
Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

# Ship Scheduling and Network Design for Cargo Routing in Liner Shipping

Richa Agarwal, Özlem Ergun,

To cite this article:
Richa Agarwal, Özlem Ergun, (2008) Ship Scheduling and Network Design for Cargo Routing in Liner Shipping. Transportation Science 42(2):175-196. https://doi.org/10.1287/trsc.1070.0205

TRANSPORTATION SCIENCE
Vol. 42, No. 2, May 2008, pp. 175–196
ISSN 0041-1655 | EISSN 1526-5447 | 08 | 4202 | 0175

informs

DOI 10.1287/trsc.1070.0205




# Ship Scheduling and Network Design for Cargo Routing in Liner Shipping

Richa Agarwal, Özlem Ergun
School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332 {ragarwal@isye.gatech.edu, oergun@isye.gatech.edu}

A common problem faced by carriers in liner shipping is the design of their service network. Given a set of demands to be transported and a set of ports, a carrier wants to design service routes for its ships as efficiently as possible, using the underlying facilities. Furthermore, the profitability of the service routes designed depends on the paths chosen to ship the cargo. We present an integrated model, a mixed-integer linear program, to solve the ship-scheduling and the cargo-routing problems, simultaneously. The proposed model incorporates relevant constraints, such as the weekly frequency constraint on the operated routes, and emerging trends, such as the transshipment of cargo between two or more service routes. To solve the mixed-integer program, we propose algorithms that exploit the separability of the problem. More specifically, a greedy heuristic, a column generation-based algorithm, and a two-phase Benders decomposition-based algorithm are developed, and their computational efficiency in terms of the solution quality and the computational time taken is discussed. An efficient iterative search algorithm is proposed to generate schedules for ships. Computational experiments are performed on randomly generated instances simulating real life with up to 20 ports and 100 ships. Our results indicate high percentage utilization of ships' capacities and a significant number of transshipments in the final solution.

*Key words*: maritime transportation; liner shipping; Benders decomposition
*History*: Received: July 2006; revision received: February 2007; accepted: April 2007. Published online in *Articles in Advance* April 7, 2008.

## Introduction

*Sea cargo* is the freight carried by ships, and it includes anything travelling by sea other than mail, people, and personal baggage. A global sea *carrier* is a person, business, or organization that offers transportation services via the sea on a worldwide basis. A *shipper* is a person or company that is either the supplier or the owner of the cargo that is to be shipped. With rates for sea cargo transportation at approximately one-tenth of air freight rates, fewer accidents and less pollution, maritime transportation is regarded as a cheap, safe, and clean transportation mode, compared with other modes of transportation. Increasing globalization and interdependence of various world economies is leading to a tremendous positive growth in the sea cargo industry. International and domestic trade of many nations depends on this mode of transportation. According to the American Association of Port Authorities (2006), in the United States, which is the largest trading nation in the world for both imports and exports—accounting for nearly 20% of world trade—sea cargo is responsible for moving more than 99% of the international cargo. U.S. ports and waterways handle more than 2.5 billion tons of trade annually, and this volume is projected to double within the next 15 years.

An increase in sea-borne trade worldwide has led to similar trends in the growth of the world fleet. In 2005, world sea-borne trade increased to 7.11 billion tons of loaded goods (a 3.8% annual growth rate), and the world fleet expanded to 960.0 million deadweight tons (a 7.2% increase) (United Nations Conference on Trade and Development 2006). Although the fleet mix and size have changed considerably over time, the efficient utilization of ships remains one of the main determinants of a carrier's profitability. A ship involves a major capital investment, in millions of U.S. dollars, and its daily operating costs can be in tens of thousands of dollars. Hence, development of optimization-based decision support systems is necessary for efficient fleet management.

The sea cargo industry is going through many changes that are reshaping its face. One of the most prominent changes is the increasing use of containers. *Containerized cargo* is the cargo that has been physically and economically stored in a container. Containerization of cargo has revolutionized the sea cargo industry by minimizing port labor and facilitating cargo handling. The dimensions of containers have been standardized. The term *twenty-foot equivalent unit* (TEU) is used to refer to one 20-foot long container. According to Drewry (2001), at the start of 1980s only

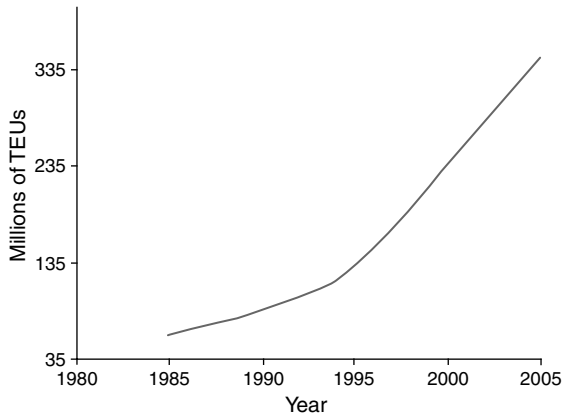


**Figure 1 Growth in Container Movements Worldwide**
*Source. Drewry Container Market Quarterly* 2001.

about 20% of all general cargo shipments were carried in containers. In 2001, this figure had increased to 60%. An IBM white paper (Hingorani, Moore, and Tornqvist 2005) shows that the container shipping market is still growing at 8%–10% per year. This trend is depicted in Figure 1.

A serious challenge associated with containerized shipping is the repositioning of empty containers. Because of the huge imbalance in trade volume on some routes, carriers need to reposition empty containers, incurring significant costs. According to ROI (2002), a 10% reduction in equipment and repositioning costs can potentially increase profitability by 30%–50%.

The sea cargo industry has seen a tremendous growth in number as well as in size of *transshipment ports*. A transshipment port is a port where cargo is transferred from one ship to another. Through a sequence of moves by cranes, cargo is transferred either directly from one ship to another or temporarily stored at the port before being loaded onto outbound ships for further transportation. Use of containers makes this transfer very convenient and cost effective. Transshipment services provide carriers with additional routing options and reduced transit times and act as facilitator of international trade. For example, the Hutchinson terminal in Freeport, Bahamas, has become a major transshipment port among the eastern Gulf Coasts of the United States, the Gulf of Mexico, the Caribbean, South America, and trade lanes to European, Mediterranean, far Eastern, and Australian destinations. Other major international transshipment ports include Singapore, Port Klang in Malaysia, and Hong Kong. In 2003, approximately 30% of worldwide container movements were transshipped, and it is believed that this number is rising. According to the U.S. Customs Service (2003), 80% of all containers handled at the port of Singapore, the world's second largest container port and the world's busiest port in terms of shipping tonnage, are transshipments.

Collaboration among sea carriers is not new. Carriers used *conferences* as a means to curb competition and control tariff rates in the market, as early as 1875. More recently, carriers are forming strategic alliances that allow them to realize economies of scale, extend their customer base, and increase asset utilization while providing customers with more frequent sailings and faster transit times (Song and Panayides 2002). Since 1990, when Sea-Land and Maersk introduced the alliance system and began sharing ships in the Atlantic and Pacific, strategic alliances have become increasingly common. Today smaller alliances collaborate to form even bigger alliances; for example, the Grand Alliance and the New World Alliance laid down the foundation for cooperating in 2006. This trend of consolidating fleets and service routes demands better decision support systems to control a large fleet of ships and to solve large-scale scheduling and optimization problems.

Christiansen, Fagerholt, and Ronen (2004) describe in detail the division of the global shipping industry into three different modes of operation: *industrial*, *tramp*, and *liner*. In industrial shipping, the shipper owns the ships and aims to minimize the total shipping costs. In tramp shipping, a carrier engages in contracts with shippers to carry bulk cargo between specified ports within a specific time frame. Additional cargo (if any is available in the market) is picked depending on the fleet capacity to maximize revenue. In liner shipping, a carrier decides on a set of trips, makes the schedule available to shippers, and operates on it. Thus, one can identify industrial shipping with owning a car, tramp shipping with a taxi service, and liner shipping with a bus service with definite schedules and a published itinerary.

The focus of this paper is liner shipping. Liner shipping mainly involves carrying containerized cargo on regularly scheduled service routes. Liner services involve higher fixed costs and administrative overhead than, for example, tramp shipping because tramps usually wait until they are full before departing from a port, whereas liner services promise to depart on predetermined schedules regardless of whether the ship is full. The number of ships required for a given liner service route is determined principally by the frequency required on the service route, the distance travelled by a ship on the route, and the speed of the ship. For example, a weekly liner service between New York and Hamburg may require four ships to maintain the necessary frequency.

As observed by Christiansen, Fagerholt, and Ronen (2004), liner shipping is growing at a high pace because of the increasing global container traffic. In the

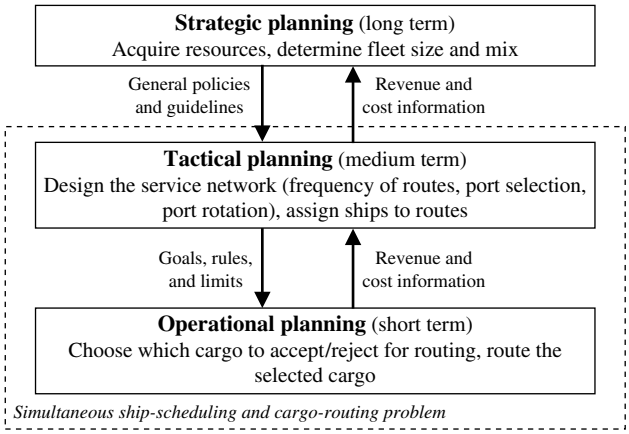


**Figure 2 Planning Levels for Liner Shipping**

United States, in 2003, liner shipping with its network of ships, containers, port terminals, and information systems handled more than 60% of the total sea-borne trade. According to Barry Rogliano Salles-AlphaLiner (2006), between January 2000 and January 2006, the TEU capacity deployed on global liner trades rose from 5,150,000 TEUs to 9,135,000 TEUs, a 77.4% increase.

Liner shipping involves decision making at strategic, tactical, and operational planning levels. Figure 2 outlines the key decisions that need to be made at different levels of the planning horizon.

In the strategic planning stage, the optimal number and mix of ships in a fleet are determined. Given that owning a ship involves a huge capital investment (usually in millions of U.S. dollars) and the cost of holding a 2,000 TEU ship idle is \$20,000–\$25,000 per day, the strategic level decisions are extremely important.

In the tactical planning stage, the service network is designed by creating the ship routes, that is, the sequence of port visits by a given fleet and the assignment of ships to these routes. Ships move in *cycles* from one port to another, following the same port rotation for the entire planning horizon. To maintain a customer base and to provide customers with a regular schedule, most carriers have at least one departure each week from each port on a service route (i.e., a cycle). This requires that the number of ships that operate on a cycle be at least equal to the number of weeks that it takes to complete the cycle. Some cycles, such as those connecting Asia to North America, may take up to eight weeks to complete, which means that a carrier requires at least eight ships to introduce a new service on such a route. The problem of designing the service network of a carrier is referred to as the *ship-scheduling problem*.

In the operational planning stage, a carrier makes decisions regarding which cargo to accept or reject for servicing and which path(s) to use to ship the selected cargo. This is referred to as *the cargo-routing problem*. A carrier may elect not to transport some cargo, either because it is not profitable or because there is other cargo, perhaps at other ports, that is relatively more profitable. A cargo starts its trip from an in-land location and arrives at its *origin port*. This network, which utilizes trucks, railroads, or waterways to bring cargo from an inland location to its origin port, is known as the *feeder network*. Cargo then moves from its origin port to its *destination port*, possibly after visiting some intermediate ports. From there it is taken to its final in-land destination using another feeder network. Some of the intermediate ports that a cargo visits during its journey from the origin port to the destination port may act as transshipment ports, where cargo is transferred from one ship to another.

The decisions made at one planning level affect the decision making at other planning levels as well. The decisions at the strategic level set the general policies and guidelines for decisions at the tactical and operational levels. Similarly, the decisions at the tactical level set the capacity limitations and network structure for the operational planning level. In the reverse direction, the information on cost and revenue generated by the system given the set parameters provides the much-needed feedback for decision making at a higher level.

Over the years, the sea cargo industry has been conservative in terms of adopting new decision support systems. It has a long tradition of manual planning by experienced planners. Moreover, in general, ship scheduling involves a large variety of problems. Hence, mostly tailor-made models for specific problems with specialized constraints and objectives are available. Furthermore, most of the available literature has been developed for industrial and tramp shipping (Christiansen, Fagerholt, and Ronen 2004). Because of the many differences in modelling and problem structure itself, it is difficult to draw comparisons between the existing literature.

We next briefly review a set of representative papers related to container and liner shipping. For a comprehensive review of literature on ship scheduling and cargo routing, we recommend Ronen (1983) for the work done before 1983, Ronen (1993) for the decade 1982–1992, and Christiansen, Fagerholt, and Ronen (2004) for the last decade.

Rana and Vickson (1991) provide a nonlinear integer program to maximize total profit by finding an optimal sequence of ports to visit for each containership and an optimal number of cargo units to be transported between each pair of ports by each ship. They allow multiple pickups and deliveries on their ships. However, a special network structure with a restriction on loading and unloading of cargo at the end ports is considered. Furthermore, the model does

not consider transshipments by not allowing cargo to be carried on ships that do not visit either the port of origin or the destination port. They report that their algorithms solve instances with three ships and up to 20 ports within an hour.

Fagerholt (1999) considers the liner shipping problem in a special network where all cargo is transported from a set of production ports to a single depot. The problem is solved by first generating all feasible single-ship routes and then solving a set partitioning problem. Again, the model does not allow for transshipments. Although a weekly frequency constraint is imposed on the operated routes, the feasible routes for the particular problem considered have a maximum route time of only one week. Thus, on any of the feasible routes a single ship can maintain weekly frequency. Instances with up to 19 ships on a network with up to 40 ports are reported to be solved within a couple of seconds.

Finally, Perakis (2002) provides a review of linear and integer programming models that only consider the deployment of a fleet of liner ships with different ship types on a set of predetermined routes with targeted service frequencies to minimize operating and lay-up costs.

As noted earlier, decisions made at one planning level affect decision making at other planning levels. Given a fleet size and mix, the service network laid at the tactical planning level governs which routes can be formed at the operational planning level to route cargo. The cargo picked at the operational planning level and the routes selected determine the cost and revenue that can be generated and thus the profitability of the given service network. These two problems are highly interdependent; thus, it is important that they be studied in an integrated framework.

The contribution of this paper is twofold. First, a new mixed-integer programming (MIP) model is presented for the integrated ship-scheduling and the cargo-routing problem for containerized cargo. As shown in Figure 2, we refer to this problem as *the simultaneous ship-scheduling and cargo-routing problem*. Second, because the proposed integer program is too large to be solved economically by general mixed-integer programming codes, we develop algorithms to solve it efficiently.

Our model handles many relevant constraints and emerging trends that appear in practice but that have not been considered previously in the literature. For instance, customers expect carriers to maintain a regular schedule by following at least a weekly frequency on their routes. To the best of our knowledge, this constraint has not been considered in the literature in its full generality. We successfully impose the weekly frequency constraint at the ports visited by a carrier. As is common in container shipping, we allow multiple pickups and deliveries on our ships; that is, we allow containers loaded at one port to have more than one port of destination. Moreover, a fleet usually consists of various ship types with different characteristics that may change over time. We consider a heterogeneous fleet with ships of different sizes, cost structures, and speeds. In the literature, although there are references (see, for example, Fagerholt 1999) to models with a heterogeneous fleet, most of these models consider ships with identical service speeds. Repositioning of empty containers efficiently is a big problem in liner shipping. Our model, with some modifications, has the flexibility to incorporate empty container repositioning. Empty container repositioning has been studied by Shen and Khoong (1995) and Cheung and Chen (1998) also; however, these papers consider only the movement of empty containers on a given network. We also allow cargo routes to encompass a combination of cycles rather than a single cycle (with some simplifying assumptions on the cost of transshipment), thus providing carriers with increased routing opportunities. We are not aware of any earlier results on transshipment of containers at an intermediate port from one ship to another. These features allow the use of our model in a wide variety of settings.

In the most general approach for solving ship-scheduling problems to date, Fagerholt (1999) generates a set of feasible schedules by including nonlinear and intricate constraints, and then solves a set-partitioning problem. Our goal in this paper is to model the simultaneous ship-scheduling and cargo-routing problem in its generality and solve it for large-scale instances. Hence, rather than being limited to an initial set of routes or exhaustively listing all the routes for ships, we design algorithms that exploit the separability of the problem to iteratively generate good cycles for ships and efficiently route the demand. More specifically, we utilize the fact that the ship-scheduling problem can be reduced to a cycle generation problem and the cargo-routing problem can be reduced to a multicommodity flow (MCF) problem. We develop a greedy heuristic, a column generation-based algorithm, and a two-phase Benders decomposition-based algorithm and compare the computational effectiveness and efficiency of these approaches.

Our computational results are encouraging and establish that some of the algorithms developed can be used to solve larger instances, compared with the literature, in terms of fleet size that arise as a result of collaborations and mergers in the sea cargo industry. We report computational results on problem instances with up to 100 ships and 20 ports.

The rest of the paper is organized as follows. The next section introduces our notation, the mathematical formulation, and a note on the complexity of

the problem. Three different algorithms, a greedy heuristic, a column generation-based algorithm, and a Benders decomposition-based algorithm are discussed in §2. Section 3 provides various algorithmic and implementation details. Computational experiments are presented in §4. Conclusions and directions for future work are discussed in the final section.

## 1. Problem Description

We now present a mathematical formulation for the simultaneous ship-scheduling and containerized cargo-routing problem after introducing our notation and a space-time network.

Let $\mathcal{P}$ denote the set of ports. We will treat demand as a set of commodities with a positive supply at the origin ports and a positive demand at the destination ports. Each such commodity is characterized by an origin port, $o$; a destination port, $d$; the day of the week, $i$, when the supply is available at port $o$; the maximum demand (in TEUs) that may arise at port $d$, $D^{(o,d,i)}$; and the revenue obtained by satisfying one TEU of the demand, $R^{(o,d,i)}$. We use the triplet $(o, d, i)$ to identify a particular demand, and we let $\Theta$ be the set of all such triplets. We call such triplets *demand triplets*.

A carrier typically has several different types of ships in its fleet. Each *ship type* usually has a different capacity and speed and specifies the characteristics of a group of ships that are considered identical. We denote by $\mathcal{A}$ the set of all ship types and use the index $a$ to represent a particular ship type. We associate the following information with each ship type $a \in \mathcal{A}$: $T^a$ denotes the capacity of a ship in TEUs for a ship of type $a$; for ports $p, q \in \mathcal{P}$, $l^a_{(p,q)}$ denotes the number of days it takes a ship of type $a$ to sail from port $p$ to port $q$; and $N^a$ denotes the number of ships of type $a$ available in the given fleet.

Given that the temporal aspects of the problem are important, we formulate the simultaneous ship-scheduling and cargo-routing problem as an MCF problem with side constraints on a *space-time* network. Furthermore, we use days as our time units in the space-time network, because in general the transoceanic routes do not visit more than one port in a given day. Let $G = (V, E)$ be a directed space-time network with vertex set $V$ and edge set $E$. Each vertex $v \in V$ represents a port, $port(v)$, on a day of the week, $time(v)$. That is, for each port $p \in \mathcal{P}$ we create seven vertices in $V$. For notational convenience, we associate a subscript with each vertex, that is, $v = v_{(p,i)}$ where $port(v) = p$ and $time(v) = i$. We refer to the vertices of $G$ either by $v$ or $v_{(p,i)}$, depending on the ease of exposition.

The network $G = (V, E)$ contains three types of edges. The first is the set of *ground edges*. For every ship type $a \in \mathcal{A}$, we construct ground edges by connecting nodes $v_{(p,i)}$ to $v_{(p,i+1)}$ $\forall p \in \mathcal{P}$ and $1 \le i \le 6$. We also connect $v_{(p,7)}$ to $v_{(p,1)}$ $\forall p \in \mathcal{P}$. For a ship, these edges represent an overnight stay at a port, and for cargo they represent an overnight stay at a port either on ground or on the same or a different ship before continuing. Next, for every ship type $a \in \mathcal{A}$ and pair of ports $p, q \in \mathcal{P}$, we construct *voyage edges*, $(v_{(p,i)}, u_{(q,j)})^a$ for $1 \le i, j \le 7$ such that $i - j = l^a_{(p,q)} \mod (7)$. The voyage edges represent the movement of ships and cargo from one port to another at a given speed. Finally, we create a set of *fictitious edges*, $(v_{(d,j)}, u_{(o,i)})$, for all demand triplets $(o, d, i) \in \Theta$ and $1 \le j \le 7$. An edge $(v_{(d,j)}, u_{(o,i)})$ only allows the flow of commodity $(o, d, i)$ on it and enables us to view the flow of commodity $(o, d, i)$ in the network as a circulation. Let us denote the set of all ground edges by $E_g$, the set of all ground edges for ship type $a$ by $E^a_g$, the set of all voyage edges by $E_v$, the set of all voyage edges for ship type $a$ by $E^a_v$, and the set of all fictitious edges by $E_f$. That is, $E_g = \bigcup_{a \in \mathcal{A}} E^a_g$, $E_v = \bigcup_{a \in \mathcal{A}} E^a_v$ and $E = E_g \cup E_v \cup E_f$. We also use the following additional notation: $InEdges(v)$ denotes the set of incoming edges into vertex $v$ and $OutEdges(v)$ denotes the set of outgoing edges from vertex $v$; for an edge $e = (u, v) \in E$, $tail(e)$ denotes vertex $u$ and $head(e)$ denotes vertex $v$. Figure 3 represents a space-time network with four ports and two cycles, $C_1$ and $C_2$. Note that port C acts as a transshipment port to transport cargo from port A to port D.

The length of voyage edge $e = (v, u) \in E^a_v$, $l^a_e$, is equal to the number of days it takes for a ship of type $a$ to get from $port(v)$ to $port(u)$. We also let $l_e = 1$ for $e \in E_g$ and $l_e = 0$ for $e \in E_f$. The capacity of an edge represents the total amount of flow in TEUs that the edge can sustain. Ground edges at a port may have finite or infinite capacity, depending on whether

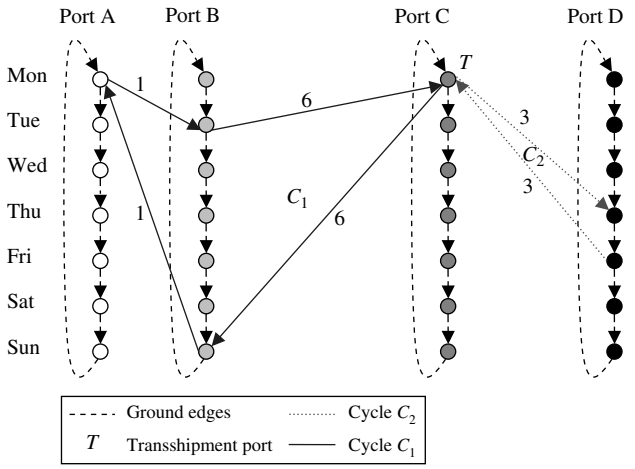


**Figure 3 Network with Four Ports and Two Cycles**

*Note.* Numbers on edges represent the length of the edge.

we wish to impose a limit on the amount of cargo a port can handle/store. Capacity on a voyage edge depends on the number of ships (and their capacities) that cover the edge.

Various fixed and variable costs are associated with the simultaneous ship-scheduling and cargo-routing problem. Although some of these costs are incurred by ships, others are incurred by cargo. For the costs related to ports, we let $c_v^{s,a}$ be the one-time cost incurred by a ship of type $a \in \mathcal{A}$ when visiting $port(v)$ and $c_v^c$ be the total cost that a TEU of cargo incurs at $port(v)$ per day. The port visit costs may be different at different ports. Similarly, $c_e^{s,a}$ reflects the cost of operating a ship of type $a$ on edge $e$ in deep sea if $e \in E_v^a$ and the cost of an overnight stay for a ship of type $a$ at $port(head(e))$ if $e \in E_g^a$. For cargo, $c_e^c$ for $e \in E_v$ reflects the cost of shipping a TEU of cargo on edge $e$ and $c_e^c$ for $e \in E_g$ reflects the cost of storing or holding a TEU of cargo at $port(head(e))$. Fictitious edges are assigned zero costs.

We make sure that the port visit cost is incurred only once at each port even if a ship makes an overnight stay at the port. To account properly for the port visit cost in the time expanded network, we subtract the port visit cost for ship type $a$ at the port from the ship cost $c_e^{s,a}$ on the ground edge $e$. Thus, if a ship makes an overnight stay at a port, then although the port visit cost is counted twice via the node costs, it is subtracted once via the ground edge cost. Idling of a ship at a port is penalized by imposing the overnight stay cost on the ship. Long stay of cargo at a port that is different from its port of destination is penalized through the holding cost at the port. In this paper, we are using days of the week as the level of time discretization; thus, we assume that if ships on two different cycles meet at a port on the same day, transshipment can occur in both directions (i.e., cargo can be transferred from either ship on to the other). By considering finer discretization of time, one can account for smaller time windows during which ships at a port meet for transshipments and determine if transshipment is possible in only one direction (without incurring the holding cost).

Given that in the space-time network the level of discretization is in days, a commodity that becomes available at port $o$ on the $i$th day of the week is represented as a supply on vertex $v_{(o,i)}$ in the network. We assume that supply appears at vertex $o$ (for destination $d$) on the same day of week every week. We believe assuming that an average amount of demand arises at a port on a given day is reasonable in our context because we are considering a tactical model. Because the demand from week to week is taken to be the same for a carrier, the service characteristics may also remain the same every week. To this end, we assume that given any cycle, the weekly frequency of the cycle is maintained using ships of the same type. We characterize a cycle $C$ by the port rotation that it follows, the days of the week it visits each port in its rotation, the type of ship that is used to service $C$, and the number of ships $L_C$ it takes to maintain a weekly frequency on $C$. Because of the indivisibility of ships, $L_C$ is integral, by definition. A cycle is said to be *feasible* if it satisfies prespecified rules in terms of the number of ships required to maintain a weekly frequency on the cycle and the number of ports visited. We denote the set of all feasible cycles for ship type $a$ by $\mathcal{C}^a$. Mathematically, for a cycle $C \in \mathcal{C}^a$, $L_C = \lceil \sum_{e \in C} l_e^a / 7 \rceil$. Whenever necessary we represent a cycle $C$ by a sequence of vertices; for example, a cycle from vertex $v_1$ to vertex $v_r$ via $v_2, \ldots, v_{r-1}$ is represented as $C = v_1 - v_2 \cdots v_r - v_1$. Cost of a cycle $C \in \mathcal{C}^a$ is denoted by $Cost_C$ and is calculated as $Cost_C = \sum_{v \in C} c_v^{s,a} + \sum_{e \in C} c_e^{s,a}$.

### 1.1. Mathematical Model

We now present a mixed-integer programming formulation for the simultaneous ship-scheduling and cargo-routing problem. Our formulation has two sets of variables. First, for every feasible cycle $C$ we define a binary variable $x_C$. $x_C = 1$ if a weekly frequency is maintained on cycle $C$ and is 0 otherwise. The $x_C$ variables are taken to be binary rather than integer, as the possibility of departing two ships of same type from a port following the same port calls on the same days of the week is highly unlikely.

Next, we define nonnegative continuous variables representing the flow on edges. For each edge $e \in E_g \cup E_v$ and each triplet $(o, d, i) \in \Theta$ we define $f_e^{(o,d,i)}$ to denote the flow of commodity represented by $(o, d, i)$ on edge $e$. For a fictitious edge $e = (v_{(d,j)}, u_{(o,i)}) \in E_f$, a single flow variable, $f_e^{(o,d,i)}$, for commodity $(o, d, i)$ is defined because the flow of other commodities on this edge is not allowed. Note that we let the flow variables be continuous because adjusting for picking a fractional container does not influence the solution quality much for the purposes of our tactical model.

Before presenting the model we summarize the following additional assumptions: Assumptions 1–3 are for the clarity of exposition and in no way restrict the usability of our model; however, Assumption 4 is more significant.

Assumption 1. *Capacity on ground edges is assumed to be infinite; that is, we do not put any restriction on the amount of cargo that can be handled/stored at a port.*

Assumption 2. *We assume that all costs on cargo can be modelled via edge costs; that is, we set* $c_v^c = 0 \ \forall v \in V$.

Assumption 3. *We assume that all cargo is available in identical* 1*-TEU containers. Thus,* $D^{(o,d,i)}$ *represents the number of containers of a commodity required at port d that become available at port o on day i of the week.*

Assumption 4. *Finally, we do not consider the costs involved with transferring cargo from one ship to another. As discussed before, cargo incurs holding costs whenever it stays at a port overnight. To account for transshipment costs correctly, we need to distinguish between the capacity provided by different cycles on the same edge of the network. If the edges are duplicated for all the feasible cycles, it will increase the size of the graph tremendously. Thus, it is hard to account correctly for transshipment costs if the network is not known* (*i.e., the cycles to be operated have not been selected*). *Because our aim is to consider the network design and cargo-routing problems simultaneously and generate feasible cycles as a subproblem, we ignore the transshipment costs for now.*

In §4.4 we present a computational study to discuss the effects of transshipment costs on cargo routing decisions once a set of cycles, to be operated by the given fleet, has been selected.

The simultaneous ship-scheduling and cargo-routing problem can be formulated as the following mixed-integer program:

$$\text{(SSSCR)}: \quad \max \sum_{(o,d,i)\in\Theta} \sum_{j=1}^{7} R^{(o,d,i)} f^{(o,d,i)}_{(v_{(d,j)},\, v_{(o,i)})} - \sum_{(o,d,i)\in\Theta} \sum_{e\in E} c_e^c f_e^{(o,d,i)} - \sum_{a\in\mathcal{A}} \sum_{C\in\mathcal{C}^a} Cost_C x_C \quad (1)$$

such that

$$\sum_{e\in InEdges(v)} f_e^{(o,d,i)} - \sum_{e\in OutEdges(v)} f_e^{(o,d,i)} = 0 \qquad \forall\, v\in V,\ \forall\,(o,d,i)\in\Theta \quad (2)$$

$$\sum_{(o,d,i)\in\Theta} f_e^{(o,d,i)} - \sum_{a\in\mathcal{A}} \sum_{\{C\in\mathcal{C}^a:\, e\in C\}} T^a x_C \le 0 \quad \forall\, e\in E_v \quad (3)$$

$$\sum_{j=1}^{7} f^{(o,d,i)}_{(v_{(d,j)},\, v_{(o,i)})} \le D^{(o,d,i)} \quad \forall\,(o,d,i)\in\Theta \quad (4)$$

$$\sum_{C\in\mathcal{C}^a} L_C x_C \le N^a \quad \forall\, a\in\mathcal{A} \quad (5)$$

$$x_C \in \{0,1\} \quad \forall\, C\in\mathcal{C}^a,\ \forall\, a\in\mathcal{A} \quad (6)$$

$$f_e^{(o,d,i)} \ge 0 \quad \forall\, e\in E,\ \forall\,(o,d,i)\in\Theta. \quad (7)$$

We now explain the above formulation. The objective function (1) maximizes the net profit by subtracting the sum of operating costs from the revenue generated. The first term in the objective function denotes the total revenue generated by transporting cargo between various origin and destination pairs. The second term captures the cost incurred by cargo during its routing from the origin port to the destination port. The third term denotes the total cost of operating ships on the selected cycles.

Constraint (2) is a flow balance constraint at every vertex of the space-time network. It ensures that the total flow into vertex $v$ of each commodity $(o,d,i)\in\Theta$ is equal to the total flow out for the same commodity. Constraints (3) and (4) are capacity constraints on the edges. Constraint (3) requires that the total flow on a voyage edge be less than the sum of the capacities of ships servicing that edge. Constraint (4) models that the total flow of a given commodity from an origin port to a destination port must be less than the demand at the destination port. Note that we do not have a capacity constraint on ground edges because of Assumption 1. Constraint (5) requires that for each fleet type, we do not use more ships than are available. Note that if cycle $C\in\mathcal{C}^a$ is selected, i.e., $x_C, = 1$, then it will utilize $L_C$ ships of type $a$ to maintain a weekly frequency. Finally, (6) denotes $x_C$ as binary variables, and (7) denotes $f_e^{(o,d,i)}$ as nonnegative continuous flow variables.

### 1.2. Hardness of the Problem

The decision version of the simultaneous ship-scheduling and cargo-routing problem is in NP; that is, given a set of cycles to be operated and a flow of cargo on the edges, it can be determined in polynomial time whether the total revenue generated is greater than a given constant $K$. We show the NP completeness of the problem by reducing a well-known NP-complete problem, *0–1 Knapsack*, into a simultaneous ship-scheduling and cargo-routing problem.

The decision version of the 0–1 Knapsack problem is defined as follows: Given set $N=\{1,2,\ldots,n\}$, integers $K$, $c_i$, and $w_i$, for every $i\in N$ is there a subset $S$ of $N$ such that $\sum_{i\in S} w_i \le W$ and $\sum_{i\in S} c_i \ge K$.

Theorem 1. *The decision version of the simultaneous ship-scheduling and cargo-routing problem is NP-complete.*

Proof. Suppose there are $W$ identical ships with capacity $T$ TEUs each. Construct a sea cargo network as follows. For each $i\in\{1,2,\ldots,n\}$ construct two ports, a demand port($d_i$) with demand $c_i$ TEUs and an origin port ($o_i$). Let a ship in the fleet take $w_i/2$ weeks to sail from port $o_i$ to port $d_i$. Assume symmetric distances between ports and that the distances between $o_i$ and $d_j$, $\forall 1\le i\ne j\le n$, is large. Thus $w_i$ ships are needed to maintain weekly frequencies on cycles $C_i = o_i - d_i - o_i$ for $1\le i\le n$, and all other cycles are infeasible. Let $T=\max_{i\in N} c_i$ and let revenue generated by satisfying unit demand between any *o-d* pair be 1. Assume there are no operating costs involved. Observe that:

- All feasible cycles are disjoint.
- A cycle $C_i$ needs $w_i$ ships to maintain weekly frequency and can generate $c_i$ units of revenue.
- If a set $S\subset N$ of cycles is chosen to be serviced, then $\sum_{i\in S} w_i \le W$ and a total of $\sum_{i\in S} c_i$ units of revenue is generated.

It follows easily now that a set of chosen cycles, $S$, will give a revenue of $K$ units or more if and only if the 0–1 Knapsack problem has a feasible solution. Thus, the 0–1 Knapsack can be solved by solving a SSSCR problem. □

## 2. Solution Methodology

The linear program given by (1)–(7) contains a large number of variables even for moderate-size problems. The large size of the model is a direct result of the exponential number of possible feasible cycles. Furthermore, each demand triplet adds a set of flow variables to the MIP model. An interesting observation, however, is that if we determine the set of cycles to be operated for each fleet type, that is, given nonnegative values $\bar{x}_C$ satisfying fleet availability constraints (5), model (1)–(7) reduces to the following *MCF problem*, where each demand triplet is considered a different commodity.

$$\text{(MCF):}\quad \max \sum_{(o,d,i)\in\Theta}\sum_{j=1}^{7} R^{(o,d,i)} f^{(o,d,i)}_{(v_{(d,j)},\,v_{(o,i)})} - \sum_{(o,d,i)\in\Theta}\sum_{e\in E} c^c_e f^{(o,d,i)}_e \tag{8}$$

such that

$$\sum_{e\in InEdges(v)} f^{(o,d,i)}_e - \sum_{e\in OutEdges(v)} f^{(o,d,i)}_e = 0 \qquad \forall v\in V,\ \forall (o,d,i)\in\Theta \tag{9}$$

$$\sum_{(o,d,i)\in\Theta} f^{(o,d,i)}_e - \sum_{a\in\mathcal{A}}\sum_{\{C\in\mathcal{C}^a:\,e\in C\}} T^a \bar{x}_C \le 0 \quad \forall e\in E_v \tag{10}$$

$$\sum_{j=1}^{7} f^{(o,d,i)}_{(v_{(d,j)},\,v_{(o,i)})} \le D^{(o,d,i)} \quad \forall (o,d,i)\in\Theta \tag{11}$$

$$f^{(o,d,i)}_e \ge 0 \quad \forall e\in E,\ \forall (o,d,i)\in\Theta. \tag{12}$$

Note that (8)–(12) is a linear program with no integrality constraints, as it only involves the flow variables $f^{(o,d,i)}_e$. Let $\pi = \{\pi^{(o,d,i)}_v\colon\ \pi^{(o,d,i)}_v$ unrestricted, $\forall v\in V,\ \forall (o,d,i)\in\Theta\}$, $\lambda = \{\lambda_e\colon\ \lambda_e \ge 0\ \forall e\in E_v\}$, and $\omega = \{\omega^{(o,d,i)}\colon\ \omega^{(o,d,i)} \ge 0\ \forall (o,d,i)\in\Theta\}$ be the set of dual variables associated with constraints (9), (10), and (11), respectively.

Next, we present three heuristic algorithms that exploit the above observation to solve the SSSCR problem. First, we provide a simple greedy heuristic that selects good cycles one by one and then assigns cargo to routes. Then we present a column generation-based algorithm that generates a pool of good cycles and then selects the best cycles among these while also routing cargo in the network. Finally, we present the details of a more involved Benders decomposition-based algorithm. Figure 4 presents an outline of the three algorithms.

### 2.1. Greedy Algorithm

Let $S$ represent the set of cycles that are in operation; that is, ships have been assigned to maintain weekly frequencies on cycles in set $S$. The desirability or the value of cycle $C$ depends on the revenue generated by routing flow on ships employed in $C$, the number of ships required to maintain weekly frequency on $C$, and the various costs involved in operating the cycle $C$. The marginal value of cycle $C$ also depends on the cycles already present in set $S$. Thus,

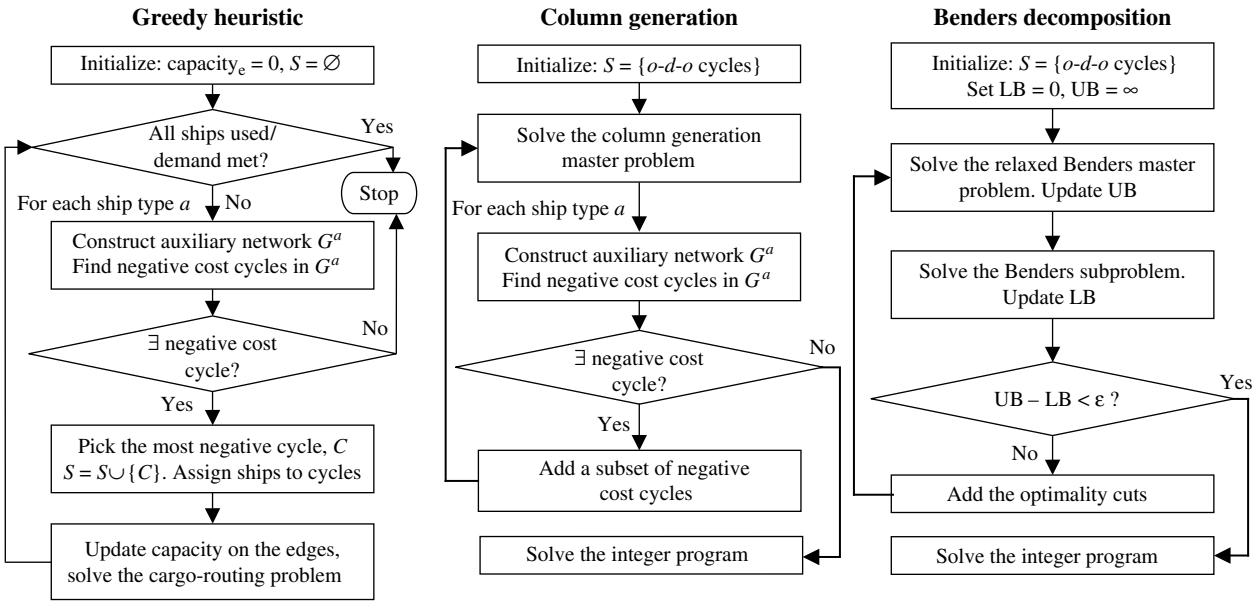


**Figure 4 Outline of the Three Algorithms Considered**

a greedy selection of cycles must take into account the set of existing operational cycles and demand triplets $(o, d, i) \in \Theta$.

Let $A^a$, $\forall a \in \mathcal{A}$, represent the number of ships of type $a$ that are currently available. The greedy algorithm starts with an empty set of selected cycles. To find profitable cycles an auxiliary network $G^a = (V^a, E^a)$ is created utilizing dual information from the solution of the MCF problem to assign edge costs. $G^a$ is constructed for each ship type $a$ such that $V^a = V$ and $E^a = E^a_v \cup E^a_g$. Each edge $e \in E^a$ is assigned a cost $c_e = c^{s,a}_e + \lambda_e$, and each vertex, $v \in V^a$ is assigned a cost $c_v = c^{s,a}_v$. For every ship type, the algorithm then finds a minimum cost cycle in the auxiliary network by using a procedure *FindCycle*$(G^a)$. Details of this procedure will be provided in §3. Finally, if feasible, the cycle with minimum cost is selected and a suitable number of ships, to maintain weekly frequency, is assigned to it. The process is repeated while there are ships to be assigned.

### 2.2. Column Generation-Based Algorithm

Although the greedy algorithm is simple and provides a feasible solution quickly, it is not very effective. It works with a very small set of feasible cycles, and once a feasible cycle is generated it is picked in the final solution without any further considerations. Next, we propose a column generation-based algorithm that iteratively generates a good pool of profitable cycles for solving the linear programming (LP) relaxation of (1)–(7).

Column generation is an effective way of solving linear programs with a large number of columns (see Bertsimas and Tsitsiklis 1997 for an introduction). Rather than enumerating all the columns explicitly, it begins by solving a restricted problem (called the master problem) with a select set of columns. A subproblem is solved to generate "attractive" columns, and they are subsequently added to the master problem. The process is repeated until no further profitable columns can be generated. The column generation technique has been used successfully to solve many large-scale optimization problems; please see (Barnhart et al. 1994) for an example in solving airline crew assignment problems.

To solve the LP relaxation of SSSCR, the master problem in the column generation is initialized by restricting the set of cycle-selection variables to one simple cycle for every demand triplet. At every step of the column generation process, the master problem is solved to find the best value for all the decision variables. The pricing subproblem for the column generation is equivalent to identifying negative cost cycles in an auxiliary network for every ship type. The auxiliary network $G^a = (V^a, E^a)$ is constructed for each ship type $a$ such that $V^a = V$ and $E^a = E^a_v \cup E^a_g$. Dual variable values from the master problem are used to assign costs to the edges and the vertices of the auxiliary network. Each edge $e \in E^a$ is assigned a cost $c_e = c^{s,a}_e + T^a\lambda_e + (l_e/7)\sigma^a$ and each vertex $v \in V^a$ is assigned a cost $c_v = c^{s,a}_v$ so that negative cost cycles in $G^a$ correspond to columns with positive reduced costs in the master problem. Note that since SSSCR is a maximization problem, profitable columns are the ones with positive reduced cost. Procedure *FindCycle*$(G^a)$ (described in §3) is used to identify negative cost cycles in the auxiliary network and corresponding columns are added to the master problem. The process is continued until no new cycles can be found. Finally, integrality constraints are imposed on the cycle-selection variables and a branch-and-bound framework is used to obtain an integer solution for the SSSCR problem. No new columns are generated during the branch-and-bound phase. Different branching rules, with different advantages, can be devised to obtain the integer solution. For example, branching on the largest feasible cycle forces many other binary variables also to satisfy the integrality constraints. However, we use the variable that affects the solution quality the most as the variable to branch on, giving preference to the up $(x_C = 1)$ branch, because this strategy performed the best in our computational experiments.

### 2.3. Benders Decomposition-Based Algorithm

As the number of ports, ships, and demand triplets increases, solving model (1)–(7) by column generation becomes increasingly difficult. The number of constraints and variables increases with the increase in the number of ports, ships, and demand triplets, in the column generation master problem. We next decompose the LP relaxation of the model using Benders decomposition to obtain a pair of problems that utilize the separability of SSSCR. The decomposition results in both a master problem, where the number of variables increases as the number of cycles increases, and a subproblem, where the number of constraints increases as the number of demand triplets increases. Thus, the effect of the increase in problem size is divided between a master problem and a subproblem. Further, this decomposition is used to solve the LP relaxation effectively, and the solution is embedded in a branch-and-bound approach to obtain an integer solution.

Benders decomposition (Benders 1962) is a popular technique to solve mixed-integer linear programming problems with linking constraints. This approach is useful when the master problem has all the integer variables and it is difficult to treat them in subproblems. The solution process iterates between an integer

master problem and subproblem(s). The master problem passes on the value of integer variables to subproblem(s), and subproblem(s) generate cuts (feasibility and optimality) to pass back to the master problem. Although this approach has proved to be suitable for many problems, it has the drawback that an integer master problem has to be solved at each iteration. McDaniel and Devine (1977) proposed a modification to this approach in which the solution of a sequence of integer programs is replaced by the solution of a sequence of linear programs and a few integer programs.

The basic techniques of McDaniel and Devine (1977) and their modifications have been used successfully to solve many hard problems. Florian et al. (1976) used the techniques to solve an engine scheduling problem, and Vander Wiel and Sahinidis (1996) used them for solving a time-dependent travelling salesman problem. More recently, these techniques have been used successfully to solve locomotive car-assignment (Cordeau, Soumis, and Desrosiers 2000, 2001a) and aircraft-routing and crew-scheduling (Cordeau et al. 2001b) problems. For these problems, enormous time reductions and significant improvements in solution quality were achieved by first relaxing the integrality constraints in the master problem. After the relaxation is solved to acceptable time or optimality criteria, the integrality constraints are introduced back into the master problem. We now present the use of Benders decomposition method to solve the SSSCR problem.

**2.3.1. Benders Reformulation.** As noted earlier for given nonnegative values $\bar{x}_C$ satisfying fleet constraints (5), the LP relaxation of model (1)–(7) reduces to the MCF. Because MCF is a problem with no integrality constraints, the optimal value of the MCF problem is equal to the optimal value of its dual. The dual problem (DP) of the MCF problem can be written as

$$\text{(DP):}\quad \min \sum_{e\in E_v}\sum_{a\in\mathcal{A}}\sum_{\{C\in\mathcal{C}^a:\, e\in C\}} T^a \bar{x}_C \lambda_e + \sum_{(o,d,i)\in\Theta} D^{(o,d,i)}\omega^{(o,d,i)} \tag{13}$$

such that

$$\pi^{(o,d,i)}_{head(e)} - \pi^{(o,d,i)}_{tail(e)} + \lambda_e \geq -c^c_e \quad \forall e \in E - E_f,\ \forall (o,d,i)\in\Theta \tag{14}$$

$$\pi^{(o,d,i)}_{head(e)} - \pi^{(o,d,i)}_{tail(e)} + \omega^{(o,d,i)} \geq R^{(o,d,i)} - c^c_e \quad \forall e \in E_f,\ \forall (o,d,i)\in\Theta \tag{15}$$

$$\pi^{(o,d,i)}_v \text{ unrestricted}, \quad \forall v \in V,\ \forall (o,d,i)\in\Theta \tag{16}$$

$$\lambda_e \geq 0 \quad \forall e \in E_v \tag{17}$$

$$\omega^{(o,d,i)} \geq 0 \quad \forall (o,d,i)\in\Theta. \tag{18}$$

Let $D$ be the feasible region of the dual problem and $P_D$ and $Q_D$ be the set of extreme points and extreme rays of $D$, respectively. Note that $D$ does not depend on $\bar{x}_C$. Also, because $R^{(o,d,i)} \geq 0\ \forall (o,d,i)\in\Theta$ and $c^c_e = 0\ \forall e \in E_f$, a feasible solution for the dual subproblem is $\pi^{(o,d,i)}_v = 0\ \forall v \in V,\ \forall (o,d,i)\in\Theta,\ \lambda_e = 0\ \forall e \in E_v$, and $\omega^{(o,d,i)} = R^{(o,d,i)}\ \forall (o,d,i)\in\Theta$, and thus $D \neq \varnothing$. By strong duality, either the MCF problem is infeasible or it is feasible and bounded. Clearly, the null vector 0 is a feasible solution for MCF. This means that the primal-dual pair of MCF and DP is feasible and bounded. Thus, the optimal value of MCF and DP can be characterized in terms of only the extreme points of DP; that is, the set $P_D$, and can be written as

$$\min_{(\pi,\lambda,\omega)\in P_D} \sum_{e\in E_v}\left\{\sum_{a\in\mathcal{A}}\sum_{\{C\in\mathcal{C}^a:\, e\in C\}} T^a \bar{x}_C\right\}\lambda_e + \sum_{(o,d,i)\in\Theta} D^{(o,d,i)}\omega^{(o,d,i)}.$$

Introducing an additional free variable $z$, model (1)–(7) can be reformulated as the following *Benders master problem* (BMP). This problem has integer variables $x_C$ and one free continuous variable $z$:

$$\text{(BMP):}\quad \max z \tag{19}$$

such that

$$z \leq \sum_{e\in E_v}\left\{\sum_{a\in\mathcal{A}}\sum_{\{C\in\mathcal{C}^a:\, e\in C\}} T^a x_C\right\}\lambda_e + \sum_{(o,d,i)\in\Theta} D^{(o,d,i)}\omega^{(o,d,i)} - \sum_{a\in\mathcal{A}}\sum_{C\in\mathcal{C}^a} Cost_C x_C \quad \forall (\lambda,\omega)\in P_D \tag{20}$$

$$\sum_{C\in\mathcal{C}^a} L_C x_C \leq N^a \quad \forall a \in \mathcal{A} \tag{21}$$

$$x_C \in \{0,1\} \tag{22}$$

$$z \text{ free}. \tag{23}$$

Note that we do not have any feasibility constraints in the Benders master problem because (DP) is bounded. The optimality constraints (20) ensure that $z$ is restricted to be smaller than or equal to the value of the right-hand side of constraint (20) at various extreme points of DP. In general, the above model contains many more constraints than the LP relaxation of model (1)–(7), but most of them are inactive at optimality. Thus, a natural approach to solve (19)–(23) is by dropping constraints (20) and generating them as needed. We now present the basic Benders algorithm to solve the linear relaxation of SSSCR problem to optimality. Later, integrality constraints are introduced to solve the original SSSCR problem. We denote the linear relaxation of BMP as LPBMP and the relaxation of LPBMP obtained by dropping constraints (20) as the RLPBMP.

**2.3.2. Overview of the Algorithm.** The basic Benders decomposition-based algorithm for solving the LP relaxation of SSSCR iteratively selects good cycles, by solving the RLPBMP for the ship-scheduling problem, and then efficiently solves the cargo-routing problem by solving the MCF problem. The MCF problem utilizes the RLPBMP solution to assign capacity to voyage edges before solving the flow problem. In return, at each iteration, the dual solution of the MCF problem provides an optimality cut to the RLPBMP.

Let $t$ be the iteration number and $P_D^t$ be the restricted set of extreme points of $D$ available at iteration $t$; that is, the RLPBMP at iteration $t$ is obtained from LPBMP by replacing $P_D$ with $P_D^t$ in (20). Note that the solution of the RLPBMP at each iteration $t$, denoted by $z^t$, provides an upper bound for the original LPBMP (because the RLPBMP has fewer constraints than the LPBMP).

Algorithm 1. The basic Benders decomposition-based algorithm:

Procedure Basic Benders( )
  Set $t=1$, $P_D^t=\varnothing$, *lower_bound* $=0$,
    *upper_bound* $=\infty$.
  **while** (*upper_bound* $>$ *lower_bound* $+\epsilon$) **do**
    *Step* 1. *SOLVE* the RLPBMP to obtain solution $z^t$ and $\{\bar{x}_C\}^t$.
    Set *upper_bound* $=z^t$.
    *Step* 2. Solve the MCF problem taking $\{\bar{x}_C\}^t$ as input to obtain $v(\{\bar{x}_C\}^t)$ and optimal dual solution $(\bar{\pi}, \bar{\lambda}, \bar{\omega})$.
    Set *lower_bound* $=\max\{$*lower_bound*, $v(\{\bar{x}_C\}^t) - \sum_{a\in\mathcal{A}}\sum_{C\in\mathcal{C}^a} Cost_C\bar{x}_C\}$.
    Set $P_D^{t+1}=\{P_D^t \cap \{(x,z)\colon z \le \sum_{e\in E_v}\{\sum_{a\in\mathcal{A}}\sum_{\{C\in\mathcal{C}^a: e\in C\}} T^a x_C\}\lambda_e + \sum_{(o,d,i)\in\Theta} D^{(o,d,i)}\omega^{(o,d,i)} - \sum_{a\in\mathcal{A}}\sum_{C\in\mathcal{C}^a} Cost_C x_C\}\}$.
    $t=t+1$.
  **end while**

The original BMP (or SSSCR) problem with integrality constraints is solved heuristically in two phases. In Phase I, all integrality constraints are relaxed and the LPBMP is solved to optimality by using basic Benders decomposition algorithm (Algorithm 1). Because the set of feasible cycles can be exponential, the RLPBMP in this phase is solved in a column generation setting. For Phase I, *SOLVE* in Algorithm 1 refers to this column generation.

Retaining all optimality cuts and cycles generated in the first phase, Phase II puts the integrality constraints back on the master problem. Algorithm 1 is started once more; however, in this phase the RLPBMP in Step 1 is replaced with the mixed-integer program BMP, together with the cuts and cycles generated in the first phase. Because the DP polytope is not affected by the integrality constraints, all optimality cuts generated in Phase I can be used to generate corresponding cuts for the mixed-integer program in Phase II. Additional optimality cuts are generated at each iteration. Note, however, that in Phase II, Algorithm 1 in Step 1 solves an integer problem at every iteration. Thus, in Phase II no new cycles are generated, and *SOLVE* simply refers to a branch-and-bound solution of the relaxed BMP. This two-phase approach for solving integer programs using Benders decomposition was originally proposed by McDaniel and Devine (1977); the intuition behind it is the hope that many of the necessary constraints for the master problem may be generated by solving a linear program in place of the more computationally expensive integer program.

The branch-and-bound tree in Phase II is searched by a depth first search, giving preference to the up ($x_C=1$) branch. As in §2.2, the variable that affects the solution quality the most is chosen as the branching variable. Note that solving the mixed-integer program in BMP is a computationally expensive step. Because any feasible integer solution can be used to generate an optimality cut, the mixed-integer program BMP does not need to be solved to optimality at every iteration. However, if the BMP is solved heuristically, the upper bound it provides during the Benders iterations could be much smaller than the true upper bound, which might lead to a premature termination of the algorithm.

In the worst case, the upper bound could become smaller than the lower bound. To avoid such a premature termination of the algorithm, the branch-and-bound search is terminated only when the solution quality obtained reaches an acceptable optimality gap (the gap between the best integer objective and the objective of the best node remaining). Searching the branch-and-bound tree for a solution with small optimality gap is likely to take large computation time but it is also likely to provide better solution quality by providing better bounds for the Benders iterations. Thus, a suitable optimality gap must be chosen to avoid the premature termination of the algorithm and to keep it computationally efficient.

**2.3.3. Column Generation for Solving the RLPBMP.** The master problem in the Benders decomposition (the RLPBMP in Algorithm 1) is solved in a column generation setting. The pricing subproblem in the column generation reduces to identifying negative cost cycles in an auxiliary network, $G^a=(V^a,E^a)$, for every ship of type $a$. As before, $G^a$ is constructed such that $V^a=V$ and $E^a=E_v^a\cup E_g^a$. We next present how we compute the costs on the vertices and the edges of network $G^a$.

Let $\Pi_{(\lambda,\omega)}$ and $\sigma^a$ denote the dual variables corresponding to constraints (20) and (21), respectively.

The reduced cost $\bar{c}_C$ of a cycle $C \in \mathscr{C}^a$ can now be written as

$$\bar{c}_C = 0 - \left( \sum_{(\lambda,\omega)} \left( Cost_C - \sum_{\{e: e \in C \cap E_v\}} T^a \lambda_e \right) \cdot \Pi_{(\lambda,\omega)} + L_C \sigma^a \right). \quad (24)$$

Note that in (24) the second summation is only over the voyage edges of cycle $C$. Since the ground edges have infinite capacity, at optimality, by complementary slackness conditions $\lambda_e = 0 \ \forall e \in E_g$. Thus, ground edges can also be included in the summation in (24). From LP theory, we know that if the reduced cost $\bar{c}_C \le 0$ for each cycle $C \in \mathscr{C}^a$ and every fleet type $a \in \mathscr{A}$, then we have the optimal solution to our problem. That is, the column generation iterates as long as there exists a cycle $C \in \mathscr{C}^a$ for some $a \in \mathscr{A}$ such that

$$\sum_{(\lambda,\omega)} \left( \sum_{v \in C} c_v^{s,a} + \sum_{e \in C} c_e^{s,a} - \sum_{e \in C} T^a \lambda_e \right) \Pi_{(\lambda,\omega)} + \left\lceil \sum_{e \in C} \frac{l_e^a}{7} \right\rceil \sigma^a < 0$$

$$\Rightarrow \sum_{v \in C} \left( \sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)} \right) c_v^{s,a} - \sum_{e \in C} \left( \sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)} * \lambda_e \right) T^a + \sum_{e \in C} \left( \sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)} \right) c_e^{s,a} + \left\lceil \sum_{e \in C} \frac{l_e^a}{7} \right\rceil \sigma^a < 0. \quad (25)$$

For the network $G^a$, we assign cost

$$c_v = \left( \sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)} \right) c_v^{s,a}$$

to every $v \in V^a$ and cost $c_e = (l_e/7)\sigma^a - (\sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)} * \lambda_e) T^a + (\sum_{(\lambda,\omega)} \Pi_{(\lambda,\omega)}) c_e^{s,a}$ to every edge $e \in E^a$. Let $Cost_C$ represent the cost of cycle $C$ with the above cost structure. Let $\lceil Cost_C \rceil$ be the cost when we replace the $\sum_{e \in C} l_e^a/7$ term in $Cost_C$ with $\lceil \sum_{e \in C} l_e^a/7 \rceil$. Note that, since $\lceil x \rceil \ge x$, if the optimal value of the pricing subproblem $\forall a \in \mathscr{A}$ is greater than zero, then there are no more profitable cycles because $Cost_C \ge 0 \Rightarrow \lceil Cost_C \rceil \ge 0$. If however, the optimal value of the pricing subproblem for some $a \in \mathscr{A}$ is less than zero, then we need to check if $\lceil Cost_C \rceil < 0$. If it is, then we have found a profitable cycle; otherwise, either there are no more profitable cycles to be added or profitable cycles have very low negative cost ($> -1$) and are therefore ignored. We use Procedure *FindCycle*($G^a$) (as will be described in §3) to identify negative cost cycles in $G^a$.

## 3. Algorithmic Issues

In this section we discuss several algorithmic ideas we use to make our algorithms more effective, efficient, and stable.

### 3.1. Solving the Pricing Subproblem

The pricing subproblems for both the column generation and Benders decomposition-based algorithms reduce to finding profitable cycles in the auxiliary network $G^a$. Similarly, the greedy algorithm needs to find profitable cycles in the auxiliary network. It is tempting to directly solve a minimum cost circulation problem in the network $G^a$, to identify negative cost cycles. However, the cycles obtained by decomposing the solution of the circulation problem into simple cycles are not guaranteed to be practical. For example, our initial computational experiments with the circulation problem suggest that most of the cycles generated this way are too long and require a large number of ships to maintain weekly frequency. Hence, we first discuss rules based on the real-world practice of liner shipping companies to define feasible cycles. Next, a recursive algorithm, *FindCycle*($G$), is presented to find negative cost cycles efficiently, satisfying predefined feasibility conditions in a given network $G$.

**3.1.1. Defining Feasible Cycles.** To solve the pricing subproblem to optimality one must consider all sequences of ports as candidates for possible profitable cycles. However, searching for negative cost cycles in such an unconstrained manner not only makes our algorithms inefficient by generating cycles that create undesirable effects such as large integrality gaps, but also it generates cycles that would never be operated in practice. Hence we impose a set of constraints that a cycle must satisfy to qualify as a feasible cycle.

Global carriers operate in different regions—for example, Orient Overseas Container Line (OOCL) operates mainly in North America, Europe, and Asia—and cater to the demand of various markets, such as trans-Atlantic, trans-Pacific, intra-Asia, and Asia-Europe trade routes. Figure 5 represents an Asia-Europe cycle for OOCL. For a carrier it is important to tap the benefits of both interregion and intraregion markets. Whereas some of the interregion markets, for example, the trans-Pacific, are the most profitable ones, some intraregion markets, for example, the intra-Asia market, form the backbone of international shipping.

We considered the cycles published by OOCL (2005) and APL (2005) in trans-Pacific and intra-Asia trade routes to come up with the following guidelines for defining the set of feasible cycles distributed in two regions, $r_i$ and $r_j$.

1. The number of ports visited by a cycle must not be too high. Most of the current trans-Pacific cycles visit 10–15 ports, and intra-Asia cycles visit 7–10 ports. Let $R_{(r_i, r_j)}$ denote the maximum number of ports that a cycle visiting region $r_i$ and $r_j$ is allowed to visit.


**Figure 5 An Asia-Europe Cycle for OOCL**
*Source.* OOCL.

2. The length (in weeks) of a cycle must be bounded by a suitable number; that is, the number of ships that can be committed to a particular cycle is limited. Most of the trans-Pacific cycles are up to 15 weeks long, and most of the intra-Asia cycles are up to six weeks long. Let $L_{(r_i, r_j)}$ be the maximum allowed length in weeks for a cycle visiting region $r_i$ and $r_j$.

3. Cycles that operate in multiple regions must enter and leave a region only once; that is, no interregion loops are allowed. However, 1–2 intraregion loops are allowed.

4. Each cycle must directly (without using capacity on other cycles) serve the origin and destination ports of at least one demand triplet. It is highly unlikely for a carrier to introduce a cycle that does not satisfy any demand directly.

**3.1.2. Finding Negative Cost Feasible Cycles.** Incorporating any of the rules that guide the feasibility of a cycle into the circulation problem yields an NP-hard problem. (This is easily seen from the fact that shortest weight-constrained path problem is NP complete (Garey and Johnson 1979).) Furthermore, an exhaustive enumeration of cycles following the above rules still yields a large number of cycles. For ports distributed in two regions, up to 10,000 cycles for a 10-port, 30-demand-triplets problem, more than 1 million cycles for a 15-port, 50-demand-triplets problem, and more than 10 million cycles for a 20-port, 80-demand-triplets problem exist.

We now describe an iterative search algorithm for constrained negative cycle detection that yields good computational results. In essence, the algorithm utilizes Lemma (1) (Lin and Kernighan 1973) to prune the search tree by ignoring paths with nonnegative costs. This pruning helps the algorithm to maintain time- and space-efficiency. Ahuja, Orlin, and Sharma (2003) have used Lemma (1) to develop a similar algorithm for detecting subset disjoint negative cycles.

Lemma 1. *For a negative cost (directed) cycle* $C = v_1 - v_2 - \cdots - v_r - v_1$ *there exists a node* $v_h$ *in* $C$ *such that each partial (directed) path* $v_h - v_{h+1}$, $v_h - v_{h+1} - v_{h+2}$, $v_h - v_{h+1} - v_{h+2} - \cdots$ *(where indices are modulo* $r$*) is a negative cost (directed) path.*

We now present a cycle generation algorithm for ports distributed in two regions: $r_1$ and $r_2$. Note that these ideas can easily be carried over to ports distributed in more than two regions. Before presenting the algorithm we define some notations. With each directed path $p$, we associate the following information: $head(p)$ and $tail(p)$ denote the last node and first node on $p$, respectively. $Cost(p)$ denotes the cost of path $p$. $NR_{r_1}(p)$ and $NR_{r_2}(p)$ denote the number of ports from region $r_1$ and $r_2$, $ER(p)$ denotes the number of interregion edges, and $l(p)$ denotes the length of path $p$. Note that each edge in the network is either between two nodes of the same region (intraregion edge) or between two nodes of different regions (interregion edge) and that the set $\{NR_{r_1}(p), NR_{r_2}(p), ER(p)\}$ completely describes the region(s) visited by a path $p$. For a path $p$ we denote the set $\{head(p), tail(p), ER(p)\}$ as $DSet(p)$. We say that a path $p$ *dominates* another path $q$ if $DSet(p) = DSet(q)$ and $Cost(p) < Cost(q)$.

A cycle can be obtained by connecting the endpoints of a path. Lemma (1) suggests that to find negative cost cycles it is enough to consider paths with negative cost. Further, the above definition of dominance suggests that among the paths with the same $DSet()$ only the path with the least cost needs to be explored further. A path $p$ is said to be feasible if it has negative cost and if it can be extended to form a feasible cycle. Let $P_k$ denote the set of all nondominated, feasible paths with $k$ nodes.

For each ship type $a$, Algorithm (2) detects negative cost cycles in the auxiliary network $G^a$, described earlier with various cost structures. It works inductively by constructing set $P_{k+1}$ from the set $P_k$. For each

path $p \in P_k$ it examines if the path can be extended by adding a single edge to form path $p'$, that is, if path $p'$ is feasible. Procedure *if_feasible_path*($p$) checks for the feasibility of path $p$ depending on the region(s) visited by $p$, by ensuring that the guidelines set in §3.1.1 are met, and accounts for the fact that for a ship type $a$ no cycle can be longer than $N^a$ (number of available ships for ship type $a$) weeks. The path is then checked for dominance in $P_{k+1}$ using procedure *if_dominated*($p', P_{k+1}$), and nondominated paths are added to $P_{k+1}$. For a cycle $C$, procedure *if_feasible_cycle*($C$) checks whether the cycle $C$ is feasible.

For a path $p$ all information can be maintained in $O(1)$ time. For example, to maintain $ER(p)$, we assign a value 0 to all intraregion edges and value 1 to interregion edges. Thus, whenever an edge is appended to a path $p \in P_k$ to obtain path $p' \in P_{k+1}$, $ER(p')$ can be obtained by adding the value of the appended edge to $ER(p)$. Because we maintain all information regarding the region(s) visited by a path, the feasibility check for a path and a cycle can be done in constant time. To check the dominance of a path $p \in P_k$, we first need to check if there exists a path $q \in P_k$ such that $DSet(p) = DSet(q)$. This is a computationally expensive step. We use standard hashing techniques to efficiently detect paths with the same $DSet$. Once such a path is found, dominance can be checked in $O(1)$ time. Note that at any given time, set $P_k$ will contain only one path with a particular $DSet$, that is, the nondominated path. Rather than storing only the most negative cycle $C^*$, Algorithm (2) can easily be modified to maintain a predefined number of best cycles.

### 3.2. Choosing an Initial Set of Cuts

Even though the Benders decomposition-based Algorithm (1) may be initialized with an empty set of extreme points, the choice of an initial set may affect its convergence. In our experiments, the addition of several cuts helped us improve the performance of Algorithm (1).

Note that $\pi_v^{(o,d,i)} = 0 \ \forall v \in V, \forall (o,d,i) \in \Theta$, $\lambda_e = 0 \ \forall e \in E$, and $\omega^{(o,d,i)} = R^{(o,d,i)} \ \forall (o,d,i) \in \Theta$ is a feasible but not necessarily an extreme point solution of the DP polytope. It can thus be used to obtain the valid cut

$$z \le \sum_{(o,d,i)} R^{(o,d,i)} D^{(o,d,i)} - \sum_{a \in \mathcal{A}} \sum_{C \in \mathcal{C}^a} Cost_C x_C. \qquad (26)$$

The above cut is equivalent to adding the constraint that the value of the optimal solution must be less than or equal to the revenue that can be generated by satisfying all the available demand minus the cost of operating the picked cycles.

Similarly,

$$\lambda_e = \max_{(o,d,i) \in \Theta} R^{(o,d,i)} \ \forall e \in E$$

and $\omega_{(o,d,i)} = 0 \ \forall (o,d,i) \in \Theta$ is a feasible solution for the DP polytope and provides the valid cut

$$z \le \max_{(o,d,i) \in \Theta} R^{(o,d,i)} \sum_{e \in E_v} \left\{ \sum_{a \in \mathcal{A}} \sum_{C \in \mathcal{C}^a : e \in C} T^a x_C \right\} - \sum_{a \in \mathcal{A}} \sum_{C \in \mathcal{C}^a} Cost_C x_C. \qquad (27)$$

Algorithm 2. An iterative constrained negative cycle detection algorithm:

```
Procedure FindCycles(G^a)
  for all e ∈ E^a do
    p = {e}
    if if_feasible_path(p) then P_1 = P_1 ∪ {p}
  end for
  k = 1, C* = ∅, Cost_C* = 0
  while k < R do
    while P_k ≠ ∅ do
      Remove a path p from P_k
      Connect the ends of path p to form cycle C.
      if if_feasible_cycle(C) and Cost_C < Cost_C*
        then C* = C
      for all {(head(p), j) ∈ OutEdges(head(p))} do
        p' = p ∪ {(head(p), j)}
        if {if_feasible_path(p')} then
          P_{k+1} = P_{k+1} ∪ {p'}
          if if_dominated(p', P_{k+1}) then Remove the
            dominated path
        end if
      end for
    end while
    k + 1
  end while
```

### 3.3. Making Column Generation Effective

While performing column generation, both in the pure column generation-based algorithm for SSSCR and for solving the RLPBMP in a Benders decomposition-based algorithm, we identify and add more than one profitable column per iteration. During the iterative cycle generation, instead of maintaining just the most negative cycle, we maintain a set of 5–10 most profitable cycles at almost no extra cost. This helps significantly reduce the number of iterations during column generation without substantially increasing the time taken per iteration.

During a typical column generation, the problem keeps growing as the column generation process keeps adding columns to the master problem. To keep the list of columns manageable, we frequently delete nonbasic columns with high negative reduced cost from the master problem. This reduces the time per iteration significantly, although it increases the number of iterations slightly in many cases. As another way to speed up the column generation process, if no new cycle is detected for a ship type in an iteration,

then cycle generation for that ship type is suspended for two to five iterations.

## 4. Computational Experiments

In this section, we present the results of our computational study after describing the schema employed for generating test cases. We first establish the dominance of the Benders decomposition-based algorithm over the greedy heuristic and the column generation-based algorithm. Next, we present a deeper analysis of the Benders decomposition-based algorithm. Finally, we discuss some of the interesting characteristics of the solutions obtained by our algorithm and show that it supports the recent trends observed in the sea cargo industry. All our algorithms were implemented in C++ in an Unix environment, and we made extensive use of the callable libraries in CPLEX 9.0. All computational experiments were performed on a Sun280R workstation with an UltraSparc-III processor. All times are reported in minutes.

### 4.1. Data Generation

We performed our computational experiments on networks with ports distributed in two regions. Each generated port is randomly assigned to one of the two regions, with equal probability, and the sailing distance between ports is chosen to represent the sailing distance between ports distributed in the Asian and North American regions. Typically, as observed from OOCL (2005) and APL (2005) service networks, intraregion sailing times for ports in Asia and North America are 2–30 days, whereas the interregion sailing times are 14–42 days.

Origin-destination pairs are chosen randomly from the pairs of ports. Day of the week on which supply arises at the origin port is assumed to be the same every week and is chosen uniformly at random from the seven days of a week. The demand sizes are randomly generated from the interval 0.1 to 1.0 times the capacity of the largest ship available. Similar proportions are used in Fagerholt (1999), and it is suggested that this represents the demand sizes observed by a liner shipping company. Revenue generated by satisfying demand for a given demand triplet $(o, d, i)$ is chosen to be in direct proportion to the distance between port $o$ and port $d$; that is, more revenue is generated by satisfying a demand at a port in North America from a port in Asia, as compared with satisfying a demand between two ports in North America. The proportionality constant is chosen randomly from $[100, 200]$.

Because a carrier's fleet usually consists of ships of different types, we considered three different ship types in our fleet. The three ship types have capacity 2,000 TEU, 4,000 TEU, and 8,000 TEU. Bendall and Stent (1999) and Imai et al. (2006) suggest that ships with 2,000 TEU and 4,000 TEU capacity are currently in use. According to OOCL (2005), OOCL has ships of different types with capacity varying from 2,500 TEU to 8,063 TEU. A recent increase in literature regarding the viability of larger ships (Imai et al. 2006) points toward the increasing use of big ships, and Bendall and Stent (1999) suggest that ships of up to 8,000+ TEUs are being designed.

There are various fixed and variable costs involved in shipping cargo. As in Christiansen and Nygreen (1998), we do not consider the daily running costs, including cost of capital, personnel, insurance, etc. because they are fixed during the planning period. However, we consider various operational costs that affect a carrier's decision regarding which ports to visit and which cycles to operate on. For every ship type, $a \in \mathcal{A}$, and for all the ports, $v \in \mathcal{P}$, we consider a port visit cost incurred by a ship of type $a$ if it visits port $v$. At port $p$, port visit cost for a ship is proportional to the capacity of the ship; that is, a ship with 8,000 TEU capacity incurs a higher port visit cost compared with a 2,000 TEU capacity ship.

At every port, $v \in \mathcal{P}$, we consider a per unit cargo per night holding cost. This cost is incurred by a unit of cargo if it is held at a port for one night and is assumed to be the same for all cargo types. At a port, holding cost per unit of cargo is chosen to be considerably smaller than the port visit cost for a ship. For every ship type, $a \in \mathcal{A}$, and for every pair of ports, $\{u, v\}$, we consider the operation cost for sailing a ship from port $u$ to $v$. The operation cost depends on the type of ship used for the sailing and is proportional to the distance between the ports.

We generate various classes of random instances, utilizing the above schema, to test the robustness of our algorithm. Classes are characterized by specifying the number of ports ($P$), the number of ships ($S$), and the number of demand triplets ($D$). For example, an instance with six ports, 30 ships, and 18 demand triplets is represented as P6S30D18. We tested our algorithm on networks with 6, 10, 15, and 20 ports to be serviced. In each of the test classes, 20%–30% of all pairs of ports are considered to be origin-destination pairs. A fleet size of up to 100 ships is scheduled. Grand Alliance, which is one of world's largest alliances, has a fleet of 100 ships, and APL (2005) has a fleet of more than 80 container ships. For each test class, results reported in this section were obtained by generating five random instances and then taking an average over them.

To report the results of our computational study in tabular form we use the following abbreviations:

*G*: The greedy algorithm.
*C*: The pure column generation-based algorithm.
*B*: The two phase Benders decomposition-based algorithm, where column generation is used for solving the master problem in Phase I.

*F*: The cycle generation algorithm based on the flow decomposition of the circulation problem.

*I*: The cycle generation algorithm based on the iterative search algorithm.

Combinations of these are used to represent the overall algorithm tested. For example, the two-phase Benders decomposition-based algorithm with the iterative search algorithm for cycle generation is represented by *BI*.

### 4.2. Effectiveness of the Algorithms

We now compare the Benders decomposition-based algorithm with the other proposed algorithms. While solving the problem with the pure column generation-based algorithm, the LP relaxation is first solved to optimality and then the integer solution is obtained using branch-and-bound. However, while solving the problem with the two-phase Benders decomposition-based algorithm, in Phase I the cuts are generated until the relative difference between the upper bound provided by the Benders relaxed master problem and the lower bound provided by the subproblem is less than 1% or the number of iterations in the first phase of Benders are less than 200. Phase I terminates when one of these criteria is met. The LP solution obtained by the pure column generation-based algorithm is used as an upper bound to estimate the quality of the final integer solution.

Table 1 presents a comparison among the greedy algorithm, the pure column generation-based algorithm, and the Benders decomposition-based algorithm. It also compares the flow decomposition-based cycle generation algorithm with the iterative search algorithm for cycle generation. The second and third column of Table 1 report the number of cycles generated and the CPU time taken to solve the problem using greedy algorithm with iterative cycle generation. The fourth and fifth column report these statistics for the pure column generation-based algorithm with iterative cycle generation. The next four columns, two each, report the corresponding statistics for the Benders decomposition-based algorithms with algorithm *F* and algorithm *I* for cycle generation, respectively. The last three columns report the gap corresponding to the relative difference between the solution value of the *GI* and the *BI* algorithm, the *CI* and the *BI* algorithm, and the *BF* and the *BI* algorithm, respectively. Initial cuts described in §3.2 are used in both of the Benders decomposition-based algorithms. Also, columns with reduced cost of *less than 1,000,000* are removed after every 10 iterations, during the column generation phase in Algorithm *C* and while solving the master problem in Phase I of Algorithm *B*. As discussed at the end of §2.3.2, care must be taken in setting the stopping conditions for the mixed-integer program BMP in Phase II of the Benders decomposition-based algorithm. In our computational experiments, stopping the MIP when a 1% optimality gap for small instances (6–10 ports) and a 3%–5% gap for large instances (15–20 ports) is reached provided a good balance of computational time and solution quality. These parameters were set after initial computational experiments. Specifically, for the six port instances when the optimality gap is reduced from 1% to 0.1%, the solution quality improves by only ∼0.04%, whereas the time taken to solve the integer program increases by ∼55%. Hence, we believe that heuristically solving the MIPs did not have a significant effect in prematurely terminating the Benders algorithm if the optimality gap was chosen properly. Also, in our computations when we use the above optimality gaps as stopping criteria, we never ran into a situation where the upper bound obtained by the MIP was less than the Benders lower bound. For the *CI*, *BF*, and *BI* algorithms, figures in the column No. of Cycles report the number of cycles in the integer program. Note that in these algorithms a larger number of cycles are generated during column generation, while solving the LP, but subsequently removed if they have high negative reduced cost.

The results of our tests show that there is a very significant difference in the solution quality obtained by the greedy algorithm and the solution quality obtained by the other two algorithms. Although the

**Table 1 Comparison Between Different Algorithms**

| | *GI* | | *CI* | | *BF* | | *BI* | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Test class | No. of cycles | Time | No. of cycles | Time | No. of cycles | Time | No. of cycles | Time | Percent of GI − BI | Percent of CI − BI | Percent of BF − BI |
| P6S18D6 | 2 | 0.03 | 62 | 0.95 | 234 | 0.68 | 49 | 0.16 | 49.27 | 0.01 | 3.92 |
| P6S18D9 | 3 | 0.05 | 70 | 2.10 | 272 | 2.59 | 64 | 0.30 | 57.84 | 0.60 | 2.90 |
| P6S30D6 | 3 | 0.09 | 181 | 7.26 | 215 | 1.30 | 96 | 0.33 | 37.28 | 1.10 | 2.81 |
| P6S30D9 | 4 | 0.13 | 239 | 34.42 | 366 | 1.73 | 120 | 0.57 | 33.01 | −0.21 | 1.99 |
| P10S30D18 | 5 | 0.76 | 312 | 760.20 | 1,926 | 67.97 | 213 | 15.41 | 60.29 | −0.28 | 3.50 |
| P10S30D27 | 5 | 1.65 | 358 | 1,312.00 | 2,355 | 160.83 | 292 | 30.56 | 67.00 | 2.21 | 3.21 |
| P10S50D18 | 7 | 1.78 | 607 | 2,077.10 | 3,102 | 583.82 | 371 | 61.34 | 47.39 | 2.10 | 5.25 |
| P10S50D27 | 8 | 2.64 | 790 | 2,910.00 | 4,587 | 1,318.68 | 578 | 116.35 | 45.01 | 0.02 | 5.65 |

greedy heuristic is fast, it works with a very small set of cycles and picks each cycle that it generates without any further considerations. The pure column generation-based algorithm yields solution qualities comparable to the Benders decomposition-based algorithm with iterative cycle generation; however, it incurs a longer computational time, and this difference increases as the problem size increases. Although the number of cycles passed on to the integer program in the pure column generation is not very high, compared with the number of cycles at the end of Phase I in algorithm *BI*, the amount of time taken is much greater. This can be contributed to the fact that as the problem size (number of ports, ships and demand triplets) increases, the number of variables as well as the number of constraints increases in the column generation-based algorithm. However, in the Benders decomposition-based algorithm, the effect of increase in problem size is distributed between the master problem and the subproblem.

Although the *BI* algorithm outperforms the *BF* algorithm uniformly, the difference between the solution quality obtained by these algorithms is less than 6%. However, the time taken in the *BF* algorithm is four to five times higher than the time taken by the *BI* algorithm. This can be attributed to the fact that, in algorithm *BF*, many infeasible cycles are generated by solving the circulation problem and decomposing its flow in the first phase of the Benders decomposition based algorithm. For a 6- (10) port problem, *BF* generated about 65% (60%) infeasible cycles in Phase I. Although more cycles are submitted at the end of Phase I by algorithm *BF*, the branch-and-bound takes far less time compared with the corresponding branch-and-bound in algorithm *CI* because most of the cycles generated by algorithm *BF* are infeasible for the integer program and are removed at the start of the branch-and-bound. Moreover, in the *CI* algorithm most of the time is spent solving the LP relaxation via column generation.

Table 1 reports results for test cases with up to 10 ports because the pure column generation-based algorithm and the flow decomposition-based cycle generation algorithm become computationally very expensive, making *CI* and *BF* ineffective. Also, the solution quality of the greedy algorithm decreases further, compared with the Benders decomposition based algorithm. Table 1 establishes the superiority of the solution, in terms of both CPU time and revenue generated, obtained by the two-phase Benders decomposition-based algorithm with iterative cycle generation. Thus, we used this algorithm to perform all further experiments.

### 4.3. Analysis of the Benders Decomposition-Based Algorithm

Our next set of experiments performs a deeper analysis of the Benders decomposition-based algorithm; results are presented in Table 2. In these experiments we used initial cuts and removed columns with large negative reduced costs after every 10 iterations in the first phase of the Benders decomposition-based algorithm. The second column in Table 2 represents the number of iterations in the first phase of the algorithm. The third, fourth, and fifth columns present a breakdown of the total time taken in various processes while solving the LPBMP. The next column represents the additional time taken to obtain an integer solution. The last column reports the gap corresponding to the relative difference between the upper bound, obtained by the *CI* algorithm, and the integer solution value obtained by the *BI* algorithm. To keep

**Table 2 Analysis of the Benders Decomposition-Based Algorithm**

| | | Phase I | | | | |
|---|---|---|---|---|---|---|
| Test class | Iters | Subproblem | Master | Cycle-gen. | Phase II | Percent of gaps |
| P6S18D6 | 13 | 0.02 | 0.11 | 0.09 | 0.01 | 10.24 |
| P6S18D9 | 16 | 0.10 | 0.19 | 0.14 | 0.03 | 12.10 |
| P6S30D6 | 20 | 0.06 | 0.23 | 0.17 | 0.03 | 2.30 |
| P6S30D9 | 27 | 0.16 | 0.36 | 0.27 | 0.05 | 3.32 |
| P10S30D18 | 47 | 7.24 | 6.13 | 5.17 | 1.99 | 8.55 |
| P10S30D27 | 56 | 17.02 | 7.65 | 3.63 | 6.54 | 9.80 |
| P10S50D18 | 75 | 23.87 | 20.64 | 16.09 | 19.65 | 1.91 |
| P10S50D27 | 95 | 52.69 | 31.20 | 18.37 | 35.55 | 3.24 |
| P15S45D42 | 130 | 105.86 | 69.46 | 52.27 | 35.25 | 8.63 |
| P15S45D63 | 175 | 141.60 | 110.69 | 72.00 | 33.80 | 8.53 |
| P15S75D42 | 181 | 172.25 | 152.49 | 118.80 | 167.72 | 5.30 |
| P15S75D63 | 200 | 254.26 | 212.56 | 156.08 | 174.78 | 5.92 |
| P20S60D76 | 200 | 1,165.87 | 73.12 | 39.92 | 42.07 | 12.70 |
| P20S60D114 | 200 | 1,750.63 | 113.18 | 47.38 | 173.37 | 7.51 |
| P20S100D76 | 200 | 2,507.51 | 164.61 | 72.81 | 262.45 | 5.05 |
| P20S100D114 | 200 | 3,784.38 | 380.11 | 149.65 | 478.01 | 7.21 |

**Table 3 Effect of Algorithmic Refinements**

| | No cuts+all cols. | | | Cuts+all cols. | | | Cuts+remove cols. | | |
|---|---|---|---|---|---|---|---|---|---|
| Test class | No. of cycles | Iters | Time | No. of cycles | Iters | Time | No. of cycles | Iters | Time |
| P6S18D6 | 62 | 15 | 0.20 | 51 | 13 | 0.17 | 49 | 13 | 0.16 |
| P6S18D9 | 93 | 15 | 0.34 | 87 | 15 | 0.33 | 64 | 16 | 0.30 |
| P6S30D6 | 194 | 22 | 0.66 | 105 | 20 | 0.44 | 96 | 20 | 0.33 |
| P6S30D9 | 240 | 30 | 1.03 | 131 | 27 | 0.88 | 120 | 27 | 0.57 |
| P10S30D18 | 494 | 52 | 18.82 | 464 | 45 | 18.68 | 213 | 47 | 15.41 |
| P10S30D27 | 673 | 60 | 64.50 | 603 | 55 | 50.45 | 292 | 56 | 30.56 |
| P10S50D18 | 882 | 79 | 271.03 | 790 | 71 | 168.82 | 371 | 75 | 61.34 |
| P10S50D27 | 1,102 | 99 | 748.23 | 889 | 91 | 364.81 | 578 | 95 | 116.35 |

computational time under control, in Phase II, only two to three iterations of the Benders algorithm were performed.

Table 2 suggests that as the number of demand triplets increases, the time taken to solve the subproblem increases. This is mainly because every demand triplet is considered a different commodity; thus, as the number of demand triplets increases, the complexity of the multicommodity flow problem or the subproblem increases (in the number of variables and constraints) significantly. Note that an increase in the number of demand triplets results in an increase in the time taken to solve the master problem also. This is because of the increased possibilities with regard to the cycles that can be generated. The overall time increases as we increase the number of ports, the number of ships, or the number of demand triplets.

For the same number of ports, as the number of ships increase, the integrality gap reduces significantly. This suggests that the set of cycles generated in the first phase is good for the second phase also, and given a sufficient number of ships, the gap can be reduced further. For small test cases with six ports, we observed that the integer solution obtained by our algorithm is indeed close to the optimal solution in many cases and that the LP-based upper bound is not very tight. It is easily seen that the integrality gap can be very bad. Consider a two-port, one-ship instance such that the sailing time between ports is one week. An LP solution will assign half a ship to each edge, whereas an integer solution will yield zero revenue, resulting in a 100% integrality gap. However, given a sufficient number of ships, such extreme cases are highly unlikely to occur.

Our next set of experiments studies the effect of using the refinements described in §§3.2 and 3.3. Using the two-phase approach we solve each instance first without the initial set of cuts, then without removing any column at intermediate steps, and finally by incorporating the initial cuts and removing columns at intermediate steps, to keep only a subset of columns. Parameters are chosen so that the solution quality is not affected by these refinements; however, the computational time is reduced significantly. Table 3 reports cycles generated, iterations performed, and the time taken for each of these cases. The total CPU time taken to find an integer solution is also reported.

Table 3 reports results for networks with up to 10 ports because the time taken in both phases of the Benders decomposition-based algorithm becomes prohibitively high for networks with more than 10 ports if we remove the initial cuts or do not remove cycles with large negative reduced costs. Note that removing columns with a negative reduced cost of less than 1,000,000 does not reduce the number of cycles significantly for six port instances, because not many cycles for such a small network have a large negative reduced cost. However, the same refinement reduces the number of cycles for 10 port instances to approximately half the size, suggesting that this refinement must be tuned according to the problem size to properly control the number of columns in the linear program.

Table 3 suggests that the CPU time as well as the number of iterations in the first phase of the Benders decomposition-based algorithm are reduced by introducing the initial cuts. However, a more significant reduction in time is achieved by removing columns with large negative reduced cost. Removing very negative reduced cost cycles does not affect the time taken in Phase I very much, but the number of columns that the integer program works with in Phase II is reduced considerably; thus the time taken in the second phase of the Benders decomposition-based algorithm reduces significantly.

Finally, we study the effect on the solution quality of having only one ship type in the fleet. Table 4 reports results for a fleet of identical ships with a 4,000 TEU capacity. For each test class, we report the CPU time taken in Phases I and II of the Benders decomposition-based algorithm with iterative search for cycle generation, the total number of cycles generated, and the optimality gap. In this case also, two

**Table 4 Effect of Identical Ships in the Fleet**

| | Phase I | | | | | |
|---|---|---|---|---|---|---|
| Test class | Subproblem | Master | Cycle-gen. | Phase II | No. of cycles | Percent gap |
| P6S18D6 | 0.02 | 0.09 | 0.07 | 0.00 | 35 | 1.43 |
| P6S18D9 | 0.04 | 0.17 | 0.14 | 0.01 | 42 | 2.14 |
| P6S30D6 | 0.03 | 0.11 | 0.09 | 0.00 | 60 | 0.16 |
| P6S30D9 | 0.06 | 0.20 | 0.18 | 0.04 | 72 | 2.01 |
| P10S30D18 | 3.92 | 2.97 | 2.41 | 0.30 | 190 | 2.25 |
| P10S30D27 | 5.06 | 2.16 | 1.82 | 0.62 | 202 | 2.23 |
| P10S50D18 | 8.32 | 6.98 | 5.25 | 2.02 | 243 | 1.45 |
| P10S50D27 | 13.38 | 8.36 | 6.82 | 2.54 | 275 | 1.74 |
| P15S45D42 | 51.72 | 17.40 | 15.35 | 3.72 | 398 | 2.53 |
| P15S45D63 | 97.05 | 25.59 | 21.97 | 4.97 | 520 | 2.01 |
| P15S75D42 | 171.12 | 52.77 | 37.77 | 5.20 | 583 | 1.93 |
| P15S75D63 | 209.07 | 87.28 | 52.78 | 6.83 | 647 | 1.56 |
| P20S60D76 | 1,023.53 | 106.96 | 91.60 | 12.83 | 450 | 1.32 |
| P20S60D114 | 1,869.77 | 193.79 | 117.76 | 13.50 | 791 | 1.16 |
| P20S100D76 | 1,825.67 | 181.85 | 144.67 | 14.82 | 957 | 1.91 |
| P20S100D114 | 2,923.28 | 189.13 | 141.51 | 16.50 | 980 | 2.01 |

to three iterations of the Benders algorithm were performed in Phase II.

Table 4 suggests that if all the ships are identical, the optimality gap reduces even further in all the test classes. Because all ships are identical, in Phase II it becomes easier to operate a service route using ships of a similar kind to maintain the weekly frequency. Comparing Table 2 with Table 4 suggests that the overall time taken also reduces. The time taken in the cycle generation process reduces significantly, as now the cycle generation needs to be solved only for one ship type at every iteration. Thus, the time taken in the master problem decreases. Also note that fewer cycles are generated, and thus the time taken in Phase II reduces significantly. As a result, the overall solution time is reduced.

### 4.4. Analysis of the Solution

In this section, we take a closer look at the solution generated by the Benders decomposition-based algorithm and its implications. Also, we perform preliminary experiments to study the effect of transshipment cost on cargo routing.

The second column in Table 5 reports the number of cycles or service routes picked in the final solution. The number of service routes increases as the numbers of ships and ports increase. The next two columns in Table 5 report the average percentage utilization of capacity on the edges of the network and the percentage of the cargo that is transshipped. These results are for the case when we do not consider transshipment cost, that is, when the cost of transshipment is 0. Utilization of capacity on an edge is calculated by dividing the total flow on that edge by the total capacity of the edge. Recall that the capacity of an edge is defined by the number of ships (and their capacities) that cross the given edge. Across our problem instances, our algorithm consistently reports high average percentage utilization (70%–90%) of capacity. Note that higher the number of service routes, the higher the number of possibilities for cargo routes. As a result, the percentage of the cargo transshipped increases as the problem size increases. This trend is observed in our computational study also as the amount of transshipped cargo increases from ∼19% for a 6-port problem to ∼30% for a 10-port problem.

**Table 5 Analysis of the Obtained Solutions**

| | | Trans_cost = 0 | | Trans_cost = 20 | | Trans_cost = 100 | | Trans_cost = 1,000 | |
|---|---|---|---|---|---|---|---|---|---|
| Test class | No. of picked cycles | Percent of utilization | Percent of transshipped | Percent diff. demand | Percent of transshipped | Percent diff. demand | Percent of transshipped | Percent of demand | Percent of transshipped |
| P6S18D6 | 3 | 0.58 | 17.28 | 0.00 | 12.84 | 0 | 12.84 | 30.99 | 15.17 |
| P6S18D9 | 3 | 0.82 | 21.26 | 0.00 | 17.12 | 0.65 | 16.86 | 28.96 | 6.41 |
| P6S30D6 | 4 | 0.71 | 20.03 | 0.00 | 13.77 | 0.27 | 13.48 | 53.54 | 0.01 |
| P6S30D9 | 5 | 0.79 | 17.33 | 0.00 | 14.47 | 1.59 | 13.48 | 33.79 | 6.58 |
| P10S30D18 | 6 | 0.83 | 22.30 | 0.00 | 15.63 | 0.66 | 14.99 | 23.00 | 3.71 |
| P10S30D27 | 6 | 0.86 | 28.07 | 0.00 | 23.08 | 0 | 23.08 | 13.94 | 11.96 |
| P10S50D18 | 7 | 0.88 | 34.09 | 0.25 | 29.46 | 1.08 | 28.81 | 31.79 | 14.42 |
| P10S50D27 | 8 | 0.91 | 32.53 | 0.26 | 27.84 | 0.74 | 27.44 | 30.44 | 11.41 |

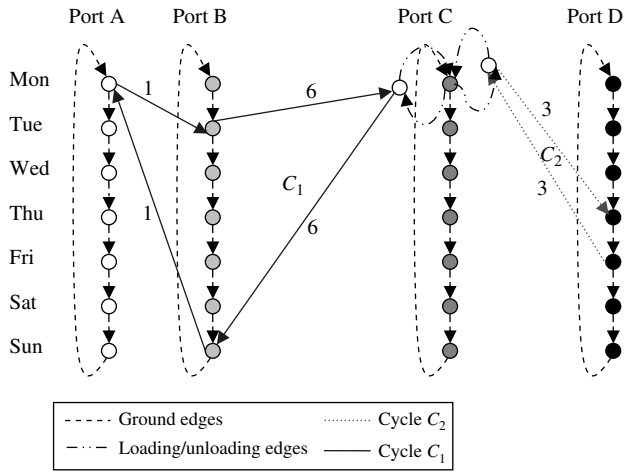


**Figure 6 New Network to Study the Effects of Transshipments**

Next, we perform preliminary experiments to study the effect of transshipment cost on cargo routing. Depending on the set of chosen service routes, we construct a new network. In the new network, at every port where two or more cycles meet a new node is constructed for every cycle. The new nodes are connected to the original port node via edges. These edges act as loading/unloading edges and have corresponding costs associated with them. For example, for the network represented in Figure 3, the new network is given by Figure 6. At port $p$, $c_p^u$ and $c_p^l$ denote the unloading and loading cost, respectively, and the transshipment cost is given by $c_p^u + c_p^l$. Thus, a transshipment occurs when at an intermediate port cargo travels on an unloading and then a loading edge. In Figure 6, cargo that is routed from port $B$ to port $D$ is transshipped at port $C$ and it uses the unloading edge from cycle $C_1$ to port $C$ and the loading edge from port $C$ to cycle $C_2$.

To perform the experiment, we construct the new network for the cycles selected at the end of the second phase of the Benders decomposition-based algorithm. The cargo-routing problem is solved for both the new and the original networks. The effect of the transshipment costs on the cargo-routing decisions is studied by observing the percentage difference between the demand satisfied in the original network (in the absence of transshipment costs) and the new network (in the presence of transshipment costs). We also compute the percentage of cargo transshipped to the total cargo shipped. These two statistics are reported in Table 5 for three different scenarios: transshipment cost = 20 units per unit of cargo; transshipment cost = 100 units per unit of cargo; and transshipment cost = 1,000 units per unit of cargo. Recall that the holding cost at ports is chosen randomly from [1, 10] and the revenue generated by satisfying demand is chosen to be proportional to the distance (proportionality constant being chosen randomly from [100, 200]) between the origin and destination ports. Note that as the distance between ports is chosen from [2, 42] days, the revenue generated is chosen from ∼[200, 8,000]. Thus, the first scenario represents the case when the transshipment cost is low and is comparable to the holding cost at a port. The third scenario represents the case when the transshipment cost is very high and is comparable to the revenue generated by satisfying demand. Such high transshipment costs are highly unlikely; however, we discuss this scenario to present an extreme case.

Our computations yield that when the cost of transshipment is of the order of the holding cost at a port or is low compared with the revenue generated by satisfying demand, the routing decision in both networks is similar. However, as the transshipment cost increases, the routing decisions change. Specifically, as the transshipment cost increases from 20 to 1,000 units, the percentage change in the amount of demand satisfied increases from 0% to ∼36%. We note that as the transshipments become more and more expensive, the percentage of the cargo transshipped decreases. An anomaly occurs in the first row of the last column of Table 5 as the percentage of transshipped cargo increases from 12.84% to 15.17% when the transshipment cost increases from 100 units to 1,000 units. This occurs because as the transshipment cost increases, not only do the transshipments decrease but also the demand that is satisfied decreases in many cases because the routing options become limited. Thus for the last column the numerator as well as the denominator decreases. For instances in the class P6S186, the denominator decreases faster than the numerator because for this class of instances we have very few demand pairs (few things to route) and on average very few selected cycles (very few alternative routing options).

## 5. Concluding Remarks and Future Research

In this paper we presented a new mathematical model for the simultaneous ship-scheduling and containerized cargo-routing problem for liner shipping. The proposed model captures the important weekly frequency constraint faced by the carriers and allows them to take advantage of transshipping cargo. The structure of the model makes it well suited for decomposition, leading to efficient algorithms. Effective service routes for ships are generated selectively in a column generation setting using an iterative search algorithm. Finally, the proposed solution approach is tested on various test classes. Considering the preliminary results obtained, we believe that the suggested solution approach has the potential to help planners

develop better routes for a fleet of up to 100 ships. The planners can also add their predetermined service routes to the model as a set of initial cycles and thus be a part of the solution process to obtain a solution that is a user's rather than a computer's solution. Our results indicate high percentage utilization of ships' capacities and a significant number of transshipments in the final solution.

Our aim in this paper is to provide a basic framework for simultaneous ship scheduling and cargo routing. The model and the solution strategy presented here can be enhanced in different ways.

Next, we present some directions for future research. The model presented in this paper allows for transshipping the cargo from one ship to another. At the end of §4.4 we presented an approach to account for transshipment costs during cargo routing. However, the model does not take into account the transshipment costs while designing the service routes. Further research is required to extend or modify the model to include transshipment costs. This aspect is expected to increase the complexity of the model and the solution procedures significantly.

In this paper we allow only one ship type to maintain weekly frequency on a service route. This provides the same capacity in the network every week and is useful when a carrier faces the same demand each week. However, it is possible that the demand structure is not the same each week. Further research is required to allow for multiple ship types on a service route. Changes in demand from week to week can be incorporated easily by expanding the planning horizon; however, incorporating cycles with multiple ship types will require changes in the model and the cycle generation scheme.

In terms of the solution approach, in the pure column generation algorithm and the Benders decomposition-based algorithm, no new columns are generated when solving the integer program. New columns can be generated by solving the integer program in a branch-and-price (rather than the branch-and-bound used in this paper) framework. The branch-and-price framework is expected to improve the solution quality. However, there are many important and challenging issues that must be resolved to develop a successful branch-and-price algorithm. Specifically, a good branching rule needs to be devised. Standard branching on the cycle or the $x_C$ variables creates a problem along the branch where a variable has been set to zero. $x_C = 0$ means that cycle $C$ needs to be excluded. However, it is possible that the next time the pricing problem is solved to generate a profitable cycle in this branch, the optimal solution is precisely the cycle $C$. Thus the second best cycle must be considered. Moreover, at depth $l$ in the branch-and-price tree it might be necessary to construct the $l$th best cycle. Note that a successful branch-and-price algorithm requires a pricing problem that can be solved very efficiently, as it will be invoked many times. Explicitly excluding the specified cycles from the pricing problem is computationally expensive. Even if a pool of cycles is generated at every column generation step, one needs to keep track of all the cycles that need to be excluded. Furthermore, because commercial software such as CPLEX cannot handle the branch-and-price framework, managing the search tree efficiently poses many implementation challenges, such as deciding which nodes to branch on and which search technique (e.g., breadth first search, depth first search, best bound, etc.) to use.

## Acknowledgments

The second author was supported in part under National Science Foundation Grant DMI-0238815. The authors thank the anonymous referees for their valuable suggestions.

## References

Ahuja, R. K., J. B. Orlin, D. Sharma. 2003. A composite very large scale neighborhood structure for the capacitated minimum spanning tree problem. *Oper. Res. Lett.* **31** 185–194.

American Association of Port Authorities. 2006. America's ports today. AAPA Policy Paper, http://www.aapa-ports.org/.

APL. 2005. American President Lines. http://www.apl.com.

Barnhart, C., E. L. Johnson, R. Anbil, L. Hatay. 1994. A column-generation technique for the long-haul crew-assignment problem. T. Ciriano, R. Leachman, eds. *Optimization in Industry 2: Mathematical Programming and Modeling Techniques in Practice*. John Wiley & Sons, New York, 7–24.

Barry Rogliano Salles-AlphaLiner. 2006. Liner shipping report. http://www.alphaliner.com.

Bendall, H. B., A. F. Stent. 1999. Longhaul feeder service in an era of changing technology: An Asia-Pacific perspective. *Maritime Policy Management* **26**(2) 145–159.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4** 238–252.

Bertsimas, D., J. N. Tsitsiklis. 1997. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA.

Cheung, R. K., C.-Y. Chen. 1998. A two-stage stochastic network model and solution methods for the dynamic empty container allocation problem. *Transportation Sci.* **32**(2) 142–162.

Christiansen, M., B. Nygreen. 1998. A method for solving ship routing problems with inventory constraints. *Ann. Oper. Res.* **81** 357–378.

Christiansen, M., K. Fagerholt, D. Ronen. 2004. Ship routing and scheduling: Status and perspectives. *Transportation Sci.* **38**(1) 1–18.

Cordeau, J.-F., F. Soumis, J. Desrosiers. 2000. A Benders decomposition approach for the locomotive and car assignment problem. *Transportation Sci.* **34**(2) 133–149.

Cordeau, J.-F., F. Soumis, J. Desrosiers. 2001a. Simultaneous assignment of locomotives and cars to passenger trains. *Oper. Res.* **49** 531–548.

Cordeau, J.-F., G. Stojkovic, F. Soumis, J. Desrosiers. 2001b. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Sci.* **35**(4) 375–388.

Drewry. 2001. *The Drewry Container Market Quarterly*, September issue. Drewry Shipping Consultants, Ltd., London.

Fagerholt, K. 1999. Optimal fleet design in a ship routing problem. *Internat. Trans. Oper. Res.* **6** 453–464.

Florian, M., G. Bushell, J. Ferland, G. Guerin, L. Nastansky. 1976. The engine scheduling problem in a railway network. *INFOR* **14** 121–138.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freedman and Co., San Francisco, 214–215.

Hingorani, N., D. Moore, K. Tornqvist. 2005. Setting a new course in the container shipping industry. IBM Technical Report, *IBM Institute for Business Value Study*.

Imai, A., E. Nishimura, S. Papadimitriou, M. Liu. 2006. The economic viability of container mega-ships. *Transportation Res. E* **42** 21–41.

Lin, S., B. W. Kernighan. 1973. An effective heuristic algorithm for the travelling salesman problem. *Oper. Res.* **21**(2) 498–516.

McDaniel, D., M. Devine. 1977. A modified Benders partitioning algorithm for mixed integer programming. *Management Sci.* **24** 312–379.

OOCL. 2005. Orient Overseas Container Line. http://www.oocl.com.

Perakis, A. N. 2002. Fleet operations optimization and fleet deployment. C. T. Grammenos, ed. *The Handbook of Maritime Economics and Business*. Lloyd's of London, London, 580–597.

Rana, K., R. G. Vickson. 1991. Routing container ships using Lagrangean relaxation and decomposition. *Transportation Sci.* **25**(3) 201–214.

ROI. 2002. Profit optimization for container carriers. http://www.imsworldgroup.com/Downloads/ROI%20Product@Services%20v1.8.pdf.

Ronen, D. 1983. Cargo ships routing and scheduling: Survey of models and problems. *Eur. J. Oper. Res.* **12** 119–126.

Ronen, D. 1993. Ship scheduling: The last decade. *Eur. J. Oper. Res.* **71**(3) 325–333.

Shen, W. S., C. M. Khoong. 1995. A DSS for empty container distribution planning. *Decision Support Systems* **15** 75–82.

Song, D. W., P. M. Panayides. 2002. A conceptual application of cooperative game theory to liner shipping strategic alliances. *Maritime Policy Management* **29**(3) 285–301.

United Nations Conference on Trade and Development. 2006. Review of maritime transport, UNCTAD Secretariat Report, http://www.unctad.org/en/docs/rmt2006_en.pdf.

United States Customs Service. 2003. Singapore, the World's busiest seaport, implements the container security initiative and begins to target and pre-screen cargo destined for U.S. (March 17, 2003) http://www.cbp.gov/xp/cgov/newsroom/press_releases/archives/cbp_press_releases/032003/03172003.xml.

Vander Wiel, R. J., N. V. Sahinidis. 1996. An exact solution approach for the time dependent travelling salesman problem. *Naval Res. Logist.* **43** 797–820.