A PDDL Domain for the Liner Shipping Fleet Repositioning Problem

Kevin Tierney (IT University of Copenhagen)
Amanda Coles (King's College London)
Andrew Coles (King's College London)
Rune Møller Jensen (IT University of Copenhagen)

IT University Technical Report Series **TR-2012-152**

ISSN 1600–6100 **Febuary 2012**

ISSN 1600–6100

ISBN 978-87-7949-253-0

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: **+45 72 18 50 00**
Telefax: **+45 72 18 50 01**
Web **`www.itu.dk`**

A PDDL Domain for the Liner Shipping Fleet Repositioning Problem

Kevin Tierney (IT University of Copenhagen)
Amanda Coles (King's College London)
Andrew Coles (King's College London)
Rune Møller Jensen (IT University of Copenhagen)

Abstract

The Liner Shipping Fleet Repositioning Problem (LSFRP) poses a large financial burden on liner shipping firms. During repositioning, vessels are moved between services in a liner shipping network. The LSFRP is characterized by chains of interacting activities, many of which have costs that are a function of their duration; for example, sailing slowly between two ports is cheaper than sailing quickly. Despite its great industrial importance, the LSFRP has received little attention in the literature. We model the LSFRP using PDDL and solve it using the planner POPF.

1 Introduction

Situated at the heart of global trade, liner shipping networks transported over 1.3 billion tons of cargo on over 9,600 container vessels in 2011 (UNCTAD 2011). Vessels are regularly repositioned between services in liner shipping networks to adjust the networks to the world economy and stay competitive. Since repositioning a single vessel can cost hundreds of thousands of US dollars, optimizing the repositioning activities of vessels is an important problem to the liner shipping industry.

The Liner Shipping Fleet Repositioning Problem (LSFRP) consists of finding minimal cost sequences of activities that move vessels from one service to another within a liner shipping network. Fleet repositioning involves sailing and loading activities subject to complex handling and timing restrictions. As is the case for many industrial problems, the objective is cost minimization (including costs for CO_2 emissions and pollution), and it is important that all cost elements, including those that are only loosely coupled with activity choices, can be accurately modeled.

In this report, we model the LSFRP in PDDL [6], a domain-specific language for modeling automated planning problems. We model the problem both forwards and backwards in order to investigate whether this improves solver performance. We solve the PDDL model with the planner POPF [4] by extending its ability to handle Timed Initial Literals. We present experimental results showing the solving time and solution quality for POPF and discuss improvements to POPF to help it solve the LSFRP faster.

2 Liner Shipping Fleet Repositioning

Container vessels are routinely *repositioned*, i.e. moved from one service to another, in order to better orient a liner shipping network to the economy. A liner shipping network consists of a set of circular routes, called services, that visit ports on a regular, usually weekly, schedule. Shipping lines regularly add and remove services from their networks in order to stay competitive, requiring vessel repositionings. The repositioning of vessels is expensive due to the cost of fuel (in the region of hundreds of thousands of dollars) and the revenue lost when a ship is not on a service carrying customers' cargo. Given that liner shippers around the world reposition hundreds of vessels per year, optimizing vessel movements can significantly reduce the economic and environmental burdens of containerized shipping.

Given a set of vessels, where each vessel is assigned an initial service and a goal service, the aim of the LSFRP is to reposition each vessel to its goal service within a given time period at minimal cost. Each vessel begins its

repositioning when it *phases out* from its current service, meaning it ceases regular operations on the service. Vessels may phase out of any port on the service they are sailing on at the time the port is normally called by the service. After a vessel has phased out, it may undertake activities that are not part of its normal operations until it *phases in* at its goal service, which, like phasing out, must happen at a goal service port at the time the goal service is scheduled to call it. Throughout the time between the phase-out and the phase-in, except where noted, the repositioning vessel pays a fixed hourly cost, referred to as the *hotel cost* in shipping parlance. For example, in Figure 1, a vessel on its initial service CHX could phase-out in TPP and sail to the port of BLB on its goal service. The hotel cost is assessed for the entire time of sailing between TPP and BLB, as well as any time spent waiting at BLB once the vessel arrives.

Between the phase-out and the phase-in, a vessel may undertake the following activities. First, vessels may *sail* directly between two ports, incurring a cost that actually declines as the duration of the sailing increases, due to the fuel efficiencies of engines at low speeds. Second, a vessel may also sail with equipment, e.g. empty containers, from ports where they are in excess to ports where they are in demand, earning a profit per TEU[1] carried, but incurring a delay to load and unload the equipment. And third, a vessel may perform a *sail-on-service* (SOS), in which the vessel replaces a vessel on an already running service.

SOS opportunities are desirable because the repositioning vessel incurs no hotel or fuel costs on an SOS, but cargo may need to be *transshipped* from the replaced vessel to the repositioning vessel, depending on where the repositioning vessel starts the SOS. Cargo transshipments are subject to a fee per TEU transshipped and vessels are delayed depending on how much cargo must be transferred. When an SOS opportunity is utilized, the replaced vessel is free to be laid up or leased out. For the purposes of the LSFRP, we ignore what happens to the replaced vessel. An SOS may only start at certain ports due to *cabotage* restrictions, which are laws that prevent foreign vessels from offering domestic cargo services. Due to the complexity of cabotage laws around the world, we simply maintain a blacklist of ports for each vessel. Note that while we do not take a detailed view of cargo flows, the activities we allow a vessel to undertake are chosen such that they do not significantly disrupt the network's cargo flows.

One of the key difficulties in the LSFRP lies in the constraints that dictate how vessels may phase in to a new service. It is essential that the liner shipping nature of the service is enforced, meaning that once a vessel visits a port on the goal service, there must be a vessel visiting that port in every subsequent week within the planning horizon. This constraint is a business requirement, as once a service is started, customers expect to be able to ship their cargo without interuption. This constraint, however, leads to $v!$ different orderings at each port on the goal service, where v is the number of vessels being repositioned. Thus, each ordering at each port is potentially associated with a different cost.

We performed a case study with our industrial partner to better understand the nature of fleet repositioning problems. A new service in the network, the "Intra-WCSA", required three vessels[2] that were sailing on services in Asia. Repositioning coordinators were tasked with moving the vessels to the Intra-WCSA at as low a cost as possible.

Figure 1 shows a subset of the case study and the cost saving opportunities that repositioning coordinators had available to them. The Intra-WCSA required three vessels, one of which was on the CHX service. Two further vessels were on services that are not shown in the figure, and were also in southeast Asia. Vessels could carry equipment from northern China to South America, as well as utilize the AC3 service as a sail-on-service opportunity. The problem was solved by hand, as no automated tools exist to assist in solving the LSFRP, with the solution sending vessels on the AC3 SOS opportunity to BLB, where they phased in.

The LSFRP has received little attention in the literature, and was not mentioned in either of the most influential surveys of work in the liner shipping domain [2, 3]. Neither the Fleet Deployment Problem [11] nor the Liner Shipping Network Design Problem [10] deals with the repositioning of ships or the important phase-in requirements. Tramp shipping problems, such as [9], also differ from the LSFRP due to a lack of cost-saving activities for vessels. Martin W. Andersen discusses the network transition problem in his PhD thesis [1], which involves vessel repositioning within a feeder network, but does not consider cost saving components like slow steaming or equipment repositioning.

It has been observed in both the AI-planning and OR-scheduling fields (e.g. [8, 12]) that the compound objectives of real-world problems, such as those found in the LSFRP, are often hard to express in terms of the simple objective criteria like makespan and tardiness minimization. Scheduling [8] has focused mainly on problems that only involve a small, fixed set of choices, while planning problems like the LSFRP often involve cascading sets of choices that

[1]TEU stands for *twenty-foot equivalent unit* and represents a single twenty-foot intermodal container.

[2]For reasons of confidentiality, some details of the case study have been changed.

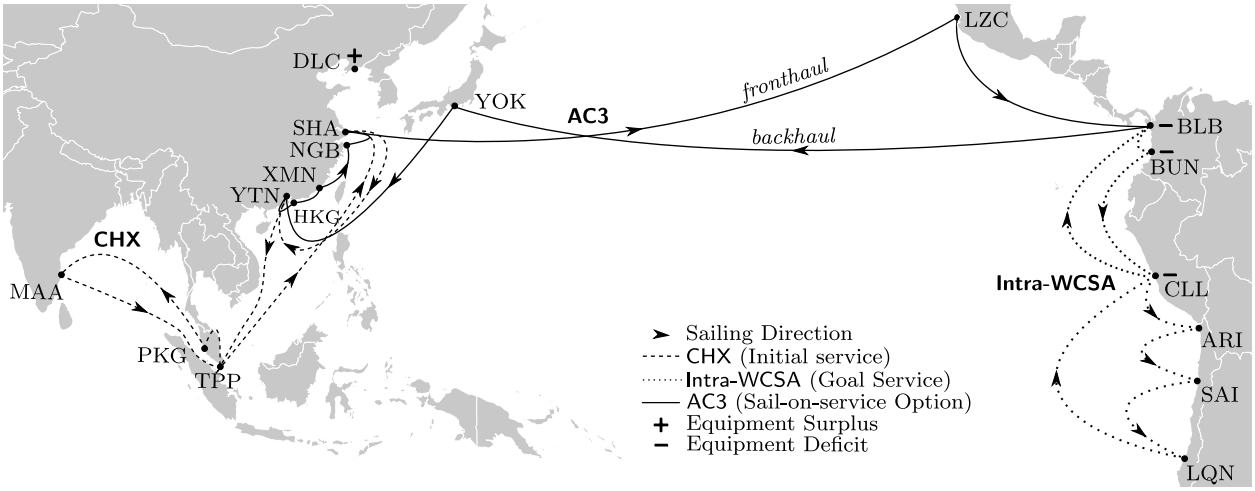

Figure 1: A subset of the case study we performed with our industrial collaborator is shown. A vessel on the CHX service must be repositioned to the new Intra-WCSA service. The vessel travels to the AC3 service and performs a *sail-on-service*, allowing it to sail to BLB at significantly lower cost.

interact in complex ways [12]. Another limitation is that mainstream scheduling research has focused mainly on the optimization of selected, simple objective criteria such as minimizing makespan or minimizing tardiness [13].

3 LSFRP PDDL Domain

The PDDL model of the LSFRP has interesting temporal features: required concurrency [5], timed initial-literals (TILs) [6] and duration-dependent effects. The PDDL domain is available for download at `http://www.decisionoptimizationlab.dk/lsfrp_pddl`. The model is based around the idea that in the initial state, all vessels are sailing on their initial services, and we therefore do not need to take their costs into account. The planner then makes a decision to phase out each vessel from its initial service, putting the vessel into a state of transit. While the vessel is in this state, its activities count towards the overall cost of the repositioning. The planner finally makes a decision as to where the vessels should be phased in and when each vessel should be phased in. Once phased-in to the goal service, the vessels no longer generate any repositioning costs and the goal state of the model is reached.

3.1 Timed Initial Literals

The model makes extensive use of timed initial literals, to encode the fact that any decision made must be suitable given the context in which repositioning is occurring. Our time basis is one PDDL time unit per hour.

The 'pre-repositioning' itinerary of a vessel, specifically the times at which it could leave that itinerary and be repositioned, is encoded using `(vessel-may-phase-out ?vessel ?port)`. If a vessel v could phase out at at time t from port p, due to being in port and having unloaded all cargo at that time, the relevant fact is added as a TIL at time t; and deleted as a TIL shortly afterwards, to reflect the fact that if the opportunity is not taken then, the vessel will continue on its incumbent service. As will be shown later, when a vessel phases out, it then is at the location dictated by the relevant then-true TIL.

The 'post-repositioning' options for a vessel — the services it could phase in to join, and at which port and time that would occur — are encoded using `(vessel-may-phase-in ?portpi)`. For instance, if one option is to start a service that calls at some port, once a week, at mid-day on a Tuesday, then a TIL for this port will be added, once a week, at time points corresponding to this; and TIL deleted shortly afterwards.

Whichever port is chosen for all the vessels to phase in at, only one may phase in per week. The aim of repositioning, in our problems, is to set up a regular service at a port, so we do not wish to have, for instance, two vessels one week, and none the next. This is regulated by a fact `(phasein-week-open)`, added as a TIL at the start of each week (time $0, 168, 336, ...$). As will be shown later, each action corresponding to phasing in a vessel deletes this fact; and hence no other vessel may phase in until the fact is added again, which at the earliest is the following week.

To encode opportunities for sailing on a service, the predicate `(sos-open ?sos ?pfrom ?pto)` is used. The fact denoting that the opportunity is available is added by a TIL at the appropriate time, and deleted by another shortly afterwards. Thus, if a vessel is to sail on a service, it must do so then, and only then.

3.2 Predicates

Other than the facts whose truth value depends on TILs, a number of facts are used to record various aspects of the state of the world.

3.2.1 Vessels

The status of a vessel is governed by a number of facts:

`(on-init-service ?vessel)` is true whilst the vessel has not yet phased out;

`(vessel-at ?vessel ?port)` records its location once it has phased out;

`(can-sail ?vessel)` records that the vessel can sail. Sensible operation of a vessel precludes sailing from A to B to C, rather than from A to C directly, so this fact precludes chaining sail actions.

`(in-transit ?vessel)` records that the vessel is in transit between two services, i.e. it has phased out but not yet phased in;

`(phased-in ?vessel)` is true once it has phased in, i.e. it has been repositioned.

`(sos-or-equipment-allowed ?vessel)` — as a vessel cannot both sail on service and sail equipment during repositioning, this fact is true initially for each vessel, and deleted if it sails on a service or sails with equipment.

3.2.2 Sailing Possibilities

Sailing possibilities are determined by:

`(sailing-allowed ?pfrom ?pto)` is true if it is possible to sail from port `?pfrom` to `?pto`.

`(sos-open ?sos ?pfrom ?pto)` is true at times when a sail-on-service option is available from port `?pfrom` to `?pto`. As noted earlier, this is controlled by timed-initial literals.

`(unused ?sos)` is true if a given sail-on-service option has not been used. This is true initially and deleted the moment the opportunity is taken.

`(equipment-sailing ?pfrom ?pto)` is true if there is a equipment sailing run from port `?pfrom` to `?pto`.

3.2.3 Phasing In

A number of facts are used to regulate the 'block' nature of phasing in, i.e. that vessels must phase in in subsequent weeks, at the port where the first phase in occurred:

`(block-phase-in-start)` is true initially, and denotes that no ship has yet phased in.

`(first-phasein-week-defined)` becomes true once the first vessel has phased in.

```
(:process time-is-passing
  :parameters ()
  :precondition  (can-start-time)
  :effect (increase (time-elapsed) (* #t 1.0) )
)
```

Figure 2: Encoding an Absolute Measure of Time Elapsed

(first-phasein-port ?port) records at which port a vessel first phased in.

Additionally, as noted earlier, phasing in is restricted by two TILs: **(vessel-may-phase-in ?portpi)** and **(phasein-week-open)**. The latter is added once weekly by a TIL, and deleted when a vessel phases in; the former is managed exclusively by TILs.

3.2.4 Phasing Out and Hotel Cost

Five predicates are used to regulate the timing of phasing out and the counting of 'hotel cost':

(allowed-to-start-cost-calc ?vessel) is true initially, and allows the hotel cost counting action for a vessel to be started.

(allowed-to-end-cost-calc ?vessel) becomes true once hotel cost can stop being counted, either when a vessel starts sailing on a service, or if it phases in.

(cost-calc-mutex ?vessel) is a semaphore fact, ensuring that hotel cost calculation cannot self-overlap for a given vessel.

(allowed-to-phase-out ?vessel) is true once hotel cost is being calculated;

(vessel-may-phase-out ?vessel ?port) is true only at times where the given vessel may phase out of its initial service at the given port. It is managed exclusively by TILs.

3.3 Functions

The domain encoding makes use of numbers to constrain the timing of activities, and to reflect their costs.

3.3.1 Phasing In

Three functions are used to encode the timing constraints on phasing in:

(first-phasein-week) is set to the week in which the first vessel phase in occurred;

(weeks-within) encodes over how many weeks the phase in window spans, i.e. the number of vessels in the problem;

(time-elapsed) is used to give an absolute reference to how much time has elapsed at the point at which vessels phase in. It initially holds the value zero, and is updated constantly at a rate of 1 per unit time by the process [7] shown in Figure 2. Its precondition is a dummy tautologous fact, i.e. it is always executing.

3.3.2 Plan Cost and Sailing Times

The cost of the current plan is captured by the variable `(total-cost)`. To determine the costs of actions, a number of constant-valued functions are defined in the initial state. First, between phasing out and phasing in, unless a vessel is sailing on a service, a per-hour 'hotel cost' made be paid to the crew. This per-hour cost is encoded in the variable **`(hotel-cost ?v - vessel)`**.

The costs of operations in addition to this depends on the actions involved. For normal 'sail' actions:

`(min-time-to-sail ?vessel - vessel ?pfrom ?pto - port)` is the minimum time to sail between two ports for a given vessel (at full speed);

`(max-time-to-sail ?vessel - vessel ?pfrom ?pto - port)` is the maximum time to sail between two ports for a given vessel (at minimum speed);

`(fixed-sail-cost ?v - vessel ?pfrom ?pto - port)` is the maximum operational cost of sailing between two ports for a given vessel. This is paid at the minimum sailing time.

`(variable-sail-cost ?v - vessel ?pfrom ?pto - port)` is a negative number, denoting the cost of taking one hour longer to complete a given sail action.

For sailing with equipment, the minimum and maximum sailing time are encoded in the same way. The operational cost is different, though, so for this we use a pair of functions:

`(fixed-eqp-cost ?v - vessel ?pfrom ?pto - port)` is the minimum cost;

`(variable-eqp-cost ?v - vessel ?pfrom ?pto - port)` is the amount extra paid per hour to complete the sail-equipment operation faster than the minimum time.

Finally, for sailing on a service, the duration of the service is fixed by **`(sos-duration ?sos - sos ?pfrom ?pto - port)`**. No operational cost must be paid, and the hotel cost instead of being the per-hour figure used otherwise is set by **`(sos-hotel-cost ?sos - sos ?pfrom ?pto - port)`**.

3.4 Actions

At a high level, the model requires 6 actions:

- `phase-out` and `phase-in`, to mark the start and end of the repositioning of each vessel;
- `calculate-hotel-cost`. to count the hotel cost when necessary;
- `sail`, `sail-on-service` and `sail-with-equipment`, to move vessels whilst, optionally, taking advantage of opportunities to sail on a service, or on an equipment run.

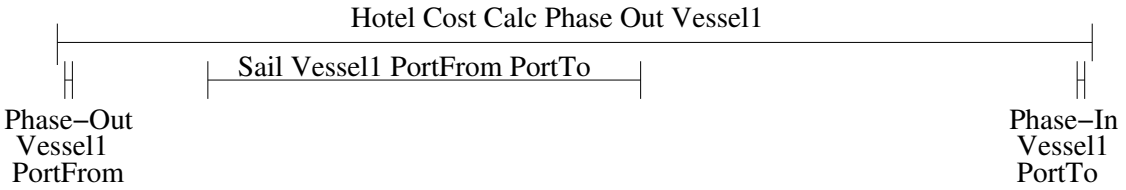

Figure 3: Actions for a single vessel, using a single `sail` action.

Due to POPF having limited support for ADL, we split the calculation of hotel cost and phasing in each into two actions.

In the simple case, the progression of actions for a single vessel is shown in Figure 3. The Hotel Cost Calculating action and the Sail action are durative actions, whose execution takes time. The actions for phasing in and phasing out are instantaneous. The construction of the domain, as we will discuss, enforces that the hotel cost calculation is an *envelope* around all the activity for the vessel: it must start before phasing out, and must end before phasing in. Note that due to the temporal constraints in the problem, discussed in Section 3.1, there may need to be gaps between the activities within the hotel-cost envelope. For instance, as `phase-in` and `phase-out` opportunities are constrained, and `sail` actions have an upper duration bound, there may be slack either side of the sail action. In a good solution, this will be minimised in such a way as to reduce overall costs.

3.4.1 Starting Hotel Cost Before Phase Out

Whilst a vessel is phased out and not sailing on a service, 'hotel cost' must be paid. We capture this in our model by using a *envelope* actions, which encompass the activities of a vessel, modulo sailing on a service.

The first of these is `(hotel-cost-calc-phase-out)`, parameterised by a vessel `?vessel`. This can be started if:

1. `(on-init-service ?vessel)`: the vessel is still on its initial service, i.e. has not yet phased out.
2. `(allow-to-start-cost-calc ?vessel)`: hotel cost calculation for the vessel can be started. This is true initially.
3. `(cost-calc-mutex ?vessel)`: the hotel cost is not already being calculated;

The action cannot then end until `(allow-to-end-cost-calc ?vessel)` is true, which occurs either once the vessel has phased in, or if a sail-on-service action has commenced.

To join the action logically to the other actions in the domain, the effects of the start of the action note that the vessel can phase out (i.e. `(allowed-to-phase-out ?vessel)`). Also, a number of steps are taken to ensure accurate cost calculating: `allow-to-start-cost-calc` is deleted at the start; `cost-calc-mutex` is deleted at the start and added at the end, to act as a semaphore; and the `allow-to-end-cost-calc` fact noted above as being required at the end is deleted at the end.

The cost of the action itself is encoded as a numeric effect at the start of the action that depends on the duration:

```
(at start (increase (total-cost) (* (hotel-cost ?vessel) ?duration)))
```

Thus, the total cost of the plan is increased in proportion to the duration of this action. As the logical conditions above ensure that `(hotel-cost-calc-phase-out)` must start before phasing out and can only end at phasing in or starting a sail-on-service, this ensures the per-hour cost for this period is paid.

Note that there is no direct mechanism for ensuring phasing out occurs immediately after starting to calculate the hotel cost, nor to ensure that the hotel cost calculation finishes immediately after phasing in/starting to sail on a service. This is not an issue, though, as any sensible cost-sensitive assignment of timestamps to the start and the end of the hotel cost actions will do this to minimise the total cost paid.

3.4.2 Re-Starting Hotel Cost after Sailing On Service

Having provided facility for the `(hotel-cost-calc-phase-out)` action to finish upon the commencement of sailing on a service, we must also capture the fact that it must resume afterwards, finishing then only once the vessel has phased in (or perhaps sails on another service). For this, a second envelope action `(hotel-cost-calc-sos)` is used. This is almost identical, apart from an additional parameter `?sos` recording the specific service after which hotel-cost calculation is being restarted, and two minor changes to the action itself:

- The precondition `(on-init-service ?vessel)` is replaced with `(sailing-on-service ?vessel ?sos)`. Thus, rather than requiring that the vessel has not yet phased out, it requires that the vessel is sailing on a service.

- The start effect `(allowed-to-phase-out ?vessel)` is replaced with `(allowed-to-end-sos ?vessel)`, i.e. starting it provides the logical ability to end sailing on service, rather than allowing phasing out.

We will revisit the interaction between sailing on service and hotel cost calculation later, when considering the sail-on-service action.

3.4.3 Phase-out

The phase out action is parameterised by a vessel `?vessel` and a port `?port`. It can be applied if three conditions hold:

1. `(vessel-may-phase-out ?vessel ?port)`: the vessel would currently be at this port on its initial itinerary. This is regulated by TILs, as described in Section 3.1.
2. `(on-init-service ?vessel)`: the vessel is still on its initial service, i.e. has not yet phased out.
3. `(allowed-to-phase-out ?vessel)`: the action to count hotel cost for the vessel has been started.

The latter two of these are deleted by the application of the action; then to reflect that the activities vessel are now under the control of the plan, three facts are added in their place:

1. `(vessel-at ?vessel ?port)`: the vessel is at the port where it phased out.
2. `(in-transit ?vessel)`: the vessel is now being repositioned;
3. `(can-sail ?vessel)`: the vessel can sail.

3.4.4 Sail

Having phased out, a vessel may sail. In the simple case, the sailing action is solely for the purpose of repositioning. We capture this with an action `sail`, parameterised by a vessel `?vessel`, and source and destination ports `?pfrom`,`?pto`.

The action can be started if four conditions are met:

1. `(in-transit ?vessel)`: the vessel has left its initial service.
2. `(vessel-at ?vessel ?pfrom)`: the vessel is at the source port.
3. `(sailing-allowed ?pfrom ?pto)`: it is possible to sail from there to `?pto`.
4. `(can-sail ?vessel)`: the last action to act upon `?vessel` was not a `sail` action. Two such actions should never be applied back-to-back, instead the vessel should omit the intermediate location.

The logical effects of the action are as one would expect: the `vessel-at` condition is deleted at the start; the appropriate `vessel-at` added at the end; and the `can-sail` condition is deleted, to reflect that the vessel has just sailed. Plan cost is updated according to:

$$\texttt{(total-cost)} \;\texttt{+=}\; \texttt{(fixed-sail-cost ?vessel ?pfrom ?pto)} + (\texttt{(variable-sail-cost ?vessel ?pfrom ?pto)} \times \texttt{?duration})$$

The duration of the action is governed by an inequality: it must lie in the range `(min-time-to-sail ?vessel ?pfrom ?pto)` to `(max-time-to-sail ?vessel ?pfrom ?pto)`, equivalent to sailing at between maximum and minimum speed. Thus, the cost formula linearly interpolates cost between these two points.

3.4.5 Sail on Service

Sailing on a service is modelled by an action `sail-on-service`, parameterised by a vessel `?vessel`, source and destination ports `?pfrom`,`?pto`, and a unique identifier `?sos` for the service on which the vessel is to sail. Unlike `sail`, its duration is fixed, according to `(sos-duration ?sos ?pfrom ?pto)`. The preconditions of the actions are slightly more involved, as unlike sail, it does not work in isolation — it must be done in the context of the service timetable — and each vessel can only do one of either sail on service or sail equipment. Thus, to start `sail-on-service` the conditions are:

1. `(in-transit ?vessel)` (as with sail);
2. `(vessel-at ?vessel ?pfrom)` (as with sail);
3. `(sos-open ?sos ?pfrom ?pto)`: a TIL, true only when the sail-on-service opportunity is available.
4. `(unused ?sos)`: no vessel must yet have sailed on this service
5. `(sos-or-equipment-allowed ?vessel)`: the vessel must not yet have sailed on a service/with equipment.

Upon the start of its execution, the latter four of these are deleted: the vessel has departed, the opportunity has been taken, and the vessel can only do conventional sail actions from thereon. As with `sail`, `(vessel-at ?vessel ?pto)` is added at the end of the action. Additionally, to permit the vessel to sail once again (because `sail`, `sail-on-service`, `sail` is entirely reasonable), the fact `(can-sail ?vessel)` is added at the end of the action too.

The remaining consideration for sailing on service is how to model the fact that during its execution, hotel cost should not be counted. To do this, we allow hotel cost calculation to finish as soon as the action has started; whilst ensuring that it must begin, again, before the action has finished. This is achieved as follows:

- At the start of the action, the fact `(allowed-to-end-cost-calc ?vessel)` is added, thereby allowing `(hotel-cost-calc-phase-out)` (described earlier) to finish.
- Also at the start of the action , the fact `(allowed-to-start-cost-calc ?vessel)` is added, thereby allowing `(hotel-cost-calc-sos)` (described earlier) to start. (The `(cost-calc-mutex ?vessel)` fact ensures that the previously executing hotel cost action must finish before this new one.)
- Finally, the action has an end condition `(allowed-to-end-sos ?vessel)`. This is only added by `(hotel-cost-calc-sos)`, thereby ensuring it actually is started (not just that it may).

As there is some hotel cost for sailing on a service, the action then has an effect:

```
(at start (increase (total-cost) (sos-hotel-cost ?sos ?pfrom ?pto)))
```

An example action sequence for a vessel employing a `sail-on-service` action is shown in Figure 4. As can be seen, there is no hotel-cost-calculating action running during the period whilst the `sail-on-service` action is executing; but there is one running at all other times.

Figure 4: Actions for a single vessel, including a `sail-on-service` action.

Note that the Figure only shows one execution case: using a `sail` action to reach 'PortSOS1', a port at which the sail-on-service opportunity 'SOSi' begins. Assuming a suitable action exists, it is also possible to `sail` after a `sail-on-service`, in cases where the destination port of the service is not the desired phase-in port. This can be done with or without sailing before the `sail-on-service`. Finally, in cases where a vessel can be repositioned using a single `sail-on-service`, the model permits this also.

3.4.6 Sail Equipment

Sailing equipment is easier to model than sailing on a service, as hotel cost does not need to be suspended during the action. The action is very similar to `sail`, with the same duration bounds. The key changes are:

- Rather than requiring `(can-sail ?vessel)` and `(sailing-allowed ?pfrom ?pto)`, the action requires `(equipment-sailing ?pfrom ?pto)` and `(sos-or-equipment-allowed ?vessel)`. As such, it can only be applied between ports for which there is an equipment-sailing run; and if the vessel has not yet sailed on a service or with equipment.
- The coefficients for calculating the action cost are different:

$$\texttt{(total-cost)} \;\texttt{+=}\; \texttt{(fixed-eqp-sail-cost ?vessel ?pfrom ?pto)} + \big(\texttt{(variable-eqp-sail-cost ?vessel ?pfrom ?pto)} \times \texttt{?duration}\big)$$

If applied, the `vessel-at` location changes as with sail; the fact `(sos-or-equipment-allowed ?vessel)` is deleted; and at the end of the action, to permit the vessel to sail once again (because `sail`, `sail-equipment`, `sail` is entirely reasonable), the fact `(can-sail ?vessel)` is added.

3.4.7 Phase-in

The `phase-in` action is split in to two actions: `phase-in-1st` and `phase-in-block`. This decomposition of the phase-in is necessary to ensure that vessels phase in to the goal service in subsequent week. The first vessel arriving is a special case: it can phase-in at any suitable port, at any suitable time. The second and later vessels must phase-in in the weeks following that first phase-in, at the same port.

First, consider the commonalities between both of the variants of the action:

- Both are parameterised by a port `?portpi` and a vessel `?vessel`;
- The vessel must be `vessel-at` the port;
- The vessel must be `in-transit`; this is deleted by the action.
- Nothing else must have previously phased in that week, i.e. `phasein-week-open`; this is deleted by the action.
- Both add `(allowed-to-end-cost-calc ?vessel)`, allowing 'hotel cost' calculation to finish.
- Both add `(phased-in ?vessel)`.

The differences lie in selecting which port all vessels are to phase in at, and the regulation of the timing constraints. Obviously, only one vessel can phase in first, so `phase-in-1st` requires, then immediately deletes, the fact `(block-phase-in-start)`. Then, it sets the fact `(first-phasein-port ?portpi)`, setting the location at which the other vessels must phase in. For timing, to note the week in which this first phase-in occurred, with a week constituting 168 time units (hours), we use the formula:

$$\texttt{(first-phasein-week)} = \left\lfloor \frac{\texttt{(time-elapsed)}}{168} \right\rfloor$$

```
(:action PHASE-IN
  :parameters (?portpi - port ?vessel - vessel)
  :precondition (and
    (or (block-phase-in-start)
        (and (>= (time-elapsed)
                 (* (first-phasein-week) 168))
             (<  (time-elapsed)
                 (+ (* (+ (weeks-within) (first-phasein-week)) 168) 168))
             (first-phasein-week-defined)
             (first-phasein-port ?portpi)
        )
    )
    (in-transit ?vessel)
    (vessel-at ?vessel ?portpi)
    (vessel-may-phase-in ?portpi)
    (phasein-week-open)

  )
  :effect (and

    (not (phasein-week-open))
    (not (in-transit ?vessel))
    (phased-in ?vessel)
    (allowed-to-end-cost-calc ?vessel)

    (when (block-phase-in-start) (and
      (first-phasein-port ?portpi)
      (not (block-phase-in-start))
      (first-phasein-week-defined)

      (when (and (>= (time-elapsed) 0)
             (< (time-elapsed) 168)
        )
        (assign (first-phasein-week) 0)
      )
      (when (and (>= (time-elapsed) 168)
             (< (time-elapsed) 336)
        )
        (assign (first-phasein-week) 1)
      )
      ; ... remainder of 'round down' conditional effects
      )
    )
  )
)
```

Figure 5: A Single Phase in Action, using Disjunctive Preconditions and Conditional Effects

Inst.	POPF (Optimal)		POPF (Satisficing)									
	Forwards	Reversed	Standard		Makespan		No MIP relax		No-TIL-Abs		Reversed	
AC3_1_0	0.7	1.4	0.4	(0.0)	0.1	(1.7)	0.7	(0.0)	105.8	(0.0)	0.4	(0.0)
AC3_2_0	-	809.6	32.5	(0.0)	3.2	(1.6)	113.2	(0.0)	13.0	(0.1)	78.1	(0.0)
AC3_3_0	-	-	1105.1	(0.0)	117.5	(2.3)	3041.6	(0.0)	88.2	(0.1)	39.2	(0.8)
AC3_1_1e	3.3	4.0	1.7	(0.0)	0.1	(0.7)	2.3	(0.0)	1079.3	(0.3)	1.2	(1.2)
AC3_2_2ce	-	-	399.2	(0.2)	9.2	(7.3)	26.3	(1.4)	303.4	(1.3)	602.8	(1.1)
AC3_3_2c	-	-	1550.6	(0.3)	1.1	(19)	2284.2	(0.0)	31.3	(3.7)	892.5	(1.6)
AC3_3_2e	-	-	1975.5	(2.3)	10.1	(15)	226.0	(3.6)	352.6	(3.4)	699.6	(2.9)
AC3_3_2ce1	-	-	1464.2	(1.6)	10.0	(12)	204.9	(2.8)	303.4	(2.7)	690.1	(2.3)
AC3_3_2ce2	-	-	291.5	(1.3)	9.6	(11)	28.4	(2.4)	310.8	(2.3)	688.6	(1.9)
AC3_3_2ce3	-	-	303.9	(1.3)	9.7	(11)	28.4	(2.4)	314.5	(2.3)	697.2	(1.9)
AC3_3_3	-	-	348.0	(1.1)	10.3	(8.7)	29.4	(1.9)	308.4	(1.7)	603.3	(1.5)

Table 1: Results of solving the LSFRP PDDL model with POPF using several different planning heuristics with a timeout of one hour. All times are the CPU time in seconds. Figures in brackets are the best optimality gap found by POPF alongside the CPU time required to find it. The optimality gap is computed by $(c - c^*)/c^*$, where c is the plan cost and c^* is the optimal solution.

As there is no 'floor' operator in PDDL, we implement this using a sequence of conditional effects, dependent on `(time-elapsed)`. This is feasible in this problem as the planning horizon is finite: all activity must complete within a number of weeks.

Once the first phase-in has been performed, subsequent vessels use `phase-in-block` to join the goal service. Recalling that `(weeks-within)` records the number of vessels in the problem, and thus the size of the phase-in block, we enforce the phase-in block size limit with a precondition pair equivalent to:

$$\texttt{(first-phasein-week)} \leq \frac{\texttt{(time-elapsed)}}{168} \leq \big(\texttt{(weeks-within)} + \texttt{(first-phasein-week)}\big)$$

That is, the week in which the action is applied must be after the first phase-in occurred, but before `(weeks-within)` weeks after that. Note that this does not capture the fact that only one vessel can phase in per week: that is handled by each phase in action deleting `phasein-week-open`, which is added only at the start of each week by a TIL.

Note that strictly, the two phase-in action variants could be amalgamated through the use of conditional effects and disjunctive preconditions. We sketch such an action in Figure 5. As POPF only has limited support for ADL, it cannot handle this action at present. As this is also true for any other planner presently able to handle this domain, we opted for a two-action phase in for the model we describe in this report.

4 Computational Results

We created ten instances based on the case study shown in Figure 1 containing up to three vessels and various combinations of sail-on-service, equipment opportunities and cabotage restrictions. Table 1 shows the results of solving these instances using the planner POPF using a variety of heuristics, including two optimal heuristics. Instances are named based on the number of vessels and sail-on-service opportunities, and then whether they have equipment opportunities (e) or cabotage restrictions (c). The experiments were conducted on AMD Opteron 2425 HE processors with a maximum of 4GB of RAM per process, with POPF using CPLEX 12.1.

In the satisficing case, POPF exhibits a number of successes. The column 'Standard' in Table 1 demonstrates that POPF is sensitive to the metric specified, and is successfully optimizing with respect to a cost function that is not makespan. The 'Makespan' results confirm that optimizing makespan would not be a surrogate for low-cost in this domain, and indeed reflect that whilst POPF does not find optimal solutions in all problems, the solutions it is finding are relatively rather good.

As an evaluation of our modifications to POPF, the 'No MIP relax' column indicates performance when not relaxing the MIP to an LP at non-goal states. This configuration suffers from high per-state costs, limiting the search space

covered in one hour. An alternative means of avoiding the MIP is to disable TIL abstraction, which again is demonstrably worse than the 'Standard' configuration. Finally, the 'Reversed' model, though better for optimal search, gives worse performance: it forces premature commitment to the phase-in port without having considered how to sail there. This is harmless in the optimal case, where all phase-in options are considered anyway, but detrimental here.

In the optimal case, the performance of POPF was more limited. In this problem, POPF splits each temporal action is into a start and end action, necessary to preserve completeness in general temporal planning [5]. The principle disadvantage of this when solving the LSFRP is that because a state in which an action has started is different to a state in which it has not, the task of state memoization (avoiding redundant search) is far harder. The state memoization in POPF is not sophisticated enough to recognize that varying the order of starting the `hotel-cost-calc` actions for different vessels does not lead to interesting different states (similarly for unrelated `hotel-cost-calc` and `sail` actions). Thus, many effectively equivalent options are retained, cautiously, to preserve completeness. When using POPF to prove optimality (A*, admissible costs from expanding the TRPG fully) almost all of its search states arise from considering these permutations of `hotel-cost-calc` actions.

To see if an alternative formulation of the problem would improve performance (particularly in the optimal case), we also made a reversed domain. As phasing in poses some of the most restricting constraints in the problem, committing to a phase-in port sooner rather than later in search may be beneficial. In this domain, vessels begin by phasing in and end by phasing out. We did not model the problem this way initially due to the 'physics, not advice' mantra of PDDL: it is less natural, though more efficient here. Using this, POPF can prove optimality in only 3 problems: AC3_1_0 and AC3_1_1e which have 1 vessel; and AC3_2_0 which has 2 vessels. In AC3_1_1e, it expands twice as many nodes (and evaluating each takes far longer.)

The overall picture is that this is an interesting new problem for temporal-numeric planning research, and modelling it using the language subset supported by POPF has been insightful in indicating future fruitful avenues of research. In particular:

- Supporting PDDL+ [7] would allow a far cleaner model to be made. To count hotel cost, for each vessel, rather than using a durative action, a hotel-cost-calculation process could be used, conditioned by a single fact, added at the point of phase out and deleted at phase in. This would also alleviate the need to explicitly suspend cost calculation whilst sailing on a service: it would suffice to delete this fact at the start of `sail-on-service`, and add it at the end. The use of events would be another way of eliminating the need to split phase-in into two actions: phase-in would need merely to update the vessel and mark that week's opportunity as having been taken, with an event then marking the first phase-in week *iff* none has yet been defined.

- The current heuristic in POPF, and in temporal planners more widely, focus on time, primarily, and cost secondarily. When minimising makespan, this is reasonable: the heuristic guidance supports producing temporally efficient plans. In problems where cost depends on time, but where low-makespan is not necessarily low-cost, a cost-based heuristic may be more useful — that is, better guidance with respect to costs but poorer guidance with respect to time, rather than vice versa.

- State memoization is demonstrably an issue in POPF, with this domains such as this. Addressing this requires novel state memoization techniques. In the general case, the approach currently used in POPF may still be required, but it would be highly beneficial to take steps to avoid this worst-case behaviour in a useful subset of tasks.

5 Conclusion

We presented a PDDL model of a novel problem, the Liner Shipping Fleet Repositioning Problem (LSFRP), and solved it using the POPF planner. More work is required to solve these problems to optimality, as well as to improve the optimality gap of POPF with inadmissible heuristics.

6 Acknowledgements

We would like to thank our industrial collaborators Mikkel Muhldorff Sigurd and Shaun Long at Maersk Line for their support and detailed description of the fleet repositioning problem. This research is sponsored in part by the Danish Council for Strategic Research as part of the ENERPLAN research project. Amanda Coles is funded by EPSRC Fellowship EP/H029001/1.

References

[1] M.W. Andersen. *Service Network Design and Management in Liner Container Shipping Applications*. PhD thesis, Technical University of Denmark, Department of Transport, 2010.

[2] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Maritime transportation. *Transportation*, 14:189–284, 2007.

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

[4] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, May 2010.

[5] W. Cushing, S. Kambhampati, Mausam, and D. Weld. When is temporal planning *really* temporal planning? In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1852–1859, 2007.

[6] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report No. 195, Institut für Informatik, 2003.

[7] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235–297, 2006.

[8] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[9] J.E. Korsvik, K. Fagerholt, and G. Laporte. A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers & Operations Research*, 38(2):474 – 483, 2011.

[10] B. Løfstedt, J.F. Alvarez, C.E.M. Plum, D. Pisinger, and M.M. Sigurd. An integer programming model and benchmark suite for liner shipping network design. Technical Report 19, DTU Management, 2010.

[11] B.J. Powell and A.N. Perakis. Fleet deployment optimization for liner shipping: An integer programming model. *Maritime Policy and Management*, 24(2):183–192, Spring 1997.

[12] D.E. Smith, J. Frank, and A.K. Jónsson. Bridging the gap between planning and scheduling. *The Knowledge Engineering Review*, 15(1):47–83, 2000.

[13] S. Smith. Is scheduling a solved problem? *Multidisciplinary Scheduling: Theory and Applications*, pages 3–17, 2005.

[14] United Nations Conference on Trade and Development. Review of maritime transport, 2011.