the transportation costs with all previous constraints. The pseudo-code of this logic is given in Algorithm 2.

Algorithm 2 Ship and Inventory Scheduling

```
Input all the data needed
Select the order to be served in the time order
if multiple orders on the same day
        randomly select one to serve
        end if
   while True:
```

Calculate maximum and minimum sailing times: $(\frac{D_{ij}^{r}}{K_{min}}, \frac{D_{ij}^{r}}{K_{max}})$

//Calculate the date when there is plenty of inventory for shipping

$t_{im}^{Imin} \leftarrow t_{i(m-1)} + (Q + I_{min} - I_{t_{i(m-1)}})/R$

// Calculate the date when inventory in supplier reaches the upper limit.

$t_{im}^{Imax} \leftarrow t_{i(m-1)} + (I_{max} - I_{t_{i(m-1)}})/R$

set $\boldsymbol{A}$ range $\leftarrow (t_{im}^{Imin}, t_{im}^{Imax})$

// Calculate the departure dates range when the ship can arrive at the customer's port on time

$t_{im}^{min} \leftarrow t_{jn}^{E} - T^{s} - \frac{D_{ij}}{K_{min}}$

$t_{im}^{max} \leftarrow t_{jn}^{E} - T^{s} - \frac{D_{ij}}{K_{max}}$

set $\boldsymbol{B}$ range $\leftarrow (t_{im}^{min}, t_{im}^{max})$

if $\boldsymbol{A} \cap \boldsymbol{B} \neq \emptyset$

Choose a departure date t_{im} from the intersection randomly

$K_{imjn} \leftarrow \max(K_{min}, \min\left(\frac{D_{ij}}{t_{jn}^{E} - t_{im} - T^{s}}, K_{max}\right))$

else:

// Calculate the earliest and latest departure dates based on the grace period

$t_{im}^{min} \leftarrow t_{jn}^{E} + T_{min} - T^{s} - \frac{D_{ij}}{K_{min}}$

$t_{im}^{max} \leftarrow t_{jn}^{E} + T_{max} - T^{s} - \frac{D_{ij}}{K_{max}}$

set $\boldsymbol{C}$ range $\leftarrow (t_{im}^{min}, t_{im}^{max})$

if $\boldsymbol{A} \cap \boldsymbol{C} \neq \emptyset$

Choose a departure date t_{im} from the intersection randomly

$K_{imjn} \leftarrow \max(K_{min}, \min\left(\frac{D_{ij}}{t_{jn}^{E} - t_{im} - T^{s}}, K_{max}\right))$

Unpunctual time length $\leftarrow (t_{im} + \frac{D_{ij}}{K_{imjn}} - t_{jn}^{E})$

else:

Assign a large penalty to this schedule

// Select an available ship

if there are ships available in supplier port

randomly select one to serve

else

search for the ship use of all unfulfilled orders

// Choose one of these ships which can finish its order before the departure date of the current order

if $t'_{im'} > t_{im}$, where $t'_{im'}$ is the finish time of one of the unfulfilled order

continue

// Calculate the ballast speed of this ship

$K_{j'n'im'} \leftarrow \max\left(K_{min}, \min\left(\frac{D_{ij}}{t'_{im'} - t_{j'n'} - T^s}, K_{max}\right)\right)$

// Calculate the ballast speed of the rest unfulfilled orders' ships

$K_{j'n'im'} \leftarrow \max\left(K_{min}, \min\left(\frac{D_{ij}}{t'_{im'} - t_{j'n'} - T^s}, K_{max}\right)\right)$

After adding the path option in the extended stochastic model, in the second step in Fig.5, in addition to settling the service start date and laden speed, the path to travel also needs to be determined. If the ship cannot reach the customer port between the grace period at a given speed, it can try another path, which is equivalent to use the different traveling distances as a time buffer.

Here, in order to minimize both inventory and shipping costs, we rounded up the non-integer arrival time. This action will increase the error, resulting in poor results. In the subsequent computation Section 4, in order to reduce the error caused by rounding, we tried to get the time period as small as possible so that the final influence could be within an acceptable range.

The Logic of The Weather Influence

In the basic stochastic model, we consider the influence of weather conditions. The weather distribution predicts the total length of one event, and the effect on ship speed is calculated based on statistics. Each ship will have access to possible weather forecasts on the day it leaves port, including the start date, influenced region, event length for the ship about to sail, and how much ship speed will be reduced (increased) by the weather condition.

After the model reads the weather forecast, it first checks if the ship which is about to travel will be affected by the rough weather event. No event can happen in all the sea regions. Thus, to define an approximate influenced area, we use the distance the ship will have traveled to determine whether there is an interaction between the ship's voyage period and the weather event. For example, assume that sea area A will experience rough weather on day 5 to day 10, and the area A is 500 nautical miles away from the customer port and 1000 nautical miles away from the supplier port. Then we should calculate the date when the ship arrives and leave this area. If the date set does not include day 5 to day 10, then it means that this event will not influence the ship. Beyond that, when we have the path choice just as in model III, we should first judge whether the current path will be influenced, and then repeat procedures mentioned before. If the ship is confirmed to be influenced, then the actual arrival date of the planned voyage is recalculated with the information of the predicted influenced speed. If the actual arrival time exceeds the grace period limit, then go back to the previous step and re-select the departure date, sailing speed, and available ships until this order can be managed on time.

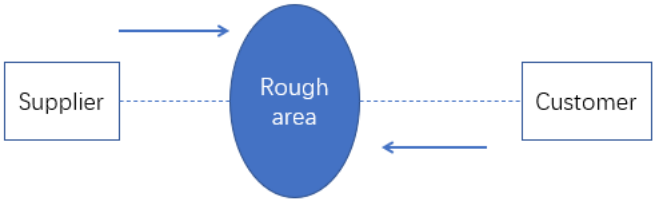

Fig. 6 Rough area define illustration

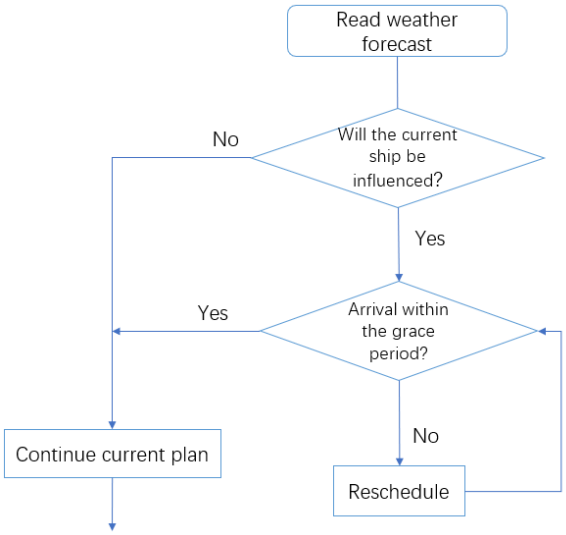

Fig. 7 Logic of Weather Influence

Rescheduling is a trade-off among speed, fuel consumption, and punctuality. For model III, there is one more decision that needs to be made, which is whether to take a long detour to avoid the rough weather event or sailing through the rough area at a reduced speed.

4. Computational Result

4.1 Case Background

4.1.1 Supplier

The supplier, in this case, is Abu Dhabi National Oil Company (ADNOC). ADNOC is one of the world's largest integrated energy companies, with sixteen subsidiary companies in upstream, midstream, and downstream stages of production ("Abu Dhabi National Oil Company," n.d.).

ADNOC LNG, as an important subsidiary of ADNOC, is a "joint venture among ADNOC, which holds the majority 70% share, Japan's Mitsui with a 15% stake, BP, which takes 10% and France's Total with 5%" (Gnana, 2019). It processes and loads liquefied petroleum gas and liquified natural gas at Das Island, and then mainly export to Asia, especially East Asia. Its annual production capacity can reach eight million tons.

Before 2019, Tokyo Electric Power Company (TEPCO) is ADNOC's sole long-term customer, accounting for nearly 90% of its exports and production (ADNOC LNG, 2019). In recent years, TEPCO and Chubu Electric in Japan have merged their LNG operations under the JERA banner. In March 2019, JERA's current long-term contract with ADNOC LNG expired and, in order to diversify its supply base and conclude more flexible terms, “the new deal finalized by the two parties bears no relationship to the legacy contract” (Riviera, 2019). Compared to the old contract, the volume of the new one is reduced by 90%, and only for a limited duration.

At the same time, the gas industry in Abu Dhabi is in flux. Although its reserves are substantial, the Emirate’s current demand is more than production capabilities and growing at a rate of 10% per annum (Riviera, 2019).

In order to seek solutions to fill the vacuum brought by JERA and cope with flux situation, ADNOC LNG has modernized its commercial approach to adopt a new business model towards a multi-customers business that includes not only global utilities but also portfolio players and traders (ADNOC LNG, 2019).

4.1.2 Customers

As mentioned above, before March 2019, Japan is ADNOC LNG's biggest client. After that, ADNOC sought to diversify its customers from Japan to supplying 90% of its output to several

clients and receiving terminals in more than eight countries, including India, China, South Korea, and Bahrain. In this thesis, four major customers, Japan, China, India, South Korea, are mainly introduced.

TEPCO previously signed a 25-year contract with ADNOC with a volume of 4.7 metric tons annually and planned the transportation through the Annual Delivery Program (ADP). The total amount of the new contract is only 0.5 metric tons annually, and many new flexibility requirements are added (ADNOC LNG, 2019). The form is more similar to Shipping Delivery Schedule (SDS). At present, the main Japanese ports for receiving LNG cargo from Das Island are Tokyo, Kawasaki, and Kisarazu.

Australia, Qatar, and the United States are the top three sources of LNG for China, so the company is not a major supplier of LNG to China, and the frequency of receiving cargoes is relatively low. A few years ago, China used gas imported from ADNOC mainly to supply domestic gas in and around Shanghai. Now with the deepening of cooperation, Tianjin port is also open to receiving goods.

At the end of 2019, ADNOC further opened up the South Korean market through energy cooperation with Samsung. However, their new contracts are more about crude oil and LPG, with no significant increase in demand for LNG.

India is the third-largest importer of oil and the fourth-largest importer of gas. Starting in 2018, ADNOC not only secures its position in the Indian energy market but also win a larger number of orders (PIB Delhi, 2018). Now, it has almost become the third-largest source of LNG and LPG of Indian total import.

4.2 Data Source

In the use of the data in this paper, most of the operational data related to ships and contracts are from Tieto Company. The rest of the data, such as weather conditions and port charges, are collected and estimated online.

4.2.1 Operation data

Each year, ADNOC LNG can sell around 95 cargoes, 80 of which are contractual, and the rest are spot. Typically, a relatively long-term contract consists of about 10 small seasonal contracts, each specifying the amount of goods to be delivered each month and destinations.

Among them, 70% of transactions choose DES mode, while only 30% choose FOB mode. That is also a reason why this paper assumes all orders in the planning horizon are DES.

The daily production rate is 50,000 m^3 on average. In the short term, frequent adjustments of production are not cost-efficient, so the model assumes constant production within the time period, but will decide the optimal production rate for the whole season before scheduling. For the medium to long term, due to seasonal fluctuation in demand, the production rate is a variable and could be an important factor for suppliers to meet customer demands on time. Therefore, in the model of this paper, the planning period is three months, and the requirements of each customer port are known in advance. Tieto provided most of the data, while the rest were generated based on ship voyage information between December 2019 and May 2020 from the 'Vessel Finder' website.

	Data	Unit
Production rate	10-70	Thousand cubic meters/day
Upper storage bound	640	Thousand cubic meters
Lower storage bound	10	Thousand cubic meters
Initial Inventory	320	Thousand cubic meters
Inventory cost	0.257K	$/Thousand cubic meters/day
Ship capacity	170	Thousand cubic meters
Ship speed	13-18	knots

Table. 4 Data and Units

So far, the ADNOC's storage capacity is limited to 320,000 cubic meters, and the safety stock is 10,000 cubic meters. The inventory cost per day is about 0.01234 $/MMBtu/day. However, since China, Japan, South Korea, and India are only a part of ADNOC's customers, ADNOC's production volume is actually higher than demands here so that the upper inventory constraints can be relaxed a bit correspondingly.

As for ships, ADNOC Logistic and Service subsidiary is in charge of eight LNG carriers, of which the speed range during the voyage is 13 to 18 knots. Section 3.1 describes the relationship between ship speed and fuel consumption. Therefore, in order to obtain the specific transportation cost, the world average bunker fuel price from January to March was

selected. The capacity of these tankers varies little, averaging 170,000 cubic meters. Usually, each ship loads at Das Island first, which is always fully loaded, then begins the laden leg according to the planned speed and route, and next directly start to return to Das Island after unloading at the customer port, waiting for the next sailing.

4.2.2 Weather and Scenario Generation

Most of the weather impact on a voyage comes from wind and waves, which are seasonal and regional, and affect different types of ships differently. In this case, we define the uncertain event in this paper as 'dates with significant wave height higher than 2.5 meters', and use the statistic results as a forecast for the future. This definition is based on sea state classification given by the World Meteorological Organization (WMO) (see Table. 5).

But in practice, a logistics company will use professional and accurate weather forecasts which are available on the data market because weather is highly uncertain, and historical data alone cannot represent the future. Here, since accurate weather data are not available to us at the moment, historical data is a good substitute for showing our methods.

Wave height	Characteristics	Assumed influence
2.5 to 4 meters	Rough	30%
4 to 6 meters	Very rough	50%
6 to 9 meters	High	80%

Table. 5 Sea state classification

The LNG tanker is among the least vulnerable ships to the rough weather (Heij and Knapp, 2014). At present, we cannot find enough data associated with the performance of LNG tankers, but these two papers Heij and Knapp (2014) and Vettor, Prpic-Orsic, and Soares (2015) describe wind and wave influence on container ships with no apparent differences. So, we directly quoted their conclusions and made some appropriate simplification, such as ignoring the wind direction details, only setting two directions (positive/negative), and choosing the maximum influence range of the shipping speed (see the last column of Table.5). Beyond that, we reduce the eventual speed impact of tankers by 20%, based on the deadweight tonnage differences between containers and LNG tankers. For example, within one event, which is characterized as 'rough', the shipping speed will be decreased by 30% according to Table. 5 if the wind and wave direction is opposite to the shipping direction.

Along the current route, from Das island to the farthest customer Tokyo, there are four regions that ships have to cross: Arabian sea, South of Bay of Bengal, South China Sea, East China Sea. Among them, Bay of Bengal is the site of a majority of the intense tropical cyclones in history. It has significant seasonality high waves and strong wind. Thus, we focus on how rough weather in the South of Bay of Bengal influences the shipping along the route.

So far, the significant wave height data we can obtain through the buoy statistics published by Pacific Islands Ocean Observing System (PacIOOS) is limited between January 2011 and June 2020. We have also collected and analyzed wave data in the Arabian Sea, the South China Sea and the East China Sea, but given the coordinates of the buoys, the observation data in the Bay of Bengal are the closest to the shipping route.

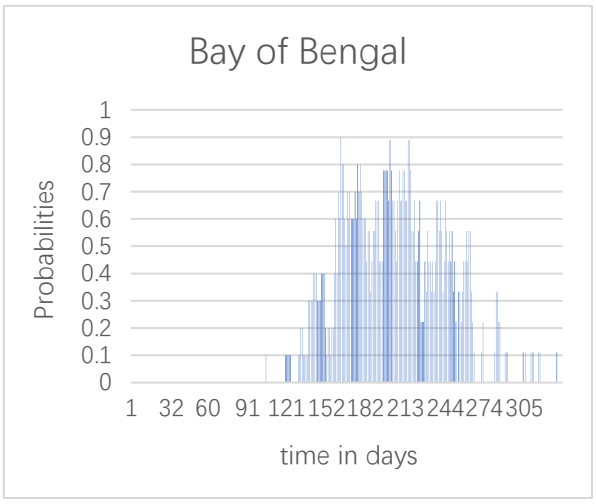

Fig. 8 Rough days Statistic in Bay of Bengal

Fig. 8 shows probabilities and seasonality when an event is likely to occur in this area. It is clear that in the Bay of Bengal, spring and winter are almost not affected by high waves. According to our statistics, the number of days with wave heights above 2.5 meters only accounts for 8% of the season. Moreover, the rough days are relatively scattered, and the duration of one event is mostly around one day, which has little impact on shipping. Summer is totally different. More than half of the season is in the rough state. Rough dates are concentrated and continuous, and the proportion of 'very rough' is significantly higher than in other seasons, leading to hard shipping. Autumn is a transitional season between the two extremes, and around 15% of the days are classified as rough in this paper.

After understanding these characteristics, we fitted the data in Fig. 8 to find a suitable probability distribution with the minimum error and finally obtained the power normal distribution.

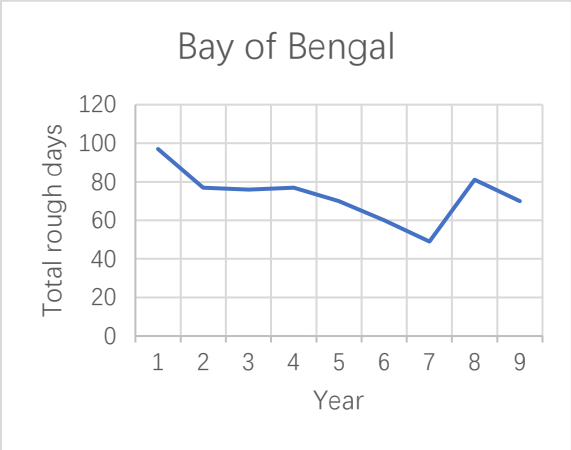

Fig. 9 Trend line of total rough days in Bay of Bengal

Fig. 9 is the trend of total rough days all year round in the past nine years. In this paper, due to the definition of events and the lack of data, we cannot get an effective distribution related to the total number of days of events. Thus, to simplify the prediction, we calculate the mean and the standard deviation and treat it as a normal distribution to predict the total number of rough days of the next year. We think that such a forecast is sufficient for the subsequent analysis, but we also understand that such an assumption is not accurate enough.

As for the distribution of speed influence, we use the statistic result directly. According to the data we have, in the past nine years, the number of days that were classified as 'rough' state accounts for around 80% of the total rough days, the state' very rough' accounts for 19%, and the state 'high' accounts for 1%. Thus, we generate the speed influence according to this discrete distribution.

With all the information above, scenarios are sampled according to the Monte Carlo sampling procedure. In general, the more scenarios used in the model, the accuracy of estimation will increase correspondingly. However, the large number of scenarios may also result in longer computation times. Therefore, we will test with increasing cardinality w from 0 to 300 to find a trade-off between the number of scenarios and computation times in Section 4.3.

In the following, we choose to mainly focus our analysis on the autumn months, as autumn is a neutralization of extremes, which is better to compare different models. Still, we will also mention the characteristics of other seasons but not go into detail.

4.2.3 Ports

There are no statistics on the exact time of unloading or loading at each port, but according to Tieto, such service times usually keep around one day, so it is assumed that all service times are one day. Furthermore, for every port, there is only one berthing at any point in time. If more than one ship arrives at the same port at the same time, or if one ship arrives at a port

where another ship has started unloading (or loading) already, then ships have to queue up to wait.

The LNG demurrage fee is usually very high, around $60,000 per day or more. This penalty will motivate the ship to arrive within the target time window. In real life, the penalty for early arrival is relatively lower than those for the delay. In addition, as for the scheduling of ships, early arrival can also leave more flexibility in the follow-up plans. Thus, in this paper, we assume that the early arrival charge is only 50% of the demurrage fee.

The different sea distances are measured in nautical miles and are collected from the 'Sea Route' website (see Table.6). Each voyage could be traveled via two waypoints in the extended stochastic model. The normal path is the shortest one, which runs directly from the south of Sri Lanka to north of Sabang and then via the Malacca strait, the Singapore Strait, and Taiwan Strait. The long path represents paths that allow ships to avoid wind and waves and is around 10% longer than the short path.

From	To	Distance (Nautical Mile)	
		Normal	Long
Das Island	Tokyo	6482	7224
	Kawasaki	6463	7217
	Kisarazu	6458	7212
	Shanghai	5778	6606
	Tianjin	6347	7219
	Caofeidian	6321	7187
	Busan	6076	6932
	Dahej	1245	1245
	Dabhol	1304	1304

Table. 6 Distance from Das Island to customer ports

This assumption is based on the distance that a ship has to travel if it wants to avoid a storm area whose fetch is 500 nautical miles. This is the typical length of a storm area in the state 'rough' (Carter, 1982), so we think it is a reasonable number. In this case, Dahej and Dabhol will not be influenced, but we want to keep them to compare the speed difference between short and long voyages. Thus, the normal path length and the long path length for both India ports are the same.

4.3 Result Analysis

4.3.1 Basic Attributes

In this section, before measuring the stochastic effect, we will first analyze the basic attributes of the solution of this case.

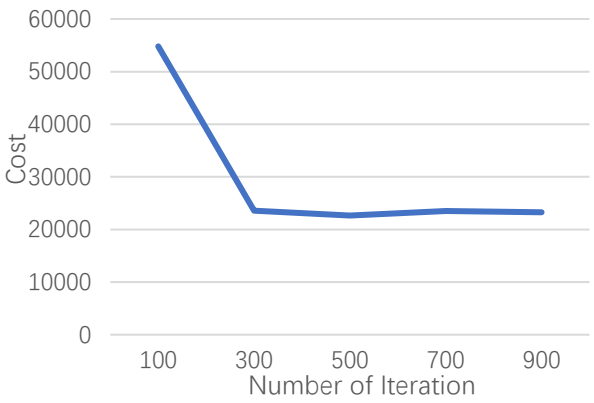

Fig. 10 Number of Iteration – model I

With the increasing number of iterations in the SA algorithm, the value of the objective function (total costs) in the basic model (model I) decrease gradually and almost stable at around 23000 (dollars in thousands) when the number of iterations exceeds 300. Therefore, in the following, we set 500 as the benchmark number of iteration and get the corresponding results.

The computational outcomes of this case are composed of two solution sets. The first set of the solution is an inventory schedule showed in Fig.11. According to this chart, the inventory is well controlled within limits even there is no new arrival order after day 80.

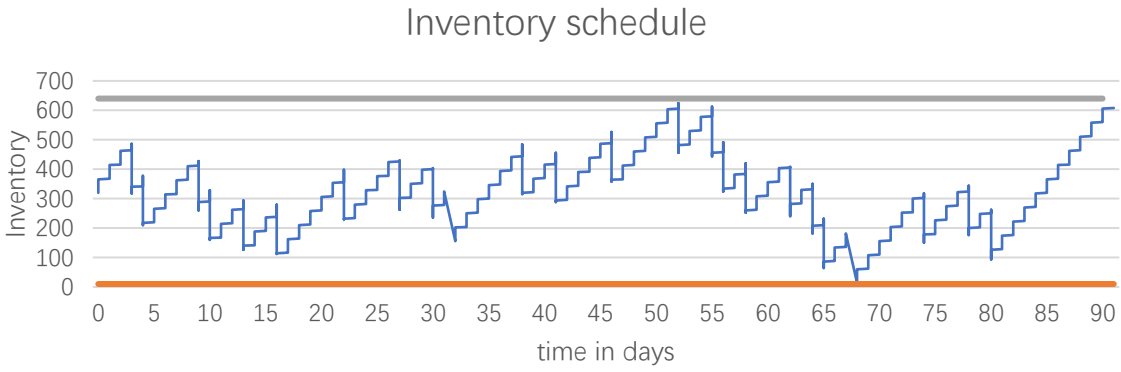

Fig. 11 Inventory Schedule – model I

The second set is the routing decisions, which include vessel assignments to each route, vessel speed, and expected arrival and departure dates.

Overall, laden voyages always have higher speed and higher volatility than the ballast voyage. We think the reason may be that the laden voyage is more time-critical. It should not only ensure that the ship leaves on the date within the inventory limit, but also tries to avoid

unpunctual penalty. Instead, the ballast voyage only needs to make the ship get back to the supplier port before the next delivery. However, this situation changed when we divide routes into two categories according to the geographical location of customer ports.

	total	short-haul	long-haul
laden speed (ave.)	16.38	13.88	17.62
std	2.26	1.42	1.29
ballast speed (ave.)	15.60	15.89	15.46
std	1.48	1.18	1.66

Table. 7 Comparison between laden and ballast speed

Routes to Indian ports are around 1300 nautical miles, while routes to East Asia are all longer than 5500 nautical miles. Therefore, we call the former as short-haul and the later as long-haul.

In terms of saving fuel costs, both long-haul and short-haul transportation should keep a relatively low speed. Nevertheless, from the last two columns of Table. 7, we see that the laden speed of the long-haul is much higher than that of a short-haul. One reason might be that sailing a long distance at a higher speed can save time for the subsequent planning and ensures on-time arrival as much as possible. For example, the total distance from Das Island to Shanghai is 5778 nautical miles. When the speed is adjusted from 13 knots to 18 knots, the one-way trip can save approximately 5 days. For short-haul, the one-way trip in total is around 3 days. Therefore, even if the ship adjusts its speed from 13 to 18 knots, the unpunctual cost will still not be much influenced. On the contrary, it will increase unnecessary fuel consumption instead.

For the ballast voyage, however, the situation is different. There is little difference in ballast speeds between long- and short-haul shipping, and the short-haul speed is even slightly higher. This can be explained by the relationship between the ship's load and fuel consumption. The ballast voyage can be deemed as zero loads, which uses only 80% as much fuel as the laden voyage. Therefore, even if we assigned a higher speed to short-haul shipping, the bunker fuel costs will not increase a lot.

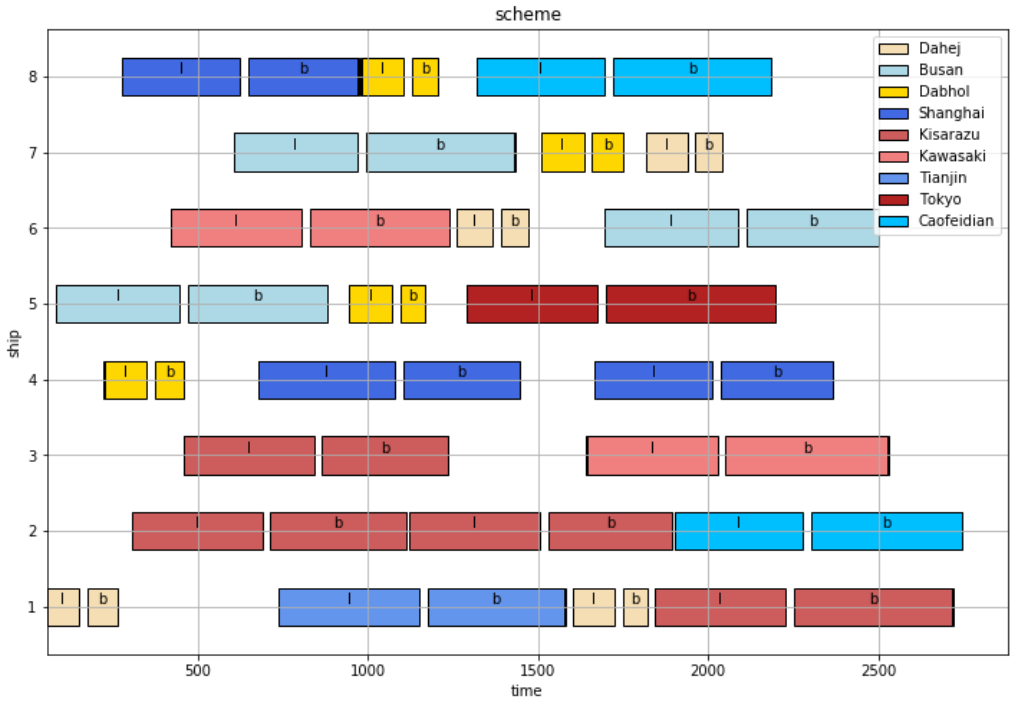

Fig. 12 Gantt Chart of model I

(time is measured in hours, 'l' for laden, 'b' for ballast)

The detailed shipping schedule is presented in a Gantt Chart in Fig. 12. Under this situation, the distribution of weather is unclear and is not taken into consideration, so the quality of this solution is also unclear.

To some extent, it could be flexible. For example, Ship 1 will depart from the supplier port on the first day, and arrive at Dahej for unloading after four days, after which it returns to the supplier port. Assuming that from day 5, Ship 1 would encounter a rough weather event lasting 3 days, during which the ship's speed would be reduced by 5.3 knots, the actual arrival time would be about 20 hours later than initially planned. Originally, Ship 1 will back to the supplier port on day 12, and now it will return one day later. However, it does not influence its next trip, since the next tour starts around 20 days later.

However, model I cannot always absorb the delays caused by rough weather. For example, assuming Ship 5 comes across a severe rough weather event lasting 4 days on its ballast voyage, reducing its speed by 9 knots, Ship 5 is forced to sail more days than expected and cannot return to the supplier's port on time. According to the original plan, after Ship 5 returns to Busan's supplier, its next sailing task has to be started within two days, which is impossible under the current rough situation. The delay will then be passed on to the next voyage and affect the inventory schedule. It is highly likely that the inventory exceeds the ceiling.

What is worse, a rough event may affect not only the route from Das Island to Busan but also other routes to East Asia. Then, it is unwise to rely only on the buffer time in model I to deal with the unknown weather events. In other words, the quality of the deterministic solution highly depends on the weather distribution in reality. If the solution happens to avoid rough events, the quality of the planning is high; if the opposite is the case, the solution of the deterministic model then becomes useless, even it has lower costs. That calls for stochastic models.

4.3.2 Uncertainty Effect Analysis

In this part, the first problem we need to solve is to figure out the best number of scenarios for both stochastic models. In theory, the total costs should increase with the increase in the number of scenarios, and the rate of growth should gradually slow down. However, after a certain threshold, using more scenarios will likely result in a marginal (or no) gain in the total costs, but a significant increase in computation time. Therefore, we conducted a sensitivity analysis to investigate this trade-off between the number of scenarios and the corresponding objective value.

As shown in Fig. 13, the convergency speed decreased significantly at the beginning as the scenario size increases from 0 to 150. After that, the line becomes stable and convergent at around 55000 (dollars in thousands). Occasionally, the total cost seems to decrease as the number of scenarios increases (see Fig.13, from 150 scenarios to 250 scenarios). This inaccuracy is probably because the solution method used in this thesis is a heuristic algorithm. However, as far as the general trend is concerned, the curve of Fig.13 is consistent with our expectations. Under this situation, about 250 scenarios appear to be a reasonable number to obtain a meaningful solution. Because if we choose a higher scenario number, we will surely get a more accurate result, but the computation time will be too high. Two hundred and fifty scenarios seem to be a good trade-off.

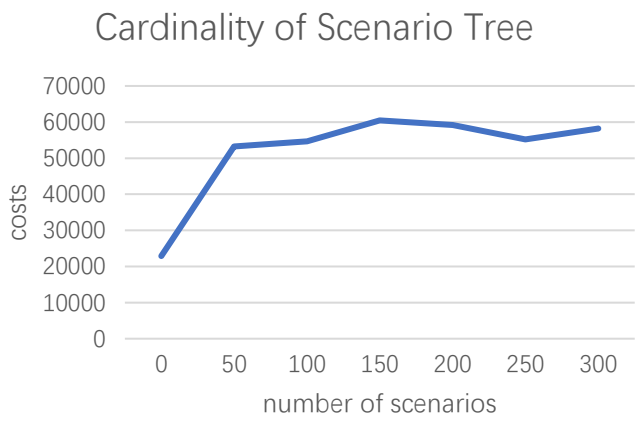

Fig. 13 Cardinality of Scenario Tree – model II

Next, we prepare to conduct the same test to model III. However, in the process of computation, we found that the objective value of model III tends to be stable after the number of scenarios is greater than 45, as shown in Fig. 14. Furthermore, its convergency value is not much different from that of model I. This may be because the loss of bypassing the rough area is much less than the loss by going through it. At the same time, when the direction of wind and wave is the same as the direction of the ship, the ship can even make use of them to increase its actual speed.

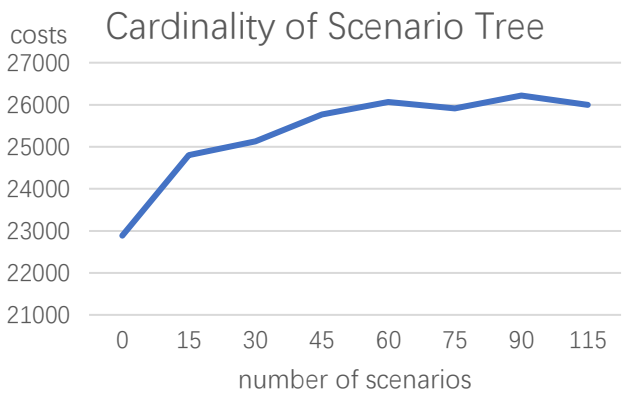

Fig. 14 Cardinality of Scenario Tree – model III

In Section 4.2.3, we give the normal distance and the long distance between ports, respectively, in which the longer distance is about 10% more than the normal one. However, to analyze the trade-off between the long path choice or sailing in the rough area, we changed the length of the longer path several times and tried when the long path is 15%, 20%, 25%, 30% and 35% longer than the normal path. The results showed that the total costs of model II and model III gradually tended to be consistent when a ship needs to travel 25% more than its normal path. In other words, the long detour option becomes meaningless in this case when the cost of bypassing the rough area is traveling 25% more distance.

There is another situation when models II and III tend to be consistent. In spring and winter, during which weather events were at extremely low frequency, the solutions are almost the same between two models. To be more precise, in that case, models I to III would be mostly the same, except that models II and III might have to pay a little more in bunker fuel in the second stage.

Summer is another extreme state. Since the wind and waves cannot be avoided, the scheme can only be optimized by adjusting the speed and changing the schedule. The cost of model II is higher than $1.2 * 10^7$(dollars in thousands). A significant component of this number is the penalty from inventory management failure. Unable to arrive at the customer port on time and unable to return to the supplier port before the next scheduled sailing, the ship will surely be unable to load LNG in time. In model III, when the length of the long path is less than 1.25 times of normal path, the total cost still has little difference with model I.

To sum up, the two stochastic models have similar effects when the rough weather effect is not significant and does not last long. As rough weather increases in its influence on ships and its frequencies, the model III's advantages become more obvious. Adjusting the departure time and sailing speed can only absorb the negative effects to a certain extent when the intervals between events are longer, but choosing a path can directly avoid all the disadvantages. However, model III also has its downside: the path choice can become invalid when the distance to bypass the rough area is too long.

Next, we analyze the specific results of the two models. First of all, in terms of speed allocation, the average laden speed of model II and III is 16.46 and 16.21 knots, respectively. Moreover, the standard deviation in model II is also slightly higher. This is because in model II, under the circumstance of sailing through the wind and waves, if ships still want to ensure the punctuality of arrival, speed up is a good option.

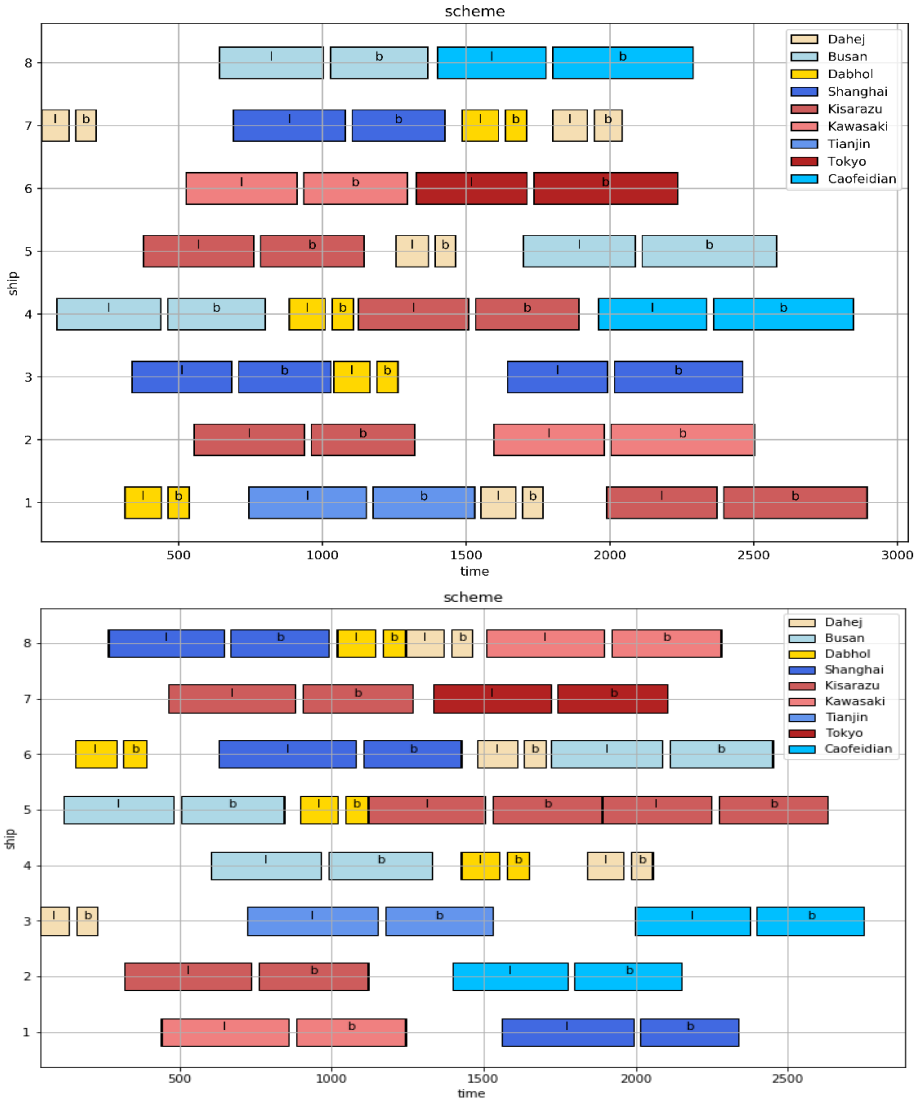

Fig. 15 Comparison of model II and III

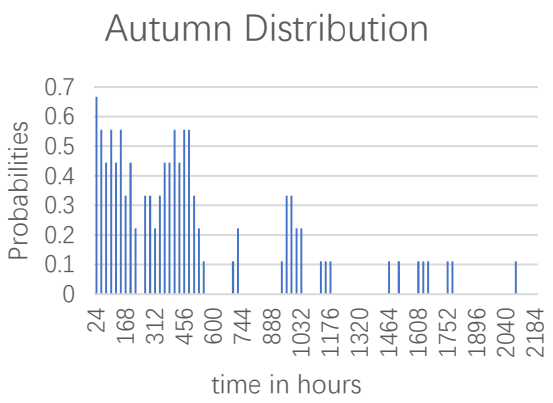

Fig. 16 Rough days distribution in Autumn

Second, in terms of ship allocation, the solution of model II clearly shows signs of avoiding wind and waves, while model III does not. Compare the two figures in Fig. 15, we can see that in the first 500 hours of the season, when the probability of rough days are high (according to Fig.16), the schedule of model II has tried its best to choose a relatively late start time and reduce the sailing time of each ship during this period. Model III can avoid the wind and waves altogether, so there is no need to do the same as model II.

(dollars in thousands)	Model I	Model II	Model III
Total Costs	22885.1	55174.9	25908.3

Table. 8 The total costs of the three models

Third, the objective values of model II and model III are nearly twice different (see results in Table.8). The first probable reason is that after considering different weather scenarios, under current constraints, model II still cannot find a way to avoid inventory management failure, leading to the inventory larger than the upper limit or lower than the lower bound. Another reason is the difference between the number of ship changes. It will cost a lot to change the ship abruptly just before departure. Based on the computational results, considering all scenarios in model II, the weighted average number of ship changes in model II in the second-stage is 7. In model III, the number is less than one, which means that it is almost enough to deal with the rough weather by path changing rather than reassign ships.

Finally, from Table.8, we can also see that both the total cost of model II and model III are higher than model I. That is because the model I, which can be viewed as a zero scenario model, does not take any external factors into account, and it only needs to find a plan with the most economical cost. Thus, the result of this kind of deterministic models will always have lower costs than a stochastic one. However, given the heuristic and the number of iterations, when adapting model I's solution into stochastic settings, the solution will still be feasible but not necessarily the best one anymore. We will use Table. 9 to do the explanation.

(dollars in thousands)	Model I (in model II setting)	Model II	Model I (in model III setting)	Model III
Total Costs	58318.7	55174.9	26067.9	25908.3
Original Schedule Costs	22885.1	28250.3	22885.1	25750.9
Stochastic part	35433.6	26924.7	3182.8	157.4

Table. 9 The deterministic solution in stochastic settings

The total costs of stochastic models, to some extent, can be divided into two parts: the first part is the costs for the original schedule, and the second part is the expected value of extra costs of changing the original schedule into a new one when different scenarios realized. If we only focus on the original schedule costs, it is clear that model I's solutions have the lowest costs. However, when comparing the value of the stochastic part, the disadvantages of model I are obvious. That means, on average, model I's solution will cost far more than the other two models' solutions to adapt to the possible rough weather. In other words, the best solution found by model I is not the best anymore and becomes fragile when we consider weather influence. If we use that schedule in practice, it will be easily affected by continuous wind and wave and ends up higher total costs than solutions given by the stochastic ones.

5. Concluding Remark

5.1 Conclusion

In this paper, we propose three models to solve LNG-SIRP, generating ‘optimal’ scheduling decisions under uncertain weather, and controlling the inventory simultaneously. We put the word 'optimal' in quotes because IRP is itself an optimization problem, and we establish and solve these models according to the logic of optimization. However, since the method we use (the SA algorithm) is heuristic, the final solution obtained is feasible, but not necessarily optimal.

Among the three models, model I is a deterministic model that only minimizes costs for the current state and is used as a comparison for the two stochastic models. Thus, before studying the uncertain factor, we first analyze the solution provided by this model and found that although it can generate a good solution with low costs, it may not be flexible enough to manage the schedule when rough events occur.

Models II and III are generated to deal with uncertain weather. When a rough weather event is realized in the second stage, model II will consider to change the shipping speed or reschedule to reduce the uncertain influence. In contrast, model III has another choice: taking a long detour and bypassing the rough area.

In total, model III has better performance than model II, especially when the rough dates are continuous, and the influence level is high and unavoidable by simply adjusting speed or changing the shipping schedule. However, model III also faces problems when the detour's distance is so long that it results in a higher loss than sailing through the high wave and strong wind area. In this paper, the breakpoint is 1.25 times of the normal path. When the long path is more than 1.25 times longer than the short path, the results of the two models tend to be consistent.

Compared to the deterministic model I, both models II and III result in higher total costs. It doesn't mean that the stochastic models are worse. On the contrary, they consider hundreds of possible scenarios and minimize not only the original schedule costs but also the impact brought by uncertain weather conditions. The quality of these solutions should be better than the deterministic one. To make a more apparent contrast, we put model I’s solution into stochastic settings. Although its initial schedule cost is still the lowest, the rough weather will

bring higher impacts and cause higher losses on average to it than to solutions of stochastic models.

The two-stage model proposed in this paper can be extended to multi-stage in future studies, and the definition of the uncertain event could be more in detail, including more than one event area, so that it can be adapted to a long-term plan.

5.2 Limits

5.2.1 Limits in model

First of all, all our analyses of weather conditions are based on historical data, and models will become unreliable if historical data are inaccurate. That calls for more professional and accurate weather forecasts. Secondly, the scheduling logic applied in this paper is myopic to a certain extent. This is because when assigning ships for each order, this algorithm only considers the solution if the current order cannot arrive at the customer's port on time, but does not consider whether the overall optimization can be achieved by reallocating the current order if the next request is not punctual.

Third, in the computational results in Section 4, model II always has high inventory management failure costs. This might partially be because we set the production rate as constant. Actually, the production rate, in the long run, is changeable. Finally, the supplier's subjective risk preferences are not taken into account. For uncertain events, risk aversion and risk appetite make entirely different decisions.

5.2.2 Limits in solution methods

The limits of the SA algorithm are mainly in the following two aspects: on the one hand, it needs a large amount of computation, which would be time costly. On the other hand, it is uncertain whether the final result is the optimal one. If there is an auxiliary algorithm to prove the SA algorithm's correctness, then the solution will be more reliable.

References

Abu Dhabi National Oil Company, n.d., Wikipedia. Available at: https://en.wikipedia.org/wiki/Abu_Dhabi_National_Oil_Company

ADNOC LNG. (2019) *About ADNOC LNG*. Available at: https://www.adnoc.ae/en/adnoc-lng/who-we-are/about-adnoc-lng

Anderson, H., Christiansen, M., & Fagerholt, K. (2010). Transportation planning and inventory management in the LNG supply chain. In E. Bjørndal, P. M. Pardolos, & M. Ronnqvist (Eds.), *Energy, natural resources and environmental economics* (pp. 427–439). Berlin: Springer.

Argus. (2020) *LNG Shipping Market: Shipping Cost Slightly Declining*. Available at: https://www.hellenicshippingnews.com/lng-shipping-market-shipping-cost-slightly-declining/

Bell, W.J., Dalberto, L.M., Fisher, M.L., Greenfield, A.J., Jaikumar, R., Kedia, P., Mack, R.G., Prutzman, P.J. (1983). Improving the distribution of industrial gases with an online computerized routing and scheduling optimizer. *Interfaces* 13(6), 4-23.

BP Statistics Department (2018). *BP Energy Outlook.* Technical report. British Petroleum.

Cho, J., Lim, G. J., Kim, S. J., & Biobaku, T. (2018). Liquefied natural gas inventory routing problem under uncertain weather conditions. *International Journal of Production Economics*, 204, 18–29.

Coelho, L. C., Cordeau, J.-F., & Laporte, G. (2014). Thirty years of inventory routing. *Transportation Science*, 48(1), 1–19.

D.J.T. Carter. (1982) Prediction of wave height and period for a constant wind velocity using the JONSWAP results. *Ocean Engineering*, Volume 9, Issue 1, 17-33.

Federgruen, A., & Zipkin, P. (1984). A combined vehicle-routing and inventory allocation problem. *Operations Research*, 32(5), pp. 1019–1037.

Gnana J. (2019) *Adnoc signs LNG agreements with oil majors BP and Total*. Available at: https://www.thenational.ae/business/energy/adnoc-signs-lng-agreements-with-oil-majors-bp-and-total-1.936433

Goel, V., Furman, K. C., Song, J.-H., & El-Bakry, A. S. (2012). Large neighborhood search for LNG inventory routing. *Journal of Heuristics*, 18(6), pp. 821–848.

Grønhaug, R. (2008). Optimization Models and Methods for Industrial Supply Chains. Doctoral Thesis, NTNU, Norway.

Grønhaug, R., & Christiansen, M. (2009). Supply chain optimization for the liquefied natural gas business. In M. G. Speranza, & J. A. E. E. van Nunen (Eds.), *Innovation in distribution logistics*: vol. 619 (pp. 195–218). Berlin, Germany: Springer.

Halvorsen-Weare, E. E., & Fagerholt, K. (2013). Routing and scheduling in a liquefied natural gas shipping problem with inventory and berth constraints. *Annals of Operations Research*, 203(1), pp. 167–186.

Heij C and Knapp S (2014), Effects of wind strength and wave height on ship incident risk: regional trends and seasonality, *Econometric Institute Report* 2014-15, Erasmus University

Mutlu, F., Msakni, M. K., Yildiz, H., Soenmez, E., & Pokharel, S. (2016). A comprehensive annual delivery program for upstream liquefied natural gas supply chain. *European Journal of Operational Research*, 250(1), pp. 120–130

PIB Delhi. (2018) *India – UAE bilateral investments*. Available at: https://pib.gov.in/PressReleaseIframePage.aspx?PRID=1520423

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992). *Numerical Recipes in C*. Cambridge University Press, Cambridge.

Rakke J, Stålhane M, Moe C, Andersson H, Christiansen M, Fagerholt K, Norstad I (2011) A rolling horizon heuristic for creating a liquefied natural gas annual delivery program. *Transp Res C* 19(5), pp. 896–911

Riviera. (2019) *Gulf prepares for major increase in LNG exports*. Available at: https://www.rivieramm.com/opinion/opinion/gulf-prepares-for-major-increase-in-lng-exports-22163

Roldan, R. F., Basagoiti, R., & Coelho, L. C. (2017). A survey on the inventory-routing problem with stochastic lead times and demands. *Journal of Applied Logic*, 24(A, SI), 15–24.

Shao, Y., Furman, K. C., Goel, V., & Hoda, S. (2015). A hybrid heuristic strategy for liquefied natural gas inventory routing. *Transportation Research Part C-Emerging Technologies*, 53, pp. 151–171.

S. Nurminarsih, A. Rusdiansyah, N. Siswanto, and A. Z. Gani. (2015). Dynamic-Inventory Ship Routing Problem (DIsrp) Model Considering Port Dwelling Time Information, *Industrial Engineering and Service Science*, Vol. 4, No. 4, pp. 344-351.

Stålhane, M., Rakke, J. G., Moe, C. R., Andersson, H., Christiansen, M., Fagerholt, K., et al. (2012). A construction and improvement heuristic for a liquefied natural gas inventory routing problem. *Computers & Industrial Engineering*, 62(1), pp. 245–255.

Thomas, S., Dawe, R.A., (2003) Review of ways to transport natural gas energy from countries which do not need the gas for domestic use. *Energy*, 28(14), pp. 1461-1477.

Vettor R., Prpić-Oršić J., and Guedes Soares C.. (2015) The effect of wind loads on the attainable ship speed on seaways," in *Towards Green Marine Technology and Transport*, C. Soares, Guedes, R. Dejhalla, and D. Pavletić, Eds. Taylor & Francis Group, London, pp. 867–873.

Wang X., Norstad I., Fagerholt K., Christiansen M. (2019) Green Tramp Shipping Routing and Scheduling: Effects of Market-Based Measures on CO2 Reduction. In: Psaraftis H. (eds) *Sustainable Shipping*. Springer, Cham. https://doi.org/10.1007/978-3-030-04330-8_8

World Energy Council (2016). World Energy Resource | 2016. *Technical Report*.

Zhang, H., Liang, Y., Liao, Q., Yan, X., Shen, Y., & Zhao, Y. (2017) A three-stage stochastic programming method for LNG supply system infrastructure development and inventory routing in demanding countries. *Energy*, 133, pp. 424–442.

Appendix 1 Code for SA Algorithm

```
import pandas as pd
import numpy as np
import copy
import matplotlib.pyplot as plt

import abc
import math
import random
import sys
import time

import warnings

warnings.filterwarnings("ignore")


def time_string(seconds):
    """
    Returns the time in seconds as a string  hh:MM:SS
    Recording time of annealing process
    """
    s = int(round(seconds))
    h, s = divmod(s, 3600)  # Get the hours and the remaining time
    m, s = divmod(s, 60)  # The rest of the time is divided into minutes and seconds
    return '%4i:%02i:%02i' % (h, m, s)


class Annealer(object):
    """
    Simulated Annealing
    """
    # defaults
    # Temperature, max and min
    Tmax = 25000.0
    Tmin = 2.5
    # update the model
    updates = 200

    # save the viarable
    best_state = None
    best_energy = None
    start = None

    def __init__(self, initial_state=None):
        self.state = copy.deepcopy(initial_state)

    @abc.abstractmethod
    def move(self):
        """
        state change
        """
        pass
```

```
    @abc.abstractmethod
    def energy(self):
        """
        energy calculation
        """
        pass

    def update(self, *args, **kwargs):
        """
        if don't rewrite update
        directly use default_update
        """
        self.default_update(*args, **kwargs)

    def default_update(self, step, T, E, acceptance, improvement):
        """
        process output
        print current temperature, energy, acceptance rate, improvement rate, used time and
remaining time
        Acceptance rate: Represents the percentage of moves the Metropolis algorithm has accepted
since the last update.
            It includes motion that reduces energy, motion that holds energy constant, and motion
that increases energy through thermal excitation.
        Improvement rate: Represents the percentage movement of strictly reduced energy since the
last update.
            At high temperatures, it includes both motion that improves the overall state, and
motion that simply eliminates previously increased energy through thermally induced polarization.
            At low temperatures, it tends to zero, because the motion that reduces energy is
exhausted, and the motion that increases energy is no longer thermally accessible.
        """
        elapsed = time.time() - self.start
        if step == 0:
            print(' Temperature       Energy    Accept   Improve     Elapsed
Remaining',
                  file=sys.stderr)
            print('\r%12.5f  %12.2f                    %s           ' %
                  (T, E, time_string(elapsed)), file=sys.stderr, end="\r")
            sys.stderr.flush()
        else:
            remain = (self.steps - step) * (elapsed / step)
            print('\r%12.5f  %12.2f  %7.2f%%  %7.2f%%  %s  %s\r' %
                  (T, E, 100.0 * acceptance, 100.0 * improvement,
                   time_string(elapsed), time_string(remain)), file=sys.stderr,
end="\r")
            sys.stderr.flush()

    def anneal(self):
        """
        The energy of the system is minimized by simulated annealing.
        Return: better state and energy
        """
        step = 0
        self.start = time.time()
```

```
        # Cooling factor from Tmax to Tmin
        Tfactor = -math.log(self.Tmax / self.Tmin)
        # Initial temperature and energy
        T = self.Tmax
        E = self.energy()
        # pre-state and energy
        prevState = copy.deepcopy(self.state)
        prevEnergy = E
        # best state and energy
        self.best_state = copy.deepcopy(self.state)
        self.best_energy = E
        # acceptance rate、improvement rate
        trials, accepts, improves = 0, 0, 0
        if self.updates > 0:
            updateWavelength = self.steps / self.updates
            self.update(step, T, E, None, None)
        # try state transfer
        result_per_step = []
        sec = time.time()
        # iteration
        while step < self.steps:
            step += 1
            # current temperature
            T = self.Tmax * math.exp(Tfactor * step / self.steps)
            # Energy transfer, the change in energy per iteration
            dE = self.move()
            if dE is None:
                E = self.energy()
                dE = E - prevEnergy
            else:
                E += dE
            trials += 1
            ### metropolis
            # if Δt'<0, accept S' as new solution
            # else, accept S' as new solution with the probability exp(-Δt'/T)
            if dE > 0.0 and math.exp(-dE / T) < random.random():
                # Follow the previous state
                self.state = copy.deepcopy(prevState)
                E = prevEnergy
            else:
                # accept new state
                accepts += 1
                if dE < 0.0:
                    improves += 1
                # use current state as previous state for next round
                prevState = copy.deepcopy(self.state)
                prevEnergy = E
                #             print(E, self.best_energy)
                if E < self.best_energy:
                    # If the energy drops, use it as the best energy and state
                    self.best_state = copy.deepcopy(self.state)
                    self.best_energy = E
            if self.updates > 1:
                # update console output
                if (step // updateWavelength) > ((step - 1) // updateWavelength):
```

```
                self.update(
                    step, T, E, accepts / trials, improves / trials)
                trials, accepts, improves = 0, 0, 0
            # The best state for each iteration
            sec1 = time.time()
            result_per_step.append([self.best_energy, self.best_state, sec1 - sec])
            sec = sec1
        self.state = copy.deepcopy(self.best_state)
        # return the best state and energy
        return self.best_state, self.best_energy, result_per_step


class SearchScheme(Annealer):
    def __init__(self):
        self.load_data()
        self.move()
        super(SearchScheme, self).__init__(self.state)

    def load_data(self):
        '''
        load data
        '''
        filename = 'data_hour.xlsx'
        # demand schedule
        self.Requests = pd.read_excel(filename, sheet_name='demand plan', index_col=0)
        self.dayN = len(self.Requests) * 24
        self.Requests = self.Requests.replace(0,
np.nan).dropna(how='all').stack().reset_index()
        ll = []
        for i, line in self.Requests.iterrows():
            if line[0] == 1:
                ll.append(line)
            else:
                for _ in range(int(line[0])):
                    line[0] = 1
                    ll.append(line)
        self.Requests = pd.concat(ll, axis=1).T
        self.Requests.index = range(len(self.Requests))
        self.Requests['level_0'] *= 24
        self.day_request = self.Requests.iloc[:, 0].tolist()
        # distance from port to supplier
        self.distance_port = pd.read_excel(filename, sheet_name='supplier2port',
index_col=0)
        self.portsL = self.distance_port.index.tolist()
        # distance from initial point to supplier
        self.distance_producer = pd.read_excel(filename, sheet_name='init2supplier',
index_col=0)['distance']  # .sort_values()
        # discharge window and grace period
        self.port_days_min, self.port_days_max = pd.read_excel(filename,
sheet_name='window_limit').iloc[0][['earliest', 'latest']]
        self.port_days_min *= 24
        self.port_days_max *= 24
        # unpunctual penalty
        self.port_shift_cost = pd.read_excel(filename, sheet_name='window_penalty',
index_col=0)[['a', 'b']]
```

```
        # ships
        self.ships = self.distance_producer.index.tolist()
        # data of producer
        self.data_producer = dict(pd.read_excel(filename, sheet_name='producer',
index_col=0)['data'])
        self.data_producer['load_time'] *= 24
        self.data_producer['storage_cost'] /= 24
        self.data_producer['prod_rate'] = 2
        self.data_producer['inv_t_unit'] =
pd.concat([pd.Series(self.data_producer['Initial_inv'], index=range(self.dayN + 1)),
                                            pd.Series(self.data_producer['prod_rate'],
                                                    index=range(1, self.dayN +
1)).cumsum()], axis=1).sum(axis=1)
        # ship_data
        self.data_ship = dict(pd.read_excel(filename, sheet_name='ship',
index_col=0)['data'])
        for k in ['ship_speed', 'relation_speed_fuel']:
            self.data_ship[k] = np.array(list(map(float,
self.data_ship[k].split(','))))
        self.data_ship['unload_time'] *= 24
        self.data_ship['load_time'] *= 24
        self.data_ship['days_ini2pro'] = self.distance_producer.apply(
            lambda x: np.ceil(x / (self.data_ship['ship_speed'])).astype(int))
        self.data_ship['days_pro2port'] = self.distance_port.stack().apply(
            lambda x: np.ceil(x / self.data_ship['ship_speed']).astype(int)).unstack()
        D = {}
        for port, paths in self.data_ship['days_pro2port'].iterrows():
            D[port] = paths.apply(lambda x: pd.Series(x, index=['max',
'min']))[['min', 'max']]
        self.data_ship['days_pro2port'] = D
        # event data
        self.data_storm = pd.read_excel(filename, sheet_name='event_influence')
        self.data_probability = pd.read_excel(filename,
sheet_name='probability',index_col=None)
        self.data_storm['start_time'] *= 24
        self.data_storm['end_time'] = self.data_storm['start_time'] + 23
        self.data_storm['influenced_dates'] = self.data_storm.apply(lambda x:
set(range(int(x['start_time']), int(x['end_time'] + 1))), axis=1)

    def move(self):
        # Loop until find the suitable solution
        while True:
            try:
                self.move_()
                break
            except:
                pass

    def move_(self):
        '''
        ship scheme generation
        '''
```

```
        spd1, spd2 = self.data_ship['ship_speed']
        # if orders are in the same day, randomly choose one
        Requests = self.Requests.sample(frac=1).sort_values('level_0')
        ## save variables
        # available ships
        ships_empty = copy.deepcopy(self.ships)
        ships_empty_w = {w: copy.deepcopy(self.ships) for w in range(1,11)}
        # product rate per hour
        producer_fuel = copy.deepcopy(self.data_producer['inv_t_unit'])
        producer_fuel_w = {w: copy.deepcopy(self.data_producer['inv_t_unit']) for w
in range(1,11)}
        # Date of last visit to supplier
        DAY_pro_last = ship_day_pro = 0
        DAY_pro_last_w = {w: 0 for w in range(1,11)}
        ship_day_pro_w = {w: 0 for w in range(1,11)}
        # shipping details
        shipL = []
        shipL_w = {w: [] for w in range(1,11)}
        # Ships to return to supplier
        ship_backto_pro = pd.DataFrame()
        ship_backto_pro_w = {w: pd.DataFrame() for w in range(1,11)}
        # The latest variable for the ship to return to the supplier
        ship_backto_port_day = {}
        ship_backto_port_day_w = {w: {} for w in range(1,11)}
        # Queuing days in supplier port
        producer_queue_Days = []
        producer_queue_Days_w = {w: [] for w in range(1,11)}
        # Queuing days in customer ports
        port_queue_Days = {port: [] for port in self.portsL}
        port_queue_Days_w = {w:{port: [] for port in self.portsL} for w in
range(1,11)}
        state_w = {w: [] for w in range(1,11)}
        # the number of ship reassignment
        ship_changeN = 0
        delta_fc1 = 0
        delta_fc2 = 0
        delta_shift = 0
        EastA = ['Busan', 'Kawasaki', 'Kisarazu', 'Tokyo', 'Tianjin', 'Shanghai',
'Caofeidian']

        i = 0
        for DAY0, port, need in Requests.values:
            i += 1
            #           if i == 1 :
            #               break
            DAY = DAY0
            # judge the date of arrival at the port and the number of queuing days 1
            day_shift_ = 0
            while True:
                if DAY in port_queue_Days[port]:
                    if random.random() < 0.5:
                        DAY += 24
                        day_shift_ += 24
                    else:
                        DAY -= 24
```

```
                    day_shift_ -= 24
                else:
                    break
            # days_pro2port
            day_pro2port = copy.deepcopy(self.data_ship['days_pro2port'][port])

            # if there's no idle ship available
            if len(ships_empty) == 0:
                # The date of the fastest return ship to the supplier
                day_ship_nearest = int(ship_backto_pro.min().min())
            else:
                day_ship_nearest = 0

            # the least number of days for the current ship to return to supplier
            if len(ships_empty) > 0:
                day_ship2pro_nearest =
self.data_ship['days_ini2pro'][ships_empty].apply(lambda x: min(x)).min()
            else:
                day_ship2pro_nearest = 0

            # Find the appropriate path from the supplier to customer, and record the days of the
arrival and queuing days 2
            path_notok = []
            while True:
                queue_day_pro = 0
                queue_day_port = 0

                port_fuel = self.data_ship['ship_cap']
                # the day when inventory reach the ship capacity
                day_producer_need = \
                producer_fuel[(producer_fuel.index > ship_day_pro) &
(producer_fuel >= port_fuel)].index[0]
                # DAY_last to DAY, the day when inventory reach the upper limits
                day_producer_full = DAY_pro_last + math.ceil(
                    (self.data_producer['upper_limit'] - producer_fuel[DAY_pro_last]) /
self.data_producer['prod_rate'])

                # first judge if the ship can arrive on time
                for path, (delta_min, delta_max) in
day_pro2port.sample(frac=1).iterrows():
                    if path in path_notok:
                        continue
                    #                 break
                    day_pro_min = DAY - delta_max - self.data_ship['load_time']
                    day_pro_max = DAY - delta_min - self.data_ship['load_time']
                    days_pro_ok = set(range(day_pro_min, day_pro_max + 1)) & set(
                        range(day_producer_need, day_producer_full + 1))
                    days_pro_ok -= set(producer_queue_Days)
                    days_pro_ok -= set(range(-1, day_ship_nearest))
                    days_pro_ok -= set(range(-1, day_ship2pro_nearest))
                    if len(days_pro_ok) == 0:
                        continue
                    # departure date
                    ship_day_pro = random.choice(list(days_pro_ok))
```

```
                # laden speed
                ship_speed_2 = self.distance_port.loc[port, path] / (DAY -
ship_day_pro - self.data_ship['load_time'])
                ship_speed_2 = self.adj_speed(ship_speed_2)
                day_shift = day_shift_
                break
            # If it cannot arrive on time, find a path that satisfies the grace period
            else:
                for path, (delta_min, delta_max) in
day_pro2port.sample(frac=1).iterrows():
                    #                 break
                    if path in path_notok:
                        continue
                    day_pro_min = DAY - delta_max - self.data_ship['load_time'] +
self.port_days_min - day_shift_
                    day_pro_max = DAY - delta_min - self.data_ship['load_time'] +
self.port_days_max - day_shift_
                    days_pro_ok = set(range(day_pro_min, day_pro_max + 1)) & set(
                        range(day_producer_need, day_producer_full + 1))
                    days_pro_ok -= set(producer_queue_Days)
                    days_pro_ok -= set(range(-1, day_ship_nearest))
                    days_pro_ok -= set(range(-1, day_ship2pro_nearest))
                    if len(days_pro_ok) == 0:
                        continue
                    # departure date
                    ship_day_pro = random.choice(list(days_pro_ok))
                    ship_day_delta2 = DAY - ship_day_pro -
self.data_ship['load_time']
                    ship_day_delta2_real = np.clip([ship_day_delta2], delta_min,
delta_max)[0]
                    # laden speed
                    ship_speed_2 = self.distance_port.loc[port, path] /
ship_day_delta2_real
                    ship_speed_2 = self.adj_speed(ship_speed_2)
                    # unpunctual days
                    day_shift = ship_day_pro + ship_day_delta2_real +
self.data_ship['load_time'] - DAY + day_shift_
                    break
                # If it cannot arrive on time and there is no path to meet the grace period,
the supplier will be penalized for exceeding the inventory limit and postpone sailing
                else:
                    day_pro_outlier_min = DAY + self.port_days_min -
day_pro2port.max().max() - self.data_ship[
                        'load_time']
                    # If a ship is available to return and arrive at the supplier in time
                    if day_pro_outlier_min >= max(day_ship2pro_nearest,
day_ship_nearest):
                        # departure date
                        ship_day_pro = day_pro_outlier_min
                        # unpunctual date
                        day_shift = self.port_days_min + day_shift_
                        path = 'far'
                        # laden speed
                        ship_speed_2 = self.data_ship['ship_speed'][0]
```

```
                        ship_speed_2 = self.adj_speed(ship_speed_2)
                    else:
                        # departure date
                        ship_day_pro = max(day_ship2pro_nearest, day_ship_nearest)
                        # laden speed
                        path = 'near'
                        ship_speed_2 = self.data_ship['ship_speed'][1]
                        ship_speed_2 = self.adj_speed(ship_speed_2)
                        # unpunctual dates
                        day_shift = ship_day_pro + math.ceil(
                            self.distance_port.loc[port, path] / ship_speed_2) - DAY
                # arrival date
                ship_day_port = DAY + day_shift
                fc1 = (0.0019 * (ship_speed_2 ** 2) - 0.045 * ship_speed_2 + 0.3739)
* (ship_day_port - ship_day_pro - 24)
                break
            # When there are ships at the port, then queue up
            while True:
                if ship_day_port + queue_day_port in port_queue_Days[port]:
                    queue_day_port += 1
                else:
                    break
            # When no ships are available, randomly return ships from the list of ships awaiting
return
            if len(ships_empty) == 0:
                for tp in [('near', 'min'), ('near', 'max'),('far', 'min'), ('far',
'max')]:
                    for ship, day in
ship_backto_pro[tp].sample(frac=1).astype(int).iteritems():
                        if day <= ship_day_pro:
                            # break
                            d1, d2, port_, speed_1, DAY_ = ship_backto_port_day[ship]
                            speed_2 = self.adj_speed(self.distance_port.loc[port_,
tp[0]] / (day - 2 * d2 - d1))
                            fc2 = ((0.0019 * (speed_2 ** 2) - 0.045 * speed_2 + 0.3739)
* 0.8) * (day - 2 * d2 - d1)
                            shipL.append(['back', port_, ship, 0, day - d2, day - 2 *
d2, d1, speed_1, speed_2,
                                          0, 0, tp[0], DAY_, fc2])
                            ships_empty.append(ship)
                            ship_backto_pro.drop(ship, inplace=True)
                            del ship_backto_port_day[ship]
                            break
                    else:
                        continue
                    break

                # print(ships_empty)
                # ships chosen
            try:
                while True:
                    ship = random.choice(ships_empty)
                    ship_deltas_pro = self.data_ship['days_ini2pro'][ship]
                    if ship_deltas_pro[1] <= ship_day_pro:
                        break
```

```
            except:
                self.state = [[], 0, 0, 0, 0, 0, 0]
                #             return
                # print(ships_empty, ship)
            ship_delta_1 = random.randint(ship_deltas_pro[1], min(ship_day_pro,
ship_deltas_pro[0]))
            ship_speed_1 = self.distance_producer[ship] / ship_delta_1
            ship_speed_1 = self.adj_speed(ship_speed_1)
            ship_day_spot = ship_day_pro - ship_delta_1

            # record variables
            shipL.append(
                ['load', port, ship, day_shift, ship_day_spot, ship_day_pro,
ship_day_port, ship_speed_1, ship_speed_2,
                 queue_day_pro, queue_day_port, path, DAY, fc1])
            ships_empty.remove(ship)
            DAY_pro_last = ship_day_pro
            producer_fuel.loc[DAY_pro_last:] -= port_fuel
            producer_queue_Days += list(range(ship_day_pro + queue_day_port,
ship_day_pro + queue_day_port + 23))
            port_queue_Days[port] += list(range(ship_day_port + queue_day_port,
ship_day_port + queue_day_port + 23))

            # Record the demand variables of the ship to facilitate the future planning of the
ballast voyage of the ship
            delta_2, delta_1 = self.data_ship['days_ini2pro'][ship]
            delta_pro2spot = random.randint(delta_1, delta_2)
            speed_ = self.adj_speed(self.distance_producer[ship] / delta_pro2spot)
            back_days = (ship_day_port + self.data_ship[
                'unload_time'] + queue_day_port + day_pro2port + delta_pro2spot *
2).stack()
            back_days.name = ship
            ship_backto_pro = ship_backto_pro.append(back_days)
            ship_back_port = ship_day_port + self.data_ship['unload_time'] +
queue_day_port
            ship_backto_port_day[ship] = [ship_back_port, delta_pro2spot, port,
speed_, DAY]


# Take weather into consideration, repeat the code above
            for w in range(1,11):
                DAY_w = DAY0
                day_shift_w_ = 0
                while True:
                    if DAY_w in port_queue_Days_w[w][port]:
                        if random.random() < 0.5:
                            DAY_w += 24
                            day_shift_w_ += 24
                        else:
                            DAY_w -= 24
                            day_shift_w_ -= 24
                    else:
                        break
                # days_pro2port
                day_pro2port = copy.deepcopy(self.data_ship['days_pro2port'][port])
```

```
            if len(ships_empty_w[w]) == 0:

                day_ship_nearest_w = int(ship_backto_pro_w[w].min().min())
            else:
                day_ship_nearest_w = 0

            if len(ships_empty_w[w]) > 0:
                day_ship2pro_nearest_w =
self.data_ship['days_ini2pro'][ships_empty_w[w]].apply(lambda x: min(x)).min()
            else:
                day_ship2pro_nearest_w = 0

            path_notok_w = []
            while True:
                queue_day_pro_w = 0
                queue_day_port_w = 0

                port_fuel = self.data_ship['ship_cap']
                day_producer_need_w = \
                producer_fuel_w[w][(producer_fuel_w[w].index > ship_day_pro_w[w])
& (producer_fuel_w[w] >= port_fuel)].index[0]
                day_producer_full_w = DAY_pro_last_w[w] + math.ceil(
                    (self.data_producer['upper_limit'] -
producer_fuel_w[w][DAY_pro_last_w[w]]) / self.data_producer['prod_rate'])

                for path_w, (delta_min, delta_max) in
day_pro2port.sample(frac=1).iterrows():
                    if path_w in path_notok_w:
                        continue
                    #                 break
                    day_pro_min_w = DAY_w - delta_max - self.data_ship['load_time']
                    day_pro_max_w = DAY_w - delta_min - self.data_ship['load_time']
                    days_pro_ok_w = set(range(day_pro_min_w, day_pro_max_w + 1)) &
set(
                        range(day_producer_need_w, day_producer_full_w + 1))
                    days_pro_ok_w -= set(producer_queue_Days_w[w])
                    days_pro_ok_w -= set(range(-1, day_ship_nearest_w))
                    days_pro_ok_w -= set(range(-1, day_ship2pro_nearest_w))
                    if len(days_pro_ok_w) == 0:
                        continue
                    ship_day_pro_w[w] = random.choice(list(days_pro_ok_w))
                    ship_speed_2_w = self.distance_port.loc[port, path_w] / (DAY_w
- ship_day_pro_w[w] - self.data_ship['load_time'])
                    ship_speed_2_w = self.adj_speed(ship_speed_2_w)
                    day_shift_w = day_shift_w_
                    break

                else:
                    for path_w, (delta_min, delta_max) in
day_pro2port.sample(frac=1).iterrows():
                        #                 break
                        if path_w in path_notok_w:
                            continue
                        day_pro_min_w = DAY_w - delta_max -
```

```
self.data_ship['load_time'] + self.port_days_min - day_shift_w_
                        day_pro_max_w = DAY_w - delta_min -
self.data_ship['load_time'] + self.port_days_max - day_shift_w_
                        days_pro_ok_w = set(range(day_pro_min_w, day_pro_max_w +
1)) & set(
                            range(day_producer_need_w, day_producer_full_w + 1))
                        days_pro_ok_w -= set(producer_queue_Days_w[w])
                        days_pro_ok_w -= set(range(-1, day_ship_nearest_w))
                        days_pro_ok_w -= set(range(-1, day_ship2pro_nearest_w))
                        if len(days_pro_ok_w) == 0:
                            continue

                        ship_day_pro_w[w] = random.choice(list(days_pro_ok_w))
                        ship_day_delta2_w = DAY_w - ship_day_pro_w[w] -
self.data_ship['load_time']
                        ship_day_delta2_real_w = np.clip([ship_day_delta2_w],
delta_min, delta_max)[0]

                        ship_speed_2_w = self.distance_port.loc[port, path_w] /
ship_day_delta2_real_w
                        ship_speed_2_w = self.adj_speed(ship_speed_2_w)

                        day_shift_w = ship_day_pro_w[w] + ship_day_delta2_real_w +
self.data_ship['load_time'] - DAY_w + day_shift_w_
                        break

                    else:
                        day_pro_outlier_min_w = DAY_w + self.port_days_min -
day_pro2port.max().max() - self.data_ship[
                            'load_time']

                        if day_pro_outlier_min_w >= max(day_ship2pro_nearest_w,
day_ship_nearest_w):
                            ship_day_pro_w[w] = day_pro_outlier_min_w

                            day_shift_w = self.port_days_min + day_shift_w_
                            path_w = 'far'
                            ship_speed_2_w = self.data_ship['ship_speed'][0]
                            ship_speed_2_w = self.adj_speed(ship_speed_2_w)
                        else:

                            ship_day_pro_w[w] = max(day_ship2pro_nearest_w,
day_ship_nearest_w)

                            path_w = 'near'
                            ship_speed_2_w = self.data_ship['ship_speed'][1]
                            ship_speed_2_w = self.adj_speed(ship_speed_2_w)

                            day_shift_w = ship_day_pro_w[w] + math.ceil(
                                self.distance_port.loc[port, path] / ship_speed_2_w)
- DAY_w

                ship_day_port_w = DAY_w + day_shift_w
                fc1_w = (0.0019 * (ship_speed_2_w ** 2) - 0.045 * ship_speed_2_w +
```

```
0.3739) * ( \
                            ship_day_port_w - ship_day_pro_w[w] - 24)
                if port in EastA:
                    # judge if there is a weather event
                    storm_port = self.data_storm[(self.data_storm['scenario'] ==
w)& ('near' == path_w)]
                else:
                    break
                if len(storm_port) == 0:
                    break
                path_days = set(range(ship_day_pro_w[w] + (math.ceil(
                    self.distance_port.loc[port,path_w] * 0.25 / ship_speed_2_w)),
ship_day_pro_w[w] + (math.ceil(
                    self.distance_port.loc[port,path_w] * 0.45 / ship_speed_2_w))
+ 1))
                storm_port['influenced_dates_ship'] = storm_port.apply(lambda x:
path_days & x['influenced_dates'], axis=1)
                storm_port['influenced_days_ship'] =
storm_port['influenced_dates_ship'].apply(lambda x: len(x))
                if storm_port['influenced_days_ship'].max() == 0:
                    break

                ship_speed_2_w_ = ship_speed_2_w * (1 -
storm_port['influenced_speed'])
                distance_delta2_pro2port = (storm_port['influenced_days_ship'] *
ship_speed_2_w_).sum()

                ship_day_port_w = ship_day_pro_w[w] + math.ceil(
                    (self.distance_port.loc[port, path_w] -
distance_delta2_pro2port) / ship_speed_2_w + (
                    storm_port['influenced_days_ship']).sum())
                fc1_w = (0.0019 * (ship_speed_2_w ** 2) - 0.045 * ship_speed_2_w +
0.3739) * (
                        ship_day_port_w - ship_day_pro_w[w] - 24)
                if day_shift_w <= self.port_days_min or day_shift_w >=
self.port_days_max:

                    day_shift_w = ship_day_port_w - DAY_w
                    break
                else:
                    day_shift_w = ship_day_port_w - DAY_w
                    if day_shift_w < self.port_days_min or day_shift_w >
self.port_days_max:
                        speed_up = self.distance_port.loc[port, path_w] /
(self.port_days_max + DAY_w - ship_day_pro_w[w]
                                                                -
(storm_port['influenced_speed'] * storm_port[
                                    'influenced_days_ship']).sum())
                        speed_low = self.distance_port.loc[port, path_w] /
(self.port_days_min + DAY_w - ship_day_pro_w[w]
                                                                -
(storm_port['influenced_speed'] * storm_port[
                                    'influenced_days_ship']).sum())
                        ship_speed_2_w = random.uniform(speed_up, speed_low)
```

```
                        ship_speed_2_w = self.adj_speed(ship_speed_2_w)
                        ship_speed_2_w_ = ship_speed_2_w * (1 -
storm_port['influenced_speed'])
                        distance_delta2_pro2port = (storm_port['influenced_days_ship']
* ship_speed_2_w_).sum()

                        ship_day_port_w = ship_day_pro_w[w] + math.ceil(
                            (self.distance_port.loc[port, path_w] -
distance_delta2_pro2port) / ship_speed_2_w + (
                                storm_port['influenced_days_ship']).sum())
                        day_shift_w = ship_day_port_w - DAY_w
                        fc1_w = (0.0019 * (ship_speed_2_w ** 2) - 0.045 *
ship_speed_2_w + 0.3739) * (
                                ship_day_port_w - ship_day_pro_w[w] - 24)
                        break
                    else:
                        break
            # calculate fuel consumption change & day_shift change
            delta_fc1 += self.data_probability['probability'].iloc[w-1] * (fc1_w
- fc1)
            if day_shift <= 0 & day_shift_w <= 0:
                ds = (day_shift - day_shift_w)*30/24
            elif day_shift_w >= 0 & day_shift >= 0:
                ds = (day_shift_w - day_shift)*50/24
            elif day_shift <= 0 & day_shift_w >= 0:
                ds = day_shift_w * 50/24 - abs(day_shift)*30/24
            else:
                ds = abs(day_shift_w) * 30/24 - day_shift * 50/24
            delta_shift += self.data_probability['probability'].iloc[w-1] * ds

            # queueing up when there are ships already in the port
            while True:
                if ship_day_port_w + queue_day_port_w in
port_queue_Days_w[w][port]:
                    queue_day_port_w += 1
                else:
                    break
            # When no ships are available, randomly return ships from the list of ships
awaiting return
            if len(ships_empty_w[w]) == 0:
                for tp_w in [('near', 'min'), ('near', 'max'), ('far', 'min'),
('far', 'max')]:
                    for ship_w, day_w in
ship_backto_pro_w[w][tp_w].sample(frac=1).astype(int).iteritems():
                        if day_w <= ship_day_pro_w[w]:
                            # break
                            d1_w, d2_w, port_w_, speed_1_w, DAY_w_ =
ship_backto_port_day_w[w][ship_w]
                            speed_2_w =
self.adj_speed(self.distance_port.loc[port_w_, tp_w[0]] / (day_w - 2 * d2_w -
d1_w))
                            fc2_w = ((0.0019 * (speed_2_w ** 2) - 0.045 * speed_2_w
+ 0.3739) * 0.8) * (day_w - 2 * d2_w - d1_w)

                            storm_pro = self.data_storm[(self.data_storm['scenario']
```

```
== w) & ('near'== tp_w[0])]
                            if len(storm_pro) == 0:
                                shipL_w[w].append(
                                    ['back', port_w_, ship_w, 0, day_w - d2_w, day_w
- 2 * d2_w, d1_w, speed_1_w,
                                     speed_2_w,
                                     0, 0, tp_w[0], DAY_w_, fc2_w])
                                ships_empty_w[w].append(ship_w)
                                ship_backto_pro_w[w].drop(ship_w, inplace=True)
                                del ship_backto_port_day_w[w][ship_w]
                                break
                            path_back_days_w = set(range(d1_w +
(math.ceil(self.distance_port.loc[port_w_,tp_w[0]] * 0.55 / speed_2_w)),
                                                         d1_w +
(math.ceil(self.distance_port.loc[port_w_, tp_w[0]] * 0.75 / speed_2_w)) + 1))
                            storm_pro['influenced_dates_ship'] = storm_pro.apply(lambda
x: path_back_days_w & x['influenced_dates'],
                                                                 axis=1)
                            storm_pro['influenced_days_ship'] =
storm_pro['influenced_dates_ship'].apply(lambda x: len(x))
                            if storm_pro['influenced_days_ship'].max() == 0:
                                shipL_w[w].append(['back', port_w_, ship_w, 0, day_w
- d2_w, day_w - 2 * d2_w, d1_w, speed_1_w,
                                       speed_2_w, 0, 0, tp_w[0], DAY_w_, fc2_w])
                                ships_empty_w[w].append(ship_w)
                                ship_backto_pro_w[w].drop(ship_w, inplace=True)
                                del ship_backto_port_day_w[w][ship_w]
                                break
                            speed_2_w = self.adj_speed(speed_2_w)
                            speed_2_w_ = ship_speed_2_w * (1 +
storm_pro['influenced_speed'])
                            distance_delta2_back2pro =
(storm_pro['influenced_days_ship'] * speed_2_w_).sum()
                            day_w = d1_w + math.ceil(
                                (self.distance_port.loc[port_w_, tp_w[0]] -
distance_delta2_back2pro) / speed_2_w + (
                                    storm_pro['influenced_days_ship']).sum())
                            fc2_w = ((0.0019 * (speed_2_w ** 2) - 0.045 * speed_2_w
+ 0.3739) * 0.8) * (day_w - 2 * d2_w - d1_w)
                            if day_w <= ship_day_pro_w[w]:
                                shipL_w[w].append(['back', port_w_, ship_w, 0, day_w
- d2_w, day_w - 2 * d2_w, d1_w, speed_1_w, speed_2_w,
                                                   0, 0, tp_w[0], DAY_w_, fc2_w])
                                ships_empty_w[w].append(ship_w)
                                ship_backto_pro_w[w].drop(ship_w, inplace=True)
                                del ship_backto_port_day_w[w][ship_w]
                                break
                        else:
                            continue
                    break

                    # print(ships_empty)
                    # ships chosen
                try:
                    while True:
```

```
                    ship_w = random.choice(ships_empty_w[w])
                    if ship_w != ship:
                        ship_changeN +=
self.data_probability['probability'].iloc[w-1]

                    ship_deltas_pro = self.data_ship['days_ini2pro'][ship_w]
                    if ship_deltas_pro[1] <= ship_day_pro_w[w]:
                        break
                except:
                    self.state = [[], 0, 0, 0, 0, 0]
                    #             return
                    # print(ships_empty, ship)


                ship_delta_1_w = random.randint(ship_deltas_pro[1],
min(ship_day_pro_w[w], ship_deltas_pro[0]))

                ship_speed_1_w = self.distance_producer[ship_w] / ship_delta_1_w
                ship_speed_1_w = self.adj_speed(ship_speed_1_w)


                ship_day_spot_w = ship_day_pro_w[w] - ship_delta_1_w


                shipL_w[w].append(
                    ['load', port, ship_w, day_shift_w, ship_day_spot_w,
ship_day_pro_w[w], ship_day_port_w, ship_speed_1_w, ship_speed_2_w,
                     queue_day_pro_w, queue_day_port_w, path_w, DAY_w, fc1_w])
                ships_empty_w[w].remove(ship_w)
                DAY_pro_last_w[w] = ship_day_pro_w[w]
                producer_fuel_w[w].loc[DAY_pro_last_w[w]:] -= port_fuel
                producer_queue_Days_w[w] += list(range(ship_day_pro_w[w] +
queue_day_port_w, ship_day_pro_w[w] + queue_day_port_w + 23))
                port_queue_Days_w[w][port] += list(range(ship_day_port_w +
queue_day_port_w, ship_day_port_w + queue_day_port_w + 23))

                delta_2_w, delta_1_w = self.data_ship['days_ini2pro'][ship_w]
                delta_pro2spot_w = random.randint(delta_1_w, delta_2_w)
                speed_w_ = self.adj_speed(self.distance_producer[ship_w] /
delta_pro2spot_w)
                back_days_w = (ship_day_port_w + self.data_ship[
                    'unload_time'] + queue_day_port_w + day_pro2port + delta_pro2spot_w
* 2).stack()
                back_days_w.name = ship_w
                ship_backto_pro_w[w] = ship_backto_pro_w[w].append(back_days_w)
                ship_back_port_w = ship_day_port_w + self.data_ship['unload_time'] +
queue_day_port_w
                ship_backto_port_day_w[w][ship_w] = [ship_back_port_w,
delta_pro2spot_w, port, speed_w_, DAY_w]
                state_w[w] = [ship_backto_pro_w[w], ship_back_port_w,
ship_backto_port_day_w[w]]

            # let the rest ships back to supplier
        for ship, days in ship_backto_pro.sample(frac=1).astype(int).iterrows():
            #          break
```

```
            for tp, day in days.iteritems():
                #             break
                d1, d2, port_, speed_1, DAY_ = ship_backto_port_day[ship]
                speed_2 = self.adj_speed(self.distance_port.loc[port_, tp[0]] / (day
- 2 * d2 - d1))
                fc2 = ((0.0019 * (speed_2 ** 2) - 0.045 * speed_2 + 0.3739) * 0.8) *
(day - 2 * d2 - d1)
                shipL.append(['back', port_, ship, 0, day - d2, day - 2 * d2, d1,
speed_1, speed_2,
                              0, 0, tp[0], DAY_, fc2])
                ships_empty.append(ship)
                ship_backto_pro.drop(ship, inplace=True)
                del ship_backto_port_day[ship]
                break
        df_ship = pd.DataFrame(shipL, columns=['direction', 'port', 'ship',
'unpunctual_days', 'initial_date', 'date_producer', 'date_port', 'speed_initial',
'speed_deliver', 'queue_producer', 'queue_port', 'path', 'date', 'fuel_consumption'])
        cost_producer_stock = (producer_fuel * self.data_producer['inv_cost']).sum()
        # cost of inventory management failure
        cost_producer_outlier = len(producer_fuel[producer_fuel >
self.data_producer['upper_limit']]) * self.data_producer[
            'penalty_uplimit'] + \
                            len(producer_fuel[producer_fuel <
self.data_producer['lower_limit']]) * self.data_producer[
                                'penalty_lowlimit']
        #======let the rest ships back to supplier under different scenarios======
        for w in range(1,11):
            for ship_w, days_w in
state_w[w][0].sample(frac=1).astype(int).iterrows():
                #          break
                for tp_w, day_w in days_w.iteritems():
                    #              break
                    d1_w, d2_w, port_w_, speed_1_w, DAY_w_ = state_w[w][2][ship_w]
                    speed_2_w = self.adj_speed(self.distance_port.loc[port_w_,
tp_w[0]] / (day_w - 2 * d2_w - d1_w))
                    fc2_w = ((0.0019 * (speed_2_w ** 2) - 0.045 * speed_2_w + 0.3739)
* 0.8) * (day_w - 2 * d2_w - d1_w)
                        # judge the weather event
                    if port_w_ in EastA:
                        storm_pro = self.data_storm[(self.data_storm['scenario'] ==
w)]
                    else:
                        shipL_w[w].append(
                            ['back', port_w_, ship_w, 0, day_w - d2_w, day_w - 2 *
d2_w, d1_w, speed_1_w,
                             speed_2_w, 0, 0, tp_w[0], DAY_w_, fc2_w])
                        ships_empty_w[w].append(ship_w)
                        ship_backto_pro_w[w].drop(ship_w, inplace=True)
                        del ship_backto_port_day_w[w][ship_w]
                        break
                    if len(storm_pro) == 0:
                        shipL_w[w].append(
                            ['back', port_w_, ship_w, 0, day_w - d2_w, day_w - 2 *
d2_w, d1_w, speed_1_w,
                             speed_2_w, 0, 0,tp_w[0], DAY_w_, fc2_w])
```

```
                    ships_empty_w[w].append(ship_w)
                    ship_backto_pro_w[w].drop(ship_w, inplace=True)
                    del ship_backto_port_day_w[w][ship_w]
                    break
                path_back_days_w = set(range(d1_w +
(math.ceil(self.distance_port.loc[port_w_, tp_w[0]] * 0.55 / speed_2_w)),
                                             d1_w +
(math.ceil(self.distance_port.loc[port_w_, tp_w[0]] * 0.75 / speed_2_w)) + 1))
                storm_pro['influenced_dates_ship'] = storm_pro.apply(lambda x:
path_back_days_w & x['influenced_dates'],
                                                     axis=1)
                storm_pro['influenced_days_ship'] =
storm_pro['influenced_dates_ship'].apply(lambda x: len(x))
                if storm_pro['influenced_days_ship'].max() == 0:
                    shipL_w[w].append(
                        ['back', port_w_, ship_w, 0, day_w - d2_w, day_w - 2 *
d2_w, d1_w, speed_1_w,
                         speed_2_w, 0, 0, tp_w[0], DAY_w_, fc2_w])
                    ships_empty_w[w].append(ship_w)
                    ship_backto_pro_w[w].drop(ship_w, inplace=True)
                    del ship_backto_port_day_w[w][ship_w]
                    break
                speed_2_w = self.adj_speed(speed_2_w)
                speed_2_w_ = ship_speed_2_w * (1 + storm_pro['influenced_speed'])
                distance_delta2_back2pro = (storm_pro['influenced_days_ship'] *
speed_2_w_).sum()
                day_w = d1_w + math.ceil((self.distance_port.loc[port_w_, tp_w[0]]
- distance_delta2_back2pro) / speed_2_w + (storm_pro['influenced_days_ship']).sum())
                fc2_w = ((0.0019 * (speed_2_w ** 2) - 0.045 * speed_2_w + 0.3739)
* 0.8) * (day_w - 2 * d2_w - d1_w)
                shipL_w[w].append(['back', port_w_, ship_w, 0, day_w - d2_w, day_w
- 2 * d2_w, d1_w, speed_1_w, speed_2_w,
                                   0, 0, tp_w[0], DAY_w_, fc2_w])
                delta_fc2 += self.data_probability['probability'].iloc[w-1] * (fc2
- fc2_w)
                ships_empty_w[w].append(ship_w)
                ship_backto_pro_w[w].drop(ship_w, inplace=True)
                del ship_backto_port_day_w[w][ship_w]
                break
        cost_producer_stock_w = 0
        cost_producer_outlier_w = 0
        cost_producer_stock_w += (((producer_fuel_w[w] *
self.data_producer['inv_cost']).sum() - cost_producer_stock) *
                                  self.data_probability['probability'].iloc[w - 1])
        # cost of inventory management failure
        cost_producer_outlier_w += (((len(producer_fuel_w[w][producer_fuel_w[w] >
self.data_producer['upper_limit']]) * \
                                      self.data_producer[
                                          'penalty_uplimit'] + \
                                      len(producer_fuel_w[w][producer_fuel_w[w] <
self.data_producer['lower_limit']]) * \
                                      self.data_producer[
                                          'penalty_lowlimit']) - cost_producer_outlier) *
                                    self.data_probability['probability'].iloc[w - 1])
```

```
        cost_2nd = cost_producer_stock_w + cost_producer_outlier_w
        delta_fc = delta_fc1 + delta_fc2
        self.state = [df_ship, producer_fuel, ship_changeN, delta_fc, delta_shift,
cost_2nd]

    def adj_speed(self, speed):
        '''
        speed control within the limits
        '''
        return min(max(speed, self.data_ship['ship_speed'][0]),
self.data_ship['ship_speed'][1])

    def energy(self):
        '''
        cost calculation
        '''
        df_ship, producer_fuel, ship_changeN, delta_fc, delta_shift, cost_2nd=
self.state
        if len(df_ship) == 0:
            return 10 ** 12
        # storage costs
        cost_producer_stock = (producer_fuel * self.data_producer['inv_cost']).sum()
        cost_producer_outlier = len(producer_fuel[producer_fuel >
self.data_producer['upper_limit']]) * self.data_producer[
            'penalty_uplimit'] + \
                            len(producer_fuel[producer_fuel <
self.data_producer['lower_limit']]) * self.data_producer[
                                'penalty_lowlimit']
        # transportation costs
        a1, a2, a3 = self.data_ship['relation_speed_fuel']
        df_ship.dtypes
        df_ship['fuel_cost'] = df_ship['fuel_consumption'] * df_ship['speed_deliver'] *
self.data_ship['fuel_price']
        cost_ship_transport = df_ship['fuel_cost'].sum(axis=1).sum()
        # port queueing cost
        cost_port_shift = 0
        for diff, (a, b) in self.port_shift_cost.iterrows():
            if diff < 0:
                df_shift = df_ship[df_ship['unpunctual_days'] <
diff]['unpunctual_days'].abs()
                df_shift_day = np.ceil(df_shift / 24)
                cost_port_shift += (a * df_shift_day + b).sum()
            else:
                df_shift = df_ship[df_ship['unpunctual_days'] >
diff]['unpunctual_days'].abs()
                df_shift_day = np.ceil(df_shift / 24)
                cost_port_shift += (a * df_shift_day + b).sum()
        # cost of out of the grace period
        request_out = df_ship[((df_ship['date_port'] - df_ship['day']) <
self.port_days_min) | (
                    (df_ship['date_port'] - df_ship['day']) > self.port_days_max)]
        cost_request_out = self.data_ship['out_grace'] * len(request_out)
```

```
        cost_ship_changeN = self.data_producer['cost_ship_change'] * ship_changeN
        cost = cost_port_shift + cost_ship_transport + cost_producer_stock +
cost_producer_outlier + cost_request_out \
               + cost_ship_changeN + delta_fc * self.data_ship['fuel_price'] +
delta_shift
        return cost

    def save_result(self):
        suffix = 'storm'
        df_ship, producer_fuel, ship_changeN, delta_fc, delta_shift,cost_2nd= state

        # Iteration chart
        pd.Series([k[0] for k in rounds]).plot(figsize=(8, 5))
        plt.savefig('iteration_%s.png' % suffix, dpi=200)
        plt.show()
        plt.close()

        # Gantt Chart
        df_ship_to = df_ship[df_ship['direction'] == 'load'][['port', 'ship',
'date_initial', 'date_producer', 'date_port', 'day']]
        df_ship_back = df_ship[df_ship['direction'] == 'back'][['port', 'ship',
'date_initial', 'date_producer', 'date_port', 'day']]
        fig, ax = plt.subplots(1, 1, figsize=(12, 8))
        ports = df_ship['port'].drop_duplicates().tolist()
        colors = {'Kawasaki': 'lightcoral', 'Kisarazu': 'indianred', 'Tokyo':
'firebrick', 'Busan': 'lightblue',
                  'Tianjin': 'cornflowerblue', 'Caofeidian': 'deepskyblue',
'Shanghai': 'royalblue', 'Dahej': 'wheat',
                  'Dabhol': 'gold'}
        port_del = copy.deepcopy(ports)
        for port, ship, day_spot, day_pro, day_port, _ in df_ship_to.values:
            if port in port_del:
                plt.barh(ship, day_pro - day_spot, left=day_spot, color=colors[port],
edgecolor='k', height=0.5,
                         label=port)
                port_del.remove(port)
            else:
                plt.barh(ship, day_pro - day_spot, left=day_spot, color=colors[port],
edgecolor='k', height=0.5)
            plt.barh(ship, day_port - day_pro, left=day_pro, color=colors[port],
edgecolor='k', height=0.5)
            plt.text(day_port / 2 + day_spot / 2, ship, 'l', ha='center',
va='bottom', fontsize=10)
        for port, ship, day_spot, day_pro, day_port, _ in df_ship_back.values:
            plt.barh(ship, day_spot - day_pro, left=day_pro, color=colors[port],
edgecolor='k', height=0.5)
            plt.barh(ship, day_pro - day_port, left=day_port, color=colors[port],
edgecolor='k', height=0.5)
            plt.text(day_port / 2 + day_spot / 2, ship, 'b', ha='center',
va='bottom', fontsize=10)
        plt.legend()
        plt.grid(True)
        plt.title('scheme')
        plt.xlabel('time')
        plt.ylabel('ship')
        plt.savefig('./ShipScheme_%s.png' % suffix, dpi=500)
```

```
        plt.show()
        plt.close()

        df_ship.columns = ['direction', 'port', 'ship', 'port_advance/delay',
'day_spot', 'day_producer',
                           'day_port', 'speed_producer', 'speed_port',
'queue_producer', 'queue_port', \
                           'path', 'day_request', 'fuel_rate',
'fuel_prc_spot2producer', 'fuel_prc_producer2port']
        # save excel
        with pd.ExcelWriter('Result_%s.xlsx' % suffix) as writer:
            df_ship.to_excel(writer, sheet_name='ship_scheme', index=None)
            producer_fuel.to_excel(writer, sheet_name='producer_story')
            pd.Series([k[0] for k in rounds]).to_excel(writer,
sheet_name='target_func', header=None)
        print(ship_changeN)
        print(delta_shift)
        print(delta_fc)

if __name__ == '__main__':

    self = SearchScheme()
    # iteration
    self.steps = 200
    state, e, rounds = self.anneal()
    self.save_result()
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management
Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/lsm**