Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the mansucript number!)

Parallel Large-Neighborhood Search Techniques for LNG Inventory Routing

Badrinarayanan Velamur Asokan
ExxonMobil Upstream Research Company, badri.velamur.asokan@exxonmobil.com,

Kevin C. Furman
ExxonMobil Research and Engineering Company, kevin.c.furman@exxonmobil.com,

Vikas Goel
ExxonMobil Upstream Research Company, vikas.goel@exxonmobil.com,

Yufen Shao
ExxonMobil Upstream Research Company, yufen.shao@exxonmobil.com,

Guangye Li
previously at ExxonMobil Upstream Research Company,

Liquefied natural gas (LNG) is estimated to account for a growing portion of the world natural gas trade. For profitable operation of a capital intensive LNG project, it is necessary to optimally design various aspects of the supply chain associated with it. Of particular interest is optimization of ship schedules and the inventories on the production and re-gasification terminals. This can be achieved by modelling the LNG supply chain as a LNG inventory routing problem (LNG-IRP). In spite of significant recent developments in algorithms and heuristics for the LNG-IRP, large problems of practical significance still take a large amount of time to solve. In this paper, we address this issue by proposing several deterministic and non-deterministic parallel large neighborhood search algorithms for solving large LNG-IRPs. We characterize the performance of these algorithms using metrics that describe accuracy, speed-up, and processor utilization. Computational investigation on a test-suite of 11 large LNG-IRPs indicate that these algorithms can obtain solutions of comparable accuracy to state-of-the art serial benchmark solution techniques with appreciable speed-up.

Key words: LNG; Inventory routing; Parallel heuristics; High-performance computing; Ship scheduling

1. Introduction

Liquefied natural gas (LNG) is the preferred mode for transporting natural gas from remote production sites with little local demand to locations with an established natural gas market (M.D.Tusiana and G.Shearer (2007)). It is expected to account for an increasing

share of the total global natural gas trade in the coming years (U.S.DOE (2011)). LNG projects are large capital-intensive undertakings with an underlying supply chain that includes production of natural gas, liquefaction at very low temperatures (-163°C) to LNG, transport of LNG in large ships to markets, re-gasification of LNG, and injection into a pipeline grid. An LNG project links production and re-gasification (regas) facilities and is often bounded by long-term contracts with considerable inter-governmental scrutiny. To operate such a project profitably while satisfying the contractual obligations, it is necessary to optimize the LNG ship schedules and the inventory levels at the production and regas terminals. Mathematical optimization models for these problems can be formulated as LNG inventory routing problems (LNG-IRPs).

LNG-IRPs are a special case of Maritime Inventory Routing Problems (MIRPs) (Christiansen et al. (2007)), (Christiansen et al. (2004)), (Andersson et al. (2012)) with several additional considerations, namely, variable production and consumption rates, contractual obligations, berth constraints at production and regas terminals. Discrete-time-arc-flow and path-flow models for operational inventory routing for a time horizon of up to 60 days were developed in (Gronhaug and Christiansen. (2009)). Branch-and-price methods were developed for the same problem with an increased time-horizon of up to 75 days (Gronhaug et al. (2012)). There have also been developments in discrete-time arc-flow models that consider optimizing decisions on the regas side (Fodstad et al. (2008)), (Fodstad et al. (2011)). An arc-flow model and rolling horizon heuristic was applied to solve large LNG-IRPs with up to 46 ships and a one year planning horizon (Rakke et al. (2011)). A decomposition-based heuristic method was used to solve an arc-flow model with moderate number of ships and a planning time horizon ranging from 30 to 360 days (Halvorsen-Weare and Fagerholt (2010)). While the above consider large scale LNG-IRPs, they limit optimization of ship schedules together with inventory management at the production terminals only.

(Goel et al. (2012)) proposed a variation of LNG-IRP that simultaneously optimizes the ship schedules together with inventory management at the production and regas terminals. Our model is based on a real-world application and shares the fundamental properties of a single-product MIRP (Christiansen and Fagerholt (2009)). The solution to our LNG-IRP includes ship voyages and inventory levels at the terminals over a long time horizon. In order to solve the LNG-IRP, we developed a two-stage algorithm wherein, in the first step, a construction heuristic is used to obtain the initial feasible solution, and in the

next step, an improvement heuristic is used to obtain a solution of better quality. We performed computational experiments with a greedy construction heuristic and variants of two-ship and time-window improvement heuristics. (Shao et al. (2013)) introduced two classes of algorithms for the construction of initial feasible solution, namely, the GRASP algorithm and the Rolling-Time algorithm. In addition, we considered several extensions to the improvement heuristics viz. singletons (1-ship, 1-terminal etc.), two-ship variations, and time-window variations. Computation time for moderate to large problems ranged from several minutes to several hours. To address real world LNG-IRPs, computation time needs to be drastically reduced.

Along with the research progress in solving LNG-IRPs, considerable progress has been made on parallel meta-heuristics (PMH) for the solution of large scale combinatorial problems (Crainic and Toulouse (2009)). PMH enable solution of large problems in a reasonable amount of time. Several successful applications of these techniques in the field of optimization include tabu search (Crainic et al. (2006)), (Crainic et al. (1997)), ant-colony optimization (Janson et al. (2006)), and variable neighborhood search (Moreno-Perez et al. (2006)). Large capacitated vehicle routing problems (CVRPs) have been solved using a parallel tabu search algorithm with multiple search threads, where, each search thread encapsulated a single neighborhood or neighborhood combinations (Jin et al. (2010)). This yielded solutions of comparable quality to that found in literature. An extension to the algorithm with improved computation time was reported in (Jin et al. (2012)). Here, a group of threads were used for diversification (obtaining new starting solutions) and another group of threads were used for intensification (improving existing solutions). In (Perron (2002)) and (Perron (2003)), fast restart policies and a portfolio of algorithms in each thread were used to achieve parallelization of large neighborhood search algorithms for obtaining the best possible solution for telecommunication network design problems within a specified time.

Based on our literature survey, we have not encountered any work on development and characterization of parallel large neighborhood search algorithms for solving LNG-IRPs. In this paper, we will attempt to fill this gap by considering several deterministic and non-deterministic parallel algorithms for solving a LNG-IRP. We will also consider several performance metrics for characterizing the accuracy and the speed-up achieved by these algorithms over a wide class of large LNG-IRPs.

The rest of the paper is organized as follows: A brief description of the LNG-IRP is introduced in section (2). The mathematical description of the optimization model for the LNG-IRP together with associated parameters and sets is presented in section (3). Solution methodology, including the construction heuristics, local neighborhood search operators, and the parallel improvement heuristics are described in section (5). Computational studies on a carefully selected test-suite containing 11 large LNG-IRP instances is then considered in section (7). Finally, we provide the conclusions and future research directions.

2. Problem Description

The LNG-IRP in this paper was developed in Goel et al. (2012). The goal is to find an optimal schedule for a heterogeneous pool of ships delivering LNG from a set of production terminals to regas terminals while satisfying constraints on inventory storage, port operations and contractual obligations. All ships fully load at the production terminal and fully discharge at a regas terminal.

Formally, the problem considers a set of production and regas terminals with fixed storage capacities. While, production terminals have fixed production profiles, regas terminals can adjust their regas rates within specified bounds. Total LNG demand over the planning horizon at each regas terminal is specified. Each terminal has a limited number of berths for loading and unloading activities that span over one time period and occupy one berth for that duration.

Lost production (production in excess of storage capacity), stockout (regas in excess of inventory levels) and unmet contractual demands are penalized in the objective function. Heterogeneous fleets, seasonal travel times, limited berth availability at terminals, seasonality in production rates, and variable regas rates make the LNG-IRP complicated and hard to solve. The reader is referred to Goel et al. (2012) for a detailed discussion of the problem.

3. Mathematical Model for LNG-IRP

Sets and Parameters	
L	Production terminals. $L = \{1, 2, \ldots, \lvert L\rvert\}$
R	Regas terminals. $R = \{1, 2, \ldots, \lvert R\rvert\}$
J	All terminals. $J = L \cup R$

V	Vessels or Ships. $v = \{1, 2, \ldots, \|V\|\}$
T	Planning horizon. $T = \{1, 2, \ldots, \|T\|\}$
c_j	Storage capacity at terminal $j \in J$
$c_{v,j}$	Volume loaded (discharged) by ship v at production (regas) terminal j
D_r	Demand for LNG over planning horizon at regas terminal $r \in R$
b_j	Number of berths at terminal j
$p_{l,t}$	Production rate during time period $t \in T$ at production terminal $l \in L$
$d^L_{r,t}$	Minimum regas rate of regas terminal r
$d^U_{r,t}$	Maximum regas rate of regas terminal r
$w_{j,t}$	Penalty for lost production at production or stockout at regas terminal j
w^D_r	Penalty for unmet demand at regas terminal r
	Network Variables
N	Set of all nodes
$\bar{N}$	Set of regular nodes. $\bar{N} = N \backslash \{SRC, SNK\}$
A^T_v	Set of travel arcs for ship v
A^F_v	Set of arcs from regular nodes to SNK node for ship v
A_v	Set of all arcs for ship v
A	Set of arcs. $A = \cup_v A_v$
δ^+_n	Set of outgoing arcs from node n
δ^-_n	Set of incoming arcs to node n
	Decision Variables
$I_{j,t}$	Inventory level at terminal j at the end of time period t
$d_{r,t}$	Regas rate during time period t at regas terminal r
$o_{l,t}$	Lost production at production terminal l during time period t
$s_{r,t}$	Stockout at regas terminal r during time period t
δ^D_r	Unmet demand at regas terminal r during planning horizon
x_a	Binary variable for arc a

Table 1: Notations used in the MIP model for the LNG-IRP

The LNG-IRP used in this paper was developed in (Goel et al. (2012)) as a mixed-integer program (MIP) based on the time-space network formulation (Song and Furman (2013)),

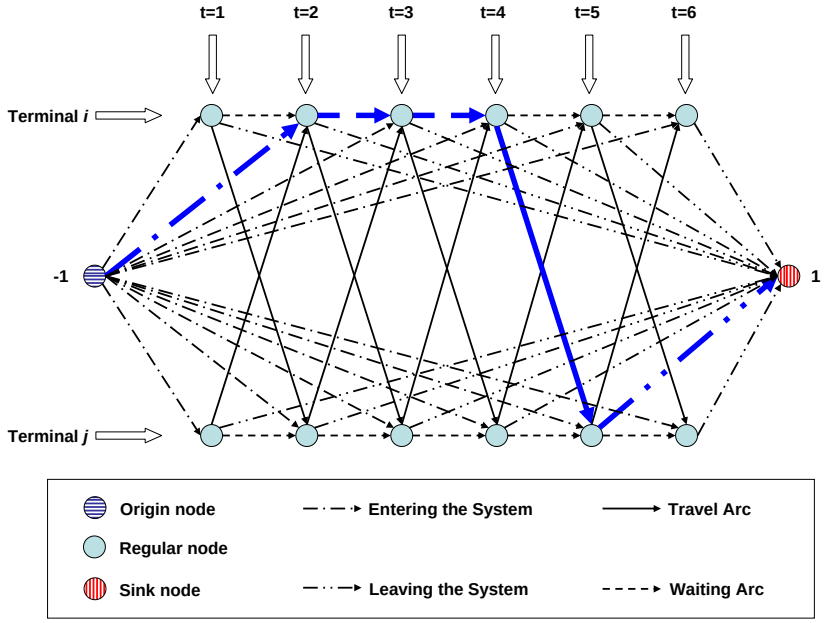


Figure 1 **Example of Time-Space Network Structure. Nodes represent a terminal at a given time.** SRC **and** SNK **represent the initial and final ship locations**

(Furman et al. (2011)). The schedule for a ship is described as a set of arcs. Different types of arcs are described in Figure 1. A feasible solution to the model consists of a set of arcs and regasification rates at each regas terminal. For completeness, we list the MIP model below using the notations in Table 1. For further details, refer (Goel et al. (2012)) and (Shao et al. (2013)).

$$\min \sum_{(l,t)\in\bar{N}|l\in L} w_{l,t}o_{l,t} + \sum_{(r,t)\in\bar{N}|r\in R} w_{r,t}s_{r,t} + \sum_{r\in R} w_r^D\delta_r^D \tag{1}$$

$$s.t. \sum_{a\in A_v\cap\delta_n^+} x_a - \sum_{a\in A_v\cap\delta_n^-} x_a = 0, \qquad \forall v\in V, n\in\bar{N}, \tag{2}$$

$$\sum_{a\in A_v\cap\delta_{SRC}^+} x_a = 1, \qquad \forall v\in V, \tag{3}$$

$$\sum_{a\in A_v\cap\delta_{SNK}^-} x_a = 1, \qquad \forall v\in V, \tag{4}$$

$$I_{l,t} = I_{l,t-1} + p_{l,t} - \sum_v \sum_{a\in(A_v^T\cup A_v^F)\cap\delta_{(l,t)}^+} c_{v,l}x_a - o_{l,t} \qquad \forall l\in L, t\in T \tag{5}$$

$$I_{r,t} = I_{r,t-1} - d_{r,t} + \sum_v \sum_{a\in(A_v^T\cup A_v^F)\cap\delta_{(r,t)}^+} c_{v,r}x_a + s_{r,t} \qquad \forall r\in R, t\in T, \tag{6}$$

$$\sum_v \sum_{a\in(A_v^T\cup A_v^F)\cap\delta_n^+} x_a \le b_j, \qquad \forall j\in J, t\in T, n=(j,t)\in\bar{N} \tag{7}$$

$$\delta_r^D \geq D_r - \sum_t \sum_v \sum_{a \in (A_v^T \cup A_v^F) \cap \delta^+_{(r,t)}} c_{v,r} x_a, \qquad \forall r \in R \tag{8}$$

$$d_{r,t}^L \leq d_{r,t} \leq d_{r,t}^U, \qquad \forall r \in R, \forall t \in T, \tag{9}$$

$$0 \leq I_{j,t} \leq c_j, \qquad \forall j \in J, t \in T \tag{10}$$

$$o_{l,t} \geq 0 \qquad \forall l \in L \tag{11}$$

$$s_{r,t} \geq 0 \qquad \forall r \in R \tag{12}$$

$$\delta_r^D \geq 0 \qquad \forall r \in R \tag{13}$$

$$x_a \in \{0,1\} \qquad \forall a \in A \tag{14}$$

The objective is to minimize the weighted sum of lost production, stockout, and unmet demands. Eqs. (2)-(4) are network-flow conservation constraints for each ship. Eqs. (5) and (6) model the inventory balance. Eq. (7) models berthing constraint at terminals. Eq. (8) enforces a lower bound on the unmet demand variable based on total deliveries scheduled for a regas terminal. Eqs. (9) through (14) are bound constraints.

In (Goel et al. (2012)) and (Shao et al. (2013)), we designed several heuristics for the solution of the LNG-IRP since commercial MIP solvers could not solve the problem within a reasonable amount of time for cases of interest. We will now present techniques for parallelization of these heuristics.

4. Background Concepts

Here, we present background concepts relevant to the rest of this paper. Consider the following notations:

Let N be the number of processors used, n be an integer index associated with each processor. n typically ranges between 0 and $N-1$ and is referred to as processor rank.

4.1. Parallel Efficiency

The total solution time for a parallel algorithm executing on many processors can be divided into three parts in each processor: computation time t_{cn}, idle time t_{in}, and inter-processor communication time t_{xn}. Parallel efficiency is a metric quantifying overall fraction of time spent by the algorithm in computation.

$$PE_{\text{overall}} = \frac{1}{N} \sum_{n=0}^{N} PE_n; \qquad \text{where} \quad PE_n = \frac{t_{cn}}{t_{cn} + t_{in} + t_{xn}}, \tag{15}$$

PE values depend on the algorithm as well as the implementation. When comparing algorithms with similar accuracy and solution times, the algorithm with higher PE is preferred since it achieves better utilization of the computational resources which directly impacts cost.

For a given algorithm, the distribution of PE_n provides insights into the implementation: For e.g. if PE_n values vary wildly, it can be due to load-balancing issues, failure of communication channels, disk input-output problems etc.

Though there are a few practical algorithms where the overall PE equals one e.g. Monte-Carlo algorithm, majority of the parallel algorithms yield a PE less than one.

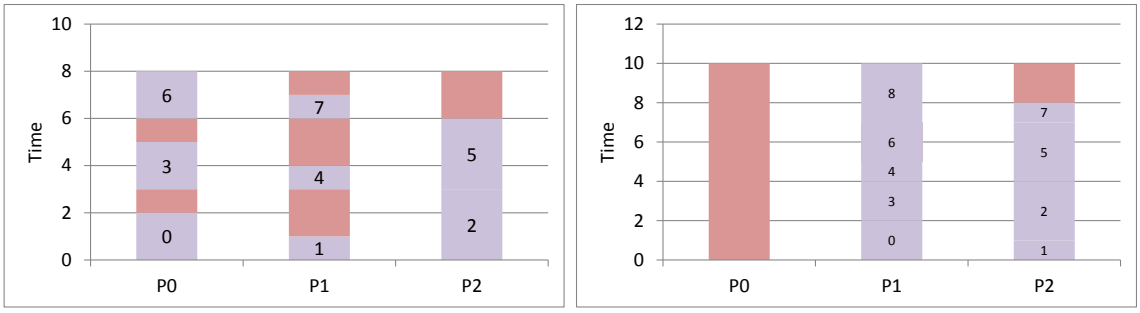
4.2. Static and Dynamic Resource Allocation

In parallel algorithm design, we often have to solve the problem of load balancing i.e. allocating P tasks with varying computational requirements to be solved among N processors, where P is typically much larger than N. Static and dynamic resource allocation provide two ways to addressing this problem.

In static allocation, the number of tasks and the order of tasks to be completed in each processor is decided beforehand. This is preferred when designing deterministic parallel algorithms wherein, the performance of the algorithm and its accuracy are constant for a given hardware configuration.

In dynamic allocation, the number of tasks and the order of tasks to be completed in each processor is decided during the run-time. The performance of an algorithm is thus affected by the hardware configuration, congestion in the inter-processor communication networks, and the computational requirements of each task. Inherently, dynamic resource allocation yields non-deterministic parallel algorithms. Typically, we consider non-deterministic algorithms only for increased speed for the same accuracy.

Choice of static vs. dynamic resource allocation could have significant impact on the parallel efficiency of an algorithm. This is illustrated using figures (2a) and (2b), where, static and dynamic allocation are used to allocate eight tasks to three processors with an algorithm that requires processors to synchronize after completing each of their tasks. The x-axis denotes the processor, and the y-axis is the total computation time. Each task is shown as a numbered box and the idle time is shown as a shaded box. For static allocation, tasks 1, 4, and 7 complete early and processor P1 waits for processors P2 and P3 to complete their tasks, leading to low parallel efficiency. In dynamic resource allocation (here,



(a) Static resource allocation (b) Dynamic resource allocation

Figure 2 Difference between static and dynamic resource allocation.

the manager-worker model), one processor acts as a manager. As soon one of the other processors (worker) completes its task, the manager allocates a new one to it. Except the manager P0 that does not compute, all other processors are fully utilized.

Dynamic resource allocation is often necessitated by heterogeneous computing resources i.e. processors with different speeds, variations in operating in inter-processor communication network speeds etc.

4.3. Manager-Worker Model: Implementing dynamic resource allocation

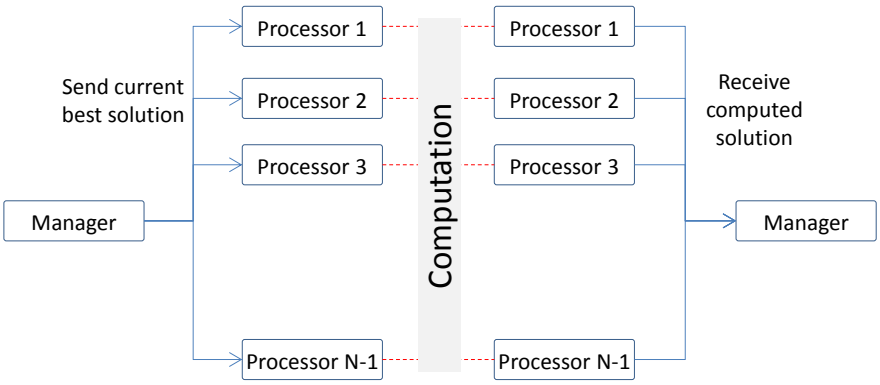


Figure 3 Schematic of the manager-worker model.

The manager-worker model described in Figure 3 is one of the simplest and widely used methods to implement dynamic resource allocation. Here, one of the processors acts as a manager (typically rank 0) and the remaining processors act as workers. The manager keeps track of tasks completed, task assignment to workers, and the tasks remaining. The workers are simple. They know how to complete their allocated task and to communicate with the manager.

In a typical manager-worker algorithm, the manager starts off with allocating a task to each of the workers. As soon as a worker completes a task, the manager allocates the next available task to that worker. If one of the workers gets a time-intensive task, others can independently continue their computations without waiting.

Under this model, manager does not take part in computation and the theoretical best overall PE is $(N-1)/N$. However, as we increase the number of processors, more and more workers try to communicate with the manager, whereby increasing inter-processor communication time nonlinearly and decreasing PE.

4.4. Strategies for solution of combinatorial problems

Local search heuristics for the solution of combinatorial problems typically use a combination of three strategies: Intensification, Diversification, and Randomization (Crainic et al. (2006)) and (Crainic and Toulouse (2009)). In intensification, we explore parts of the problem (network) where we assign a high probability of finding good solutions. In diversification, we explore different parts of the problem (network) in the hope of finding new areas with good solutions, and in randomization, we randomly perturb the solution to get out of local minima.

5. Solution Methodology

We use a two-step solution strategy analogus to the approach employed by (Goel et al. (2012)) and (Shao et al. (2013)). In the first step, a construction heuristic is used to build an initial feasible solution which acts as the starting solution for the second step, where, a parallel improvement heuristic iteratively improves the solution quality by searching in neighborhoods defined by one or more local search operators.

To define a solution scheme, we need the construction heuristic, the local neighborhood search operators embedded in the parallel improvement heuristic, and the definition of the parallel improvement heuristic. In addition, we need to define the number of processors the algorithm runs on.

5.1. Construction Heuristic

We will use two construction heuristics: Naïve construction heuristic, and One-day Rolling-Time construction heuristic referred to as RT0 by (Shao et al. (2013)).

5.1.1. Näive Construction Heuristic This heuristic yields a highly simplistic initial feasible solution, where none of the ships participate in any voyage and none of the LNG produced reaches the regas terminals. This allows us to study the performance of the parallel improvement heuristic in the absence of a good starting solution. This heuristic can serve as the default approach for complicated LNG-IRP formulations with complex side constraints where we might not be able to develop a better construction heuristic.

5.1.2. One-Day Round Trip Rolling-Time Construction Heuristic (RT0) We will provide a brief description of this algorithm. For details, refer to (Shao et al. (2013))., Here, a feasible solution is constructed by solving a sequence of optimization problems with a focus on the production terminals. We attempt to schedule a round-trip voyage starting at the production terminals for each available ship on a particular day. The round trip voyage could be achieved by one of two ways: Waiting at a production terminal, or by visiting a regas terminal and returning. The maximum number of days each ship is allowed wait at a regas terminal is a key parameter.

By construction, RT0 is a fast, sequential algorithm that offers no efficient avenues for parallelization. Since the time taken to construct a solution is a small fraction of the total computation time, the sequential nature of the construction step does not affect performance metrics appreciably.

5.2. Local Neighborhood Search Operators

The parallel algorithms considered in this paper are essentially parallel meta-heuristics embedded with local neighborhood search operators. In particular, we consider two local neighborhood search operators, namely, the two-ship operator (Song and Furman (2013)) and the time-window operator (Goel et al. (2012)). By the nature of embeddings, the parallel improvement heuristics are divided into two classes: simple (embeds a two-ship operator), and hybrid (embeds a combination of two-ship and time-window operators such that all two ship sub-problems are solved first followed by time-window sub-problems). We stress that nothing in the description of our parallel meta-heuristics precludes embedding multiple operators. Each of these operators define an associated sub-problem. The solution of a sub-problem involves starting with an incumbent solution, solving the sub-problem using a commercial MIP solver, and updating the incumbent solution.

5.2.1. Two-Ship Local Neighborhood Search Operator The search neighborhood is defined based on a pair of selected ships. It consists of the set of feasible schedules for these ships. Schedules for all other remaining ships are fixed. A two-ship sub-problem is then defined and solved in order to optimize the schedules within this neighborhood. By construction, for a given number of ships $|V|$, there are ${}^{|V|}C_2$ two-ship neighborhoods that yield an equal number of two-ship sub-problems.

5.2.2. Time-Window Local Neighborhood Search Operator This search neighborhood is constructed based on time windows each of which is defined as a sequence of consecutive days. In the associated sub-problem, arcs are freed up if its start time or end time is within the time window; otherwise, arcs are fixed to their values in the current solution. A sequence of different time window neighborhoods can be generated and solved depending on the beginning time and width of the time widows. Based on the number of days in a time window and the overlap allowed between two successive time windows, several sub-problems can be defined.

5.3. Nomenclature for Improvement Step

We shall consider the following notations while describing the heuristics involved in the improvement step of our solution strategy:

- P is the set of optimization sub-problems defined by the embedded local neighborhood search operators. A two-ship operator results in $|P| = {}^{|V|}C_2$, and a combination of two-ship and time-window operators results in $|P| = {}^{|V|}C_2 + P_{TW}$, where, $|V|$ is the number of ships and P_{TW} is the number of time-windows.
- $\mathcal{A}$ is a re-ordering or permutation of the set P. All heuristics involved in the improvement step take an initial ordering as an input. $\mathcal{A}$ has all the information contained in P with $|\mathcal{A}| = |P|$.
- A major iteration is completed when all optimization sub-problems in P are solved once. The improvement step can span multiple major iterations.
- x_0 is the starting solution for a major iteration. At the start of the improvement step, x_0 will be equal to the solution obtained at the end of the construction step.
- Improvement step terminates when the solution objective value is unchanged for an entire major iteration.
- We use the terminology $x_{\text{new}} = \text{Solve}(i,\mathcal{A},x_{\text{old}})$ as a short form for "Solve the i-th sub-problem in the ordering $\mathcal{A}$ starting from x_{old} to obtain an updated solution x_{new}".

5.4. Serial Solution Algorithm

Algorithm 1 Benchmark Serial Solution Algorithm ($\mathcal{A}$, x_0)

1: **for** i = 0 to $\mathcal{A}$-1 **do**
2: x_{i+1} = Solve(i,$\mathcal{A}$,x_i)
3: **end for**
4: Return x_{i+1}.

Algorithm 1 describes one major iteration of the benchmark serial improvement step. The ordering used has a significant effect on the final solution quality. Several orderings including lexicographic, LP-based, and heuristic orderings were considered in (Goel et al. (2012)).

5.5. Parallel Improvement Heuristics

For designing parallel improvement heuristics, we adopt the non-intrusive method that involves load balancing i.e. allocation of optimization sub-problems in $\mathcal{A}$ with appropriate starting solutions among N processors. This choice enables us to use commercial MIP solvers to solve the local search problems (individual optimization sub-problems) while parallelizing the entire improvement step. We also specify rules for inter-processor communication. These rules define the sharing of solutions among processors and the communication structure between the processors.

In order for load balancing to be efficient, we require the number of optimization sub-problems to be sufficiently large in comparison to the number of processors. This constrains the kind of local search operators we can use. In this paper, we limit ourselves to two-ship and time-window searches. Parallelization of other local search operators discussed in (Shao et al. (2013)) require further research.

We will now describe several parallel meta-heuristics for the improvement step of the solution process. These meta-heuristics can be thought of as a parameterized family of parallel algorithms. By tuning these parameters through a comprehensive performance analysis, we can obtain highly efficient parallel algorithms. The meta-heuristics we will describe can be broadly classified into two categories based on solution repeatability:

Parallel deterministic meta-heuristic: We can achieve repeatability in the accuracy and performance of the algorithm for a given test problem on a given hardware configuration. Algorithms in this category typically use static resource allocation.

Parallel non-deterministic meta-heuristic: For the same test problem and hardware configuration, accuracy and performance of the algorithm could change for different runs of the algorithm based on the load conditions on the cluster of processors and algorithm parameters. Algorithms in this category typically use a combination of randomization and dynamic resource allocation.

In practice, solution repeatability is highly desirable. Non-deterministic algorithms are pursued only to uncover potentially better solutions or to obtain a much better speed-up. We will now present the following meta-heuristics:

1. Parallel deterministic meta-heuristic
2. Parallel non-deterministic meta-heuristic
3. Parallel acceptance-rejection meta-heuristic, and
4. Parallel store-K solutions meta-heuristic

Algorithm 2 Parallel Deterministic Meta-Heuristic(N, $\mathcal{A}$, M, x_0)

Get $\mathcal{A}_k$ such that $\cup_{k=0}^{N-1}\mathcal{A}_k = \mathcal{A}$ such that $\mathcal{A}_k \cap \mathcal{A}_{k'} = \emptyset$ for $k \neq k'$
For current processor with rank r, get $\mathcal{A}_{rm}$ such that $\cup_{m=0}^{M-1}\mathcal{A}_{rm} = \mathcal{A}_r$
$\hat{x} = x_0$
for $m = 0$ to $M-1$ **do**
    $\hat{x}_0 = \hat{x}$
    for $i = 0$ to $|\mathcal{A}_{rm}| - 1$ **do**
        $\hat{x}_{i+1}$ = solve(i,$\mathcal{A}_{rm}$,$\hat{x}_i$)
    end for
    $\hat{x} = \hat{x}_{i+1}$ with the smallest objective value among all processors
end for
Return $\hat{x}$

5.5.1. Parallel Deterministic Meta-Heuristic The sequence of steps for one major iteration using the parallel deterministic meta-heuristic is described in Algorithm 2. The solution at the end of one major iteration acts as the starting solution x_0 for the next major iteration. If the solution objective value does not decrease in a major iteration, the solution is assumed to have converged and the improvement step in the solution process terminates.

In addition to the global parameters N, $\mathcal{A}$, and x_0; this algorithm uses an additional parameter M that denotes the number of times the processors synchronize during one major iteration. In the first two steps of this algorithm, we decide the order in which sub-problems are solved in each processor. This corresponds to a static resource allocation (see section 4.2). By tuning M, we can control the extent of inter-processor communication with $M = 1$ yielding the least communication.

5.5.2. Parallel Non-deterministic Meta-heuristic The parallel non-deterministic meta-heuristic comprises of a manager algorithm and a worker algorithm. One major iteration of the heuristic is described in Algorithm 3. The manager algorithm is typically executed in the processor with rank 0, and the worker algorithm is executed in the remaining processors. Though we provide an initial ordering of sub-problems, the order they are solved in each processor is decided at run-time according to the load conditions on the parallel cluster hardware. The solution quality obtained by the algorithm might not be repeatable across multiple runs unless the load conditions are the same for the parallel cluster hardware. This is rarely the case in practice.

We introduce randomization into Algorithm 3 by considering two algorithmic extensions, namely, parallel acceptance-rejection, and parallel store-K solutions.

5.5.3. Parallel Acceptance-Rejection Meta-heuristic In the parallel non-deterministic meta-heuristic, the worker obtains a sub-problem and an incumbent solution from the manager. It always uses this solution as the starting point for solving the sub-problem. This is a greedy strategy that results in rapid solution intensification typically coupled with sub-optimal final solution quality.

In the acceptance-rejection algorithm, we modify the worker algorithm. The worker sends its solution to the manager and stores a copy of the solution locally. When the worker receives a sub-problem and an incumbent solution from the manager, it makes a choice to continue with the local solution or to accept the solution sent from the manager. This choice is made randomly based on a probability criterion that compares relative solution quality of the incumbent solution and the locally stored solution. By tuning the probability criterion, we can ensure that the worker starts with a sub-optimal solution every once in a while. This avoids the rapid intensification typically seen in Algorithm 3.

The sequence of steps for the modified worker procedure is described in Algorithm 4. The manager procedure described in Algorithm 3 remains unchanged.

5.5.4. Parallel Store K Solutions Meta-heuristic This meta-heuristic is the counterpart for parallel acceptance-rejection where the worker was allowed to choose between the incumbent solution from the manager and a locally stored solution. Here, the manager holds a solution pool containing the K-best solutions at any given point in time. It chooses one of these solutions using a probabilistic criterion and sends it to the workers. This introduces randomization and diversification in the solution process. Here, the manager procedure is described in Algorithm 5 and the worker procedure remains that same as described in Algorithm 3.

6. Algorithm Implementation

In this section, we discuss the implementation of the algorithms presented in section (5.5) along with other computational considerations.

6.1. Obtaining sub-problem ordering for processors from $\mathcal{A}$

All the improvement heuristics presented in section (5.5) take a global ordered set $\mathcal{A}$ of optimization sub-problems as the input. (Goel et al. (2012)) showed that this ordering has a significant effect on the convergence of the serial benchmark algorithm and the final solution quality. For this paper, we always consider a lexicographic ordering of sub-problems for $\mathcal{A}$.

In the parallel deterministic meta-heuristic, we further need to sub-divide $\mathcal{A}$ into individual processor sub-problem orderings. This is done in two ways: using a sequential scheme, and using a round-robin scheme.

In the sequential scheme, the first $\lceil |\mathcal{A}|/N \rceil$ problems from $\mathcal{A}$ are allocated to processor with rank 0, the next $\lceil |\mathcal{A}|/N \rceil$ to processor with rank 1, and so on till all the sub-problems have been allocated. If $|\mathcal{A}| = 7$ and $N = 2$; the first four sub-problems will be allocated to processor with rank 0 and the rest will be allocated to the processor with rank 1.

In the round-robin scheme, the first sub-problem from $\mathcal{A}$ is allocated to processor with rank 0, the second sub-problem to processor with rank 1, ..., the $N-1$-th sub-problem to processor with rank $N-1$. Then the N-th sub-problem to processor with rank 0 and so on till all the sub-problems have been allocated.

Based on initial studies, we found that the round-robin scheme provides the best convergence characteristics and final solution quality. We thus, use the round-robin scheme for the parallel deterministic meta-heuristic.

6.2. Local Search Operator Embeddings

Based on the kinds of local search operators embedded, we consider two modes for all algorithms constructed from the meta-heuristics: Plain and Hybrid. In the plain mode, we use only the two-ship search operator and in the hybrid mode, we use a combination of the two-ship and time-window search operators. The ordering for the hybrid mode comprises of all the two-ship sub-problems followed by all the time-window sub-problems. Both the modes use an initial sub-problem ordering based on the round-robin approach.

6.3. Algorithms designed as specific variants of heuristics

Algorithm	Description
PDX	Parallel deterministic meta-heuristic with processor synchronization after every sub-problem solve.
PD1	Parallel deterministic meta-heuristic with M=1.
PD3	Parallel deterministic meta-heuristic with M=3.
PND	Parallel non-deterministic meta-heuristic.
PAR	Parallel acceptance-rejection meta-heuristic.
PS2	Parallel store-K solutions meta-heuristic with K=2.
PS4	Parallel store-K solutions meta-heuristic with K=4.
PS6	Parallel store-K solutions meta-heuristic with K=6.
PS8	Parallel store-K solutions meta-heuristic with K=8.

Table 2 Summary of 9 plain mode parallel improvement algorithms

We consider 9 algorithms in plain and the hybrid mode to obtain a total of 18 algorithms. The plain mode algorithms are summarized in 2. The letter "P" at the start of each algorithm denotes Plain mode. Of these PDX, PD1, and PD3 are deterministic parallel algorithms. PDX has the maximum amount of inter-processor communication. The processors communicate the least in PD1 (once every major iteration). The amount of randomization introduced in the solution process increases as we move from PS2 to PS8. The counterparts for these algorithms in the hybrid mode are: HDX, HD1, HD3, HND, HAR, HS2, HS4, HS6, and HS8.

The algorithms have been implemented in C++ with MPI (Message Passing Interface) to handle inter-processor communication. Commercial solver CPLEX version 10.4.1 was

used for solving the optimization sub-problems resulting during the construction step and the improvement steps. All test cases were tested on a parallel cluster with 2x3.2 Ghz Intel xeon X5460 processors and 48 GB RAM per node.

7. Computational Results

We will test the accuracy and performance of our algorithms against a set of 11, large LNG-IRPs constructed using realistic field data. The solution time for these cases using the serial benchmark algorithm is typically more than 10 hours. For practical reasons, it is desirable to reduce the solution wall-clock time using parallel algorithms.

Key parameters for these test cases are provided in Table 3. This includes the number of production terminals ($|L|$), regasification terminals ($|R|$), ships ($|V|$), continuous variables, binary variables, and constraints. The table also contains the final objective function value obtained by the serial benchmark Algorithm 1 while using the Näive and the RT0 construction heuristic. $P13$ and $P14$ have been investigated in (Goel et al. (2012)) and (Shao et al. (2013)).

In order to construct the time-space network for each of these test cases, we use a planning horizon of 365 days with a time discretization of one day. The guidelines provided in (Jackson et al. (1990)) lists three factors that contribute to solution time: algorithm, implementation, and hardware configuration. In order to compare algorithms, the other two factors need to be fixed. All the test cases are run on the same parallel cluster. Several aspects of the algorithm implementation are re-used through a common object-oriented code base.

Problem	Dimensions	Variables		Constraints	Serial benchmark	
	$(\|L\|, \|R\|, \|V\|)$	Continuous	0-1		Näive	RT0
P13	(1,8,40)	9987	136351	48633	1441	1453
P14	(1,10,69)	12268	240702	81908	50826	53240
P16	(2,5,29)	6599	83094	30807	1043	688
P17	(1,13,56)	15399	329359	104435	3453	3575
P18	(1,13,56)	15399	1086624	297317	3478	3454
P20	(2,18,100)	20899	488079	156500	5366	5360
P21	(2,18,100)	20899	1785748	486998	5634	5634
P22	(2,8,32)	9899	144216	49116	2221	2098
P23	(2,8,32)	9899	391712	112068	2244	2245
P24	(1,9,42)	10999	184303	61947	2434	2156
P25	(1,9,42)	10999	284695	87567	2077	2140

Table 3 Description of test cases

Traditionally, speed-up and scalability are the metrics of interest when comparing a parallel and a serial algorithm. They tacitly assume that the parallel algorithm solutions are repeatable and are comparable in accuracy with the serial algorithm solution. Under these assumptions, speed-up is defined as the ratio of time taken to convergence by the serial algorithm to the time taken by the parallel algorithm. Scalability measures how well a parallel algorithm scales with the number of processors. If the clock-time to solution for a parallel algorithm is inversely proportional to the number of processors, it is termed "linearly-scalable". In this paper, we are more interested in reducing the wall-clock time for solution. Scalability will be considered for future research.

Reporting computational results for parallel algorithms for combinatorial problems pose several challenges: a) the solutions from these algorithms could be different from the serial algorithm solutions. a) solutions for different runs of the same test case could be different for algorithms that incorporate randomization. There are different views on the appropriate definition of speed-up and algorithm efficiency for these problems (Barr and Hickman (1993)).

In this paper, we present a procedural approach for performance analysis and ranking of our parallel algorithms by defining several complementary metrics.

7.1. Performance metrics

We will use the notations described in Table 4 for describing the performance metrics. These metrics are designed such that higher values are better. They are also inherently amenable to statistical analysis.

Percentage of Improved Cases (PIC): This metric quantifies whether the accuracy of a parallel algorithm on average, is comparable to the serial benchmark.

Percentage Improvement (PI): In comparison with the serial solution, this metric quantifies the extent of improvement or worsening in solution quality for the parallel algorithm.

Speed-up to Cut-off (SC): This is defined as the ratio of time taken by the serial to the time taken by the parallel algorithm in reaching a cut-off solution quality.

Speed-up to Termination (ST): This is defined as the ratio of time taken by the serial to the time taken by the parallel algorithm to run to termination.

When required, these metrics can be supported with further histogram analysis. We will also present parallel efficiency studies for a subset of the algorithms. These provide insight into the algorithm implementation and use of the parallel cluster resources.

M_{TC}	Number of test cases indexed by $m = 1, \ldots, M_{\text{TC}}$. Here, $M_{\text{TC}} = 11$.
R_m	Number of runs of the parallel algorithm for the m-th test case. This is algorithm dependent. Here, $R_m = 1$ for deterministic algorithms and $R_m = 10$ for non-deterministic algorithms.
r	For each m, $r = 1, \ldots, R_m$.
$O_{rm}^{(p)}$	Final objective value obtained by the r-th run of parallel algorithm on the m-th test case.
$O_m^{(s)}$	Final objective value obtained by the serial benchmark algorithm for the m-th test case.
$t_{rm}^{(p)}$	Time taken by the r-th run of parallel algorithm for the m-th test case to yield a solution with objective value $\max(O_{rm}^{(p)}, O_m^{(s)})$.
$t_{rm}^{(s)}$	Time taken by the serial benchmark algorithm to yield a solution for the m-th test case with objective value $\max(O_{rm}^{(p)}, O_m^{(s)})$.
$T_{rm}^{(p)}$	Time taken by the r-th run of parallel algorithm for the m-th test case to termination.
$T_m^{(s)}$	Time taken by the serial benchmark algorithm to termination.

Table 4 Notations used in describing performance metrics

Scenario	Number of Processors	Construction Heuristic
1	8	Näive
2	32	Näive
3	8	RT0
4	32	RT0

Table 5 Scenarios for performance analysis of parallel algorithms

7.2. Ranking of algorithms

We identified four scenarios for testing the most common construction heuristic and number of processor combinations in Table 5. To rank algorithms by a given performance metric, we choose a scenario and run all algorithms over all the test cases. We then rank the algorithms in the descending order of the metric. This process is repeated for all four scenarios. The algorithms are then ranked based on their individual ranks for the scenarios.

For performance analysis, we rank the algorithms according to PIC metric. This helps us prune the algorithms that are not sufficiently accurate in comparison with the serial benchmark. We then rank the algorithms according to the PI metric. This identifies the

algorithms that provide the best average solution improvement. We then rank the algorithms according to SC and ST metrics.

In practice, it is sufficient to only consider the algorithms with high PIC and PI for speed-up analysis. We will however present the results for all the algorithms for completeness.

7.3. Accuracy

In this section, we will answer the questions:

1. Does a parallel algorithm yield solutions of comparable quality to those obtained from the serial benchmark Algorithm 1?

2. By how much does the parallel algorithm improve or worsen the solution quality on an average?

To this end, we consider the percentage of improved cases (PIC) metric and the percentage improvement (PI) metric.

	Näive		RT0	
	8 proc.	32 proc.	8 proc.	32 proc.
HND	64.55	69.09	70.00	73.33
HD1	63.64	72.73	66.67	66.67
HD3	63.64	81.82	55.56	66.67
HAR	61.82	67.27	66.67	54.44
HS4	62.73	60.00	58.89	55.56
HS8	63.64	56.36	57.78	58.89
PD1	54.55	63.64	66.67	33.33
HDX	54.55	45.45	66.67	66.67
HS2	61.82	53.64	63.33	46.67
HS6	59.09	50.91	61.11	52.22
PND	48.18	55.45	54.44	55.56
PD3	54.55	63.64	33.33	55.56
PAR	50.00	54.55	48.89	50.00
PS8	41.82	52.73	40.00	31.11
PDX	18.18	36.36	55.56	33.33
PS4	45.45	43.64	43.33	27.78
PS2	39.09	43.64	38.89	31.11
PS6	41.82	40.91	37.78	30.00

Table 6 Algorithms ranked according to the PIC metric.

7.3.1. Percentage of Improved Cases (PIC) Using notations in Table 4, PIC is defined as follows:

$$PIC = 100\frac{\sum_{m=1_{\text{TC}}^{M}} \sum_{r=1}^{R_m} I_{mr}}{\sum_{m=1}^{M} R_m}\% \tag{16}$$

$$I_{mr} = \begin{cases} 1\ , O_{mr}^{(p)} \leq O_m^{(s)} + \max(1, \epsilon O_m^{(s)}) \\ 0\ , \qquad\qquad \text{otherwise} \end{cases} \tag{17}$$

I_{mr} is an indicator function that takes the value 1 if the objective value obtained by the parallel algorithm is within a tolerance of the serial algorithm objective value. The parameter ϵ (here 10^{-3}) defines this tolerance.

Since the solutions obtained are different for the parallel and the serial algorithm, we seek parallel algorithms where the solutions obtained are of better quality for at least half the runs. This corresponds to a PIC of 50%. Any parallel algorithm with a PIC well below 50% can be considered inaccurate for all practical purposes. Based on the results in Table 6, we can divide the algorithms into three categories:

1. PIC for all 4 scenarios is greater than 50%: HND, HD1, HD3, HAR, HS4, HS8, and HS6.
2. PIC for one scenario is lesser than 50%: PD1, HDX, HS2, PND, PD3, and PAR.
3. PIC for two or more scenarios is lesser than 50%: PS8, PDX, PS4, PS2, and PS6.

Among these, the first category represents the most desirable algorithms. The second category might be desirable based on overall performance of the algorithm and the statistical significance of the PIC value that is less than 50%. For example, PD1 had a PIC of 33.33% for the fourth scenario (32 processors, RT0) whereas, PND has a PIC of 48.18% for the first scenario (8 processors, Näive). Based on its PI metric value and speed-up, we can select PND ahead of PD1 since its PIC is close to the PIC cut-off of 50%. Here, we select any algorithm from the second category if all PICs are above 45%. As the number of test cases increase, we can tighten this condition further. We now have seven algorithms that satisfy the accuracy requirements based on PIC. They are HND, HD1, HD3, HAR, HS4, HS8, and HS6. In addition, we can provisionally include HDX, HS2, PND, and PAR.

Among hybrid mode algorithms, HD1 and HD3 have higher PICs than HDX. As discussed in section (5.5.1), In HDX, the processors share their solutions after solving each sub-problem. They then use the common best solution as the incumbent for solving the next sub-problem. This leads to rapid solution intensification and in-turn, sub-optimal solution quality.

In HD1, the processors share their solutions at the end of each major iteration. A different set of sub-problems is solved in each processor, whereby, introducing diversification in

the solution process. This typically improves the final solution quality. The amount of inter-processor communication in HD3 is in between that of HD1 and HDX. HND is an intensifying algorithm by construction. The manager always sends its current best solution to the workers that in-turn use it as the incumbent solution for solving the next sub-problem.

The ranking scheme introduced in section (7.2) is to be considered as a guideline for algorithm selection. It is however, not to be solely relied on. For e.g. HS6 is ranked below PD1, HDX, and HS2. However, HS6 has PIC values for all four scenarios above 50%. Also, the ranking scheme tends to be forgiving of bad accuracy in one scenario. Though PD1 has a PIC of 33.33% for the fourth scenario, its high PIC values for the other three scenarios yield it a higher rank.

Algorithms selected - Based on PIC, we select the following deterministic algorithms (HD1, HD3, and HDX) and the following non-deterministic algorithms (HND, HAR, HS4, HS8, HS6, HS2, PND, and PAR).

	Näive						RT0					
	8 proc.			32 proc.			8 proc.			32 proc.		
	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX
HD1	-1.5	4.8	25.3	-5.4	5.2	27.8	-4.1	1.4	7.3	-2.8	1.2	7.5
HD3	-8.0	3.1	15.0	-0.8	4.2	13.9	-8.7	0.7	7.5	-7.2	0.9	7.5
PD1	-5.9	2.7	24.5	-4.4	4.1	24.0	-1.2	1.3	7.5	-3.7	0.4	7.5
HDX	-8.7	3.1	22.9	-2.5	2.0	11.7	-2.4	1.1	7.5	-5.1	0.9	7.5
HND	-9.7	2.2	27.8	-11.6	2.8	26.7	-22.9	-0.1	7.5	-8.2	1.0	7.5
HAR	-14.9	1.4	27.7	-14.4	2.4	25.8	-22.9	0.0	7.5	-8.2	0.7	7.5
HS8	-9.0	2.4	28.1	-11.1	2.3	27.9	-21.0	-0.3	7.5	-12.3	0.3	7.5
HS4	-10.5	2.2	27.0	-11.3	2.3	31.2	-23.7	-0.2	7.5	-31.1	-0.4	7.5
HS6	-15.2	1.8	33.0	-8.4	1.6	29.3	-21.0	0.1	7.5	-22.8	-0.4	7.5
HS2	-11.9	2.1	28.1	-12.2	1.9	25.8	-23.7	-0.2	7.5	-21.3	-0.5	7.5
PND	-22.2	-0.6	24.9	-16.6	0.1	31.3	-22.7	-1.1	7.5	-11.1	0.3	7.5
PDX	-6.2	-1.2	7.2	-6.7	-0.6	13.8	-2.4	0.5	7.5	-5.9	0.0	7.5
PAR	-47.1	-0.7	21.6	-20.5	0.4	34.6	-20.6	-0.7	7.5	-21.8	-0.4	7.5
PD3	-19.3	0.9	22.3	-19.4	-1.2	4.7	-19.8	-2.4	7.5	-11.3	0.0	7.5
PS2	-18.4	-1.3	14.2	-17.6	-0.4	26.1	-20.8	-1.6	7.5	-20.2	-0.9	7.5
PS8	-41.5	-1.4	25.4	-16.0	0.8	27.4	-20.8	-2.6	7.5	-25.3	-1.2	7.5
PS4	-24.7	-0.8	23.8	-27.5	-0.6	15.2	-20.8	-1.6	7.5	-28.1	-1.5	7.5
PS6	-23.4	-1.5	26.6	-22.0	-0.3	30.7	-20.8	-1.8	7.5	-30.3	-1.0	7.5

Table 7 Algorithms ranked according to PI metric.

7.3.2. Percentage improvement (PI) Based on notations in Table 4, PI can be defined as follows:

$$\text{IMP}_{mr} = 100\frac{O_m^{(s)} - O_{mr}^{(p)}}{O_m^{(s)}}\% \tag{18}$$

$$PI = \frac{\sum_{m=1_{\text{TC}}^{M}} \sum_{r=1}^{R_m} \text{IMP}_{mr}}{\sum_{m=1}^{M} R_m} \tag{19}$$

$$PI_{\min} = \min(\text{IMP}_{mr} \ \forall \ m, r) \tag{20}$$

$$PI_{\max} = \max(\text{IMP}_{mr} \ \forall \ m, r) \tag{21}$$

PI quantifies the extent by which solutions obtained by a parallel algorithm are better than the serial benchmark. By construction, for two algorithms with the same PIC, the one with higher PI is preferred. From eq (18), a positive PI indicates average improvement and a negative PI indicates average worsening of the solution.

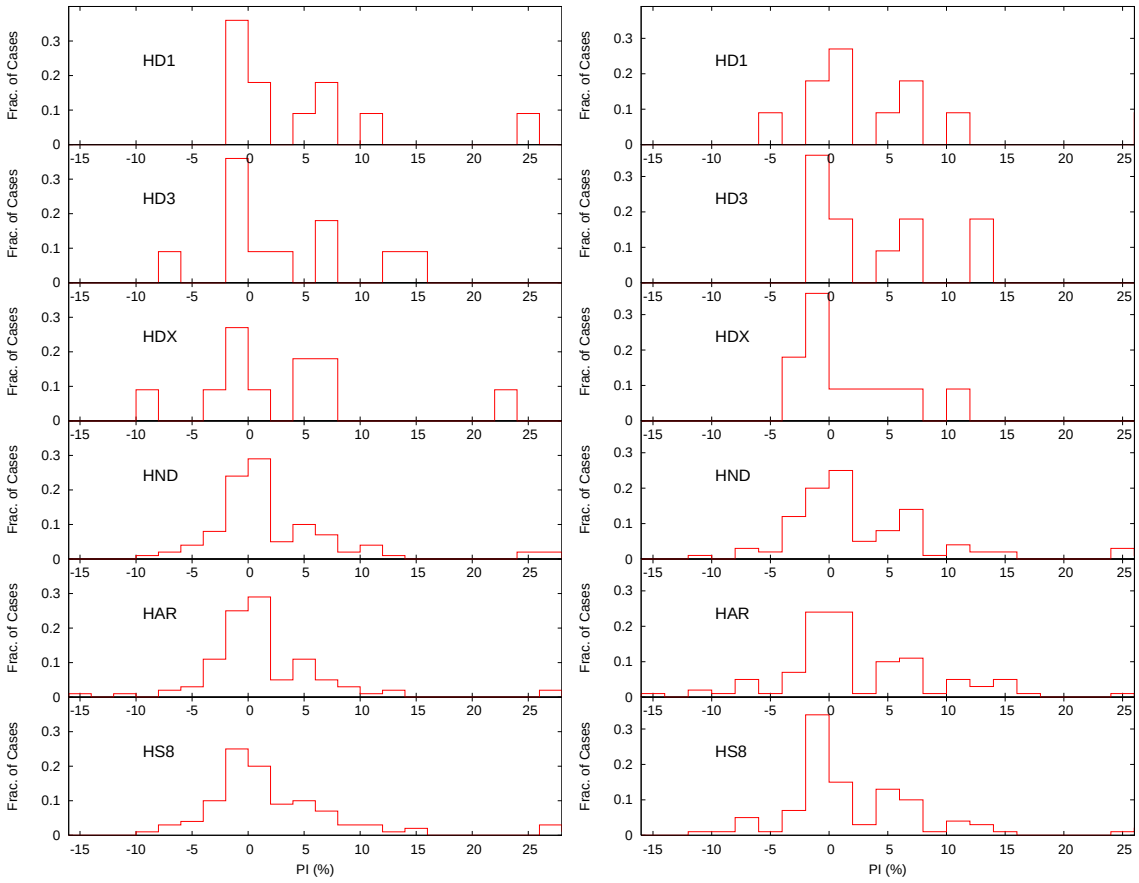
The average, minimum, and maximum PIs are provided for all the algorithms in Table 7 and the algorithms are ranked using the scheme described in section (7.2). Among the algorithms selected in section (7.3.1), we can exclude PND and PAR as they worsen the solution on an average and are ranked low. PD1 on an average, improves the solution in all four scenarios. We however, did not select the algorithm owing to its poor PIC value for scenario 4. Algorithms that had high PIC values generally have high PI values.

The minimum PI values, specially for non-deterministic algorithms are quite negative implying that these algorithms could yield really bad solutions. In practice however, we see that only a small fraction of runs yield extreme PIs. A majority of the runs yield a moderate improvement or worsening of solutions. To ascertain this, we can perform a histogram analysis. As an example, Figure 4 shows the histograms for distribution of PIs for top six algorithms HD1, HD3, HDX, HND, HAR, and HS8 for scenarios 1 and 2.

Among the algorithms, only HAR has extreme worsening of solutions (PI $<$ -12%). For the 8 processor case, HD1 has the least probability of worsening the solution. For the 32 processor case, most algorithms have a negligible probability of PI$<$-5%. This analysis can be repeated for scenarios 3 and 4 described in Table 5.

7.4. Speed-up

In this section, we present two different speed-up metrics, namely speed-up to cut-off (SC) and speed-up to termination (ST). The SC metric is based on time taken by the algorithms



(a) 8 processor run (b) 32 processor run

Figure 4 **Histogram of the distribution of PI for HD1, HD3, HDX, HND, HAR, and HS8 when starting with an initial feasible solution obtained using a näive construction heuristic. These results are obtained for 1 run of the deterministic heuristics and 10 runs of the non-deterministic heuristics for each test case respectively.**

to reach a common cut-off. The ST metric is based on wall-clock time. By analyzing these two metrics, we can obtain sufficient information to rank the algorithms according to their speed of solution.

7.4.1. Speed-up to Cut-off (SC) Using notations in Table 4, we can define the speed-up to cut-off metric as follows:

$$s = \left\{ \frac{t_{rm}^{(s)}}{t_{rm}^{(p)}} \ \forall \ m = 1, \ldots, M_{\text{TC}} \text{where for each } m, r = 1, \ldots, R_m \right\} \tag{22}$$

$$SC_{\text{avg}} = \text{geometric mean}(s) \tag{23}$$

$$SC_{\text{min}} = \min(s) \tag{24}$$

$$SC_{\text{max}} = \max(s) \tag{25}$$

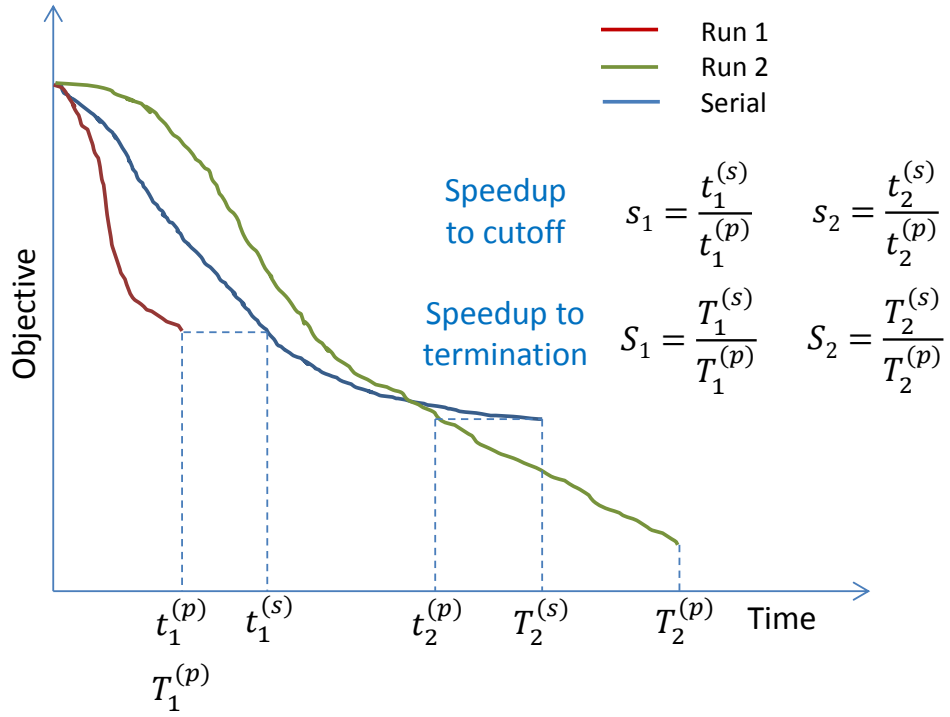


Figure 5 **Schematic describing calculations for speed-up to cut-off and speed-up to termination metrics.**

where, s is a set containing individual speed-up calculations for different runs of an algorithm on different test cases. This calculation is described for two runs of a parallel algorithm in Figure 5. When comparing run 1 with the serial solution, we see that the final objective value for run 1 is worse than the serial objective. Thus, $t_1^{(s)}$ denotes the time taken by the serial algorithm to reach the final objective value obtained by run 1. On the other hand, run 2 performs better than the serial algorithm and the cut-off objective value is equal to the final serial objective value. We can then compute SC as the geometric average of s_1 and s_2.

In our calculations, the cut-off for each run of an algorithm is different. This is to ensure that if we have only one run, the SC metric is still consistent. Furthermore, using a common cut-off for all the runs leads to a problem that can be illustrated by the following example: Consider three runs r1, r2, and r3 with final objective values 1, 2, and 3. The common cut-off will be 3. If we remove r3, the common cut-off will change to 2. The SC metric will then be a function of the runs chosen. This is undesirable.

In Table 8, the algorithms are ranked according to their *SC* values. Among the algorithms selected in section (7.3.1), we observe that the *SC* values for non-deterministic

algorithms HND, HAR, HS2, HS4, HS6, and HS8 are almost twice that of deterministic algorithms HD1, HD3, and HDX. Unless solution improvement is critical, using non-deterministic algorithms can reduce the overall clock-time for solution considerably.

SC is a fair metric. When the serial algorithm solution is of much better quality than the parallel algorithm solution, the cut-off is determined by the parallel algorithm. *SC* avoids taking into account the extra time taken by the serial algorithm in improving the solution over the final parallel algorithm solution. When the parallel solution is better than the serial benchmark, the cut-off is driven by serial benchmark objective.

	Näive						RT0					
	8 proc.			32 proc.			8 proc.			32 proc.		
	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX
HND	1.2	4.9	14.6	2.6	10.7	32.7	0.0	5.6	30.0	1.9	13.0	36.8
HAR	1.6	4.8	15.5	0.3	9.9	27.5	1.1	5.8	19.7	0.6	11.1	66.0
PND	1.6	4.4	9.4	2.9	8.6	29.3	0.2	4.1	17.4	2.4	10.8	36.8
PS2	1.6	4.3	14.3	2.9	8.5	23.5	0.8	3.6	17.5	1.2	11.4	66.1
PS8	1.6	4.1	14.3	3.6	8.2	22.5	0.8	3.7	28.2	1.7	11.4	44.5
PS4	1.4	4.3	14.3	2.3	8.0	23.5	0.7	3.8	20.5	1.8	11.0	65.4
PS6	1.5	4.1	14.2	2.1	7.7	23.8	1.0	3.6	16.9	1.8	11.2	65.2
HS8	1.4	4.1	11.6	3.3	8.1	25.8	0.8	3.4	15.4	2.9	9.9	38.1
PAR	1.8	4.7	16.7	2.4	7.6	20.5	0.3	3.6	17.7	2.1	9.3	31.5
HS4	1.5	4.0	11.8	2.7	8.4	24.1	0.8	3.4	13.7	1.5	9.1	24.0
HS6	1.5	4.0	12.9	2.8	8.2	26.9	0.8	3.4	14.8	1.0	9.0	30.5
HS2	1.3	4.1	12.2	3.1	7.9	27.9	0.8	3.4	12.6	1.9	8.7	31.4
HD3	1.2	2.8	6.9	3.1	4.4	8.4	0.6	3.1	9.7	2.4	6.7	12.4
HD1	1.0	2.7	6.2	1.8	4.0	7.1	0.5	2.9	8.0	1.5	5.9	16.5
HDX	1.3	2.3	4.5	2.1	3.4	5.8	0.8	3.1	6.8	2.0	5.8	13.3
PD1	1.1	2.4	4.2	2.3	4.3	8.3	0.5	2.5	5.8	2.0	4.2	9.2
PD3	1.2	2.0	5.0	0.9	3.7	9.3	1.3	2.4	8.4	1.1	4.5	11.8
PDX	0.9	2.0	4.5	1.2	2.7	5.1	0.8	2.3	5.0	1.3	5.2	15.5

Table 8 Algorithms ranked according to SC metric.

7.4.2. Speed-up to Termination (ST) Speed-up to termination is a conservative metric that complements the speed-up to cut-off metric. Using notations in Table 4, we can define the speed-up to termination metric as follows:

$$S = \left\{ \frac{T_m^{(s)}}{T_{rm}^{(p)}} \; \forall \; m = 1, \ldots, M_{\text{TC}} \text{where for each } m, r = 1, \ldots, R_m \right\} \tag{26}$$

$$ST_{\text{avg}} = \text{geometric mean}(S) \tag{27}$$

$$ST_{\text{min}} = \min(S) \tag{28}$$

$$ST_{\text{max}} = \max(S) \tag{29}$$

where, S is a set containing individual speed-up calculations for different runs of an algorithm on different test cases. This calculation is described for two runs of a parallel algorithm in Figure 5. The algorithms are ranked according to their ST values in Table 9. The ST values of algorithms are less than their SC values. Again, we observe that the ST values for the non-deterministic algorithms HND, HAR, HS4, HS6, HS2, and HS8 are almost 1.5 times that of deterministic algorithms HD1, HD3, and HDX.

	Näive						RT0					
	8 proc.			32 proc.			8 proc.			32 proc.		
	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX
PS4	1.2	4.6	11.9	3.0	9.4	33.0	1.6	4.8	16.8	3.8	14.4	52.1
PS2	1.0	4.5	14.6	3.0	9.8	33.3	0.9	4.7	16.9	3.8	14.5	53.9
PS8	1.2	4.5	11.9	3.4	9.2	38.4	1.6	4.7	16.8	4.8	14.0	54.1
HS4	0.8	4.3	12.4	2.8	9.3	32.6	1.5	5.0	14.9	4.3	13.7	54.6
PS6	0.8	4.3	10.7	3.4	9.2	30.6	1.3	4.8	16.8	4.5	14.0	53.8
HS2	0.8	4.3	12.9	2.3	8.8	35.8	1.5	5.0	14.6	4.2	13.8	58.2
HS8	0.9	4.1	13.3	2.3	8.8	36.5	1.5	4.9	15.3	4.2	13.9	54.7
PAR	1.0	4.4	16.9	2.4	8.1	24.2	1.5	5.3	16.8	0.2	11.9	53.6
HS6	0.9	4.2	12.5	2.5	9.2	33.4	1.5	4.8	14.6	3.7	13.5	73.7
HAR	0.8	4.1	12.5	0.3	9.0	36.6	1.5	5.1	11.1	2.4	12.2	34.0
HND	0.1	4.0	11.6	2.1	9.3	37.5	0.1	4.8	13.9	2.4	12.3	32.4
PND	1.0	4.3	11.4	3.5	9.2	25.6	1.7	5.4	16.9	4.2	13.5	42.5
HD3	0.8	2.6	5.4	1.5	4.4	11.2	1.4	4.1	16.2	3.4	7.7	26.3
PDX	1.4	2.4	4.7	1.7	3.8	8.4	1.3	2.8	5.2	1.4	6.4	27.2
PD3	0.7	2.4	5.7	1.4	4.4	10.9	1.7	3.6	9.1	3.0	6.7	18.9
HD1	0.9	2.5	5.5	1.8	3.8	8.8	1.1	3.2	9.9	3.4	7.3	19.1
PD1	0.8	2.3	5.3	1.7	4.5	11.6	1.1	2.9	8.7	3.7	6.6	19.2
HDX	0.9	2.2	3.5	1.5	3.9	9.2	1.2	3.0	7.1	2.4	6.1	12.5

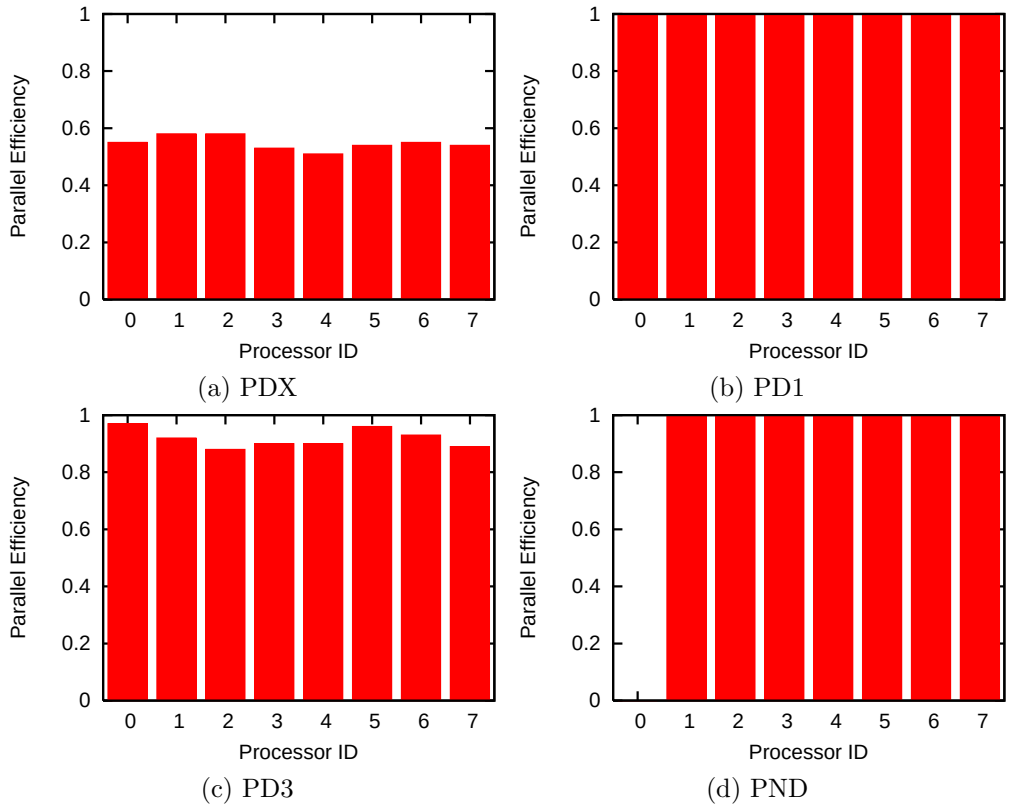
Table 9 Algorithms ranked according to ST metric.

7.5. Parallel Efficiency

As discussed in section (4.1), parallel efficiency of an algorithm depends on its implementation and the hardware state. If the parallel cluster is congested, the PE drops. Parallel efficiency studies are typically conducted for one run of a representative test case. The PE value for each processor provides insight into the inter-processor communication.

By construction, the communication strategy used in PDX, PD1, PD3, and PND provides a good sample to study parallel efficiencies of all the proposed algorithms. In Figure 6, we consider the individual processor PEs for one run of the test case P17 under scenario 1 (see Table 5).

In PDX, each processor solves one sub-problem. It then waits for all other processors to finish solving their sub-problem before sharing its solution with them. This process



(a) PDX (b) PD1

(c) PD3 (d) PND

Figure 6 PE **for each processor for one run of test case P17 under scenario 1.**

repeats multiple times in a major iteration. Each time a processor waits, it adds to its idle time, whereby yielding a low PE. On the other hand, in PD1, the processors share their solutions once per major iteration. Hence, the idle time as well as the inter-processor communication time are minimized. This yields a near ideal PE of 1 in each processor. In PD3, the processors share their solutions three times in each major iteration. Hence the PE values are in between PD1 and PDX.

PND uses dynamic resource allocation that ideally yields an average PE of around (N-1)/N over all the processors, where N is the number of processors. We also observe that the workers are always occupied and the processors that act as workers have a PE of 1. The manager does not take part in the solution process and its PE is 0.

It might seem that PD1 alleviates the load balancing issues. For a different test case with different initial ordering of sub-problems, we can easily arrive at a scenario where one processor is tasked with the solution of the most time-intensive sub-problems. In that case, PD1 will suffer with a low parallel efficiency.

8. Conclusions

Six deterministic algorithms PDX, PD1, PD3, HDX, HD1, and HD3; and twelve non-deterministic parallel algorithms PND, PAR, PS2, PS4, PS6, PS8, HND, HAR, HS2, HS4, HS6, and HS8 were proposed for the solution of large LNG-IRPs as depicted in equations (1) through (14). The performance of each of these algorithms were quantified using the PIC, PI, SC, and ST metrics and tested against 11 large scale test problem instances.

Based on the PIC metric, eleven algorithms were found to be comparable in accuracy to the serial benchmark algorithm. These are HD1, HD3, HDX, HND, HAR, HS8, HS2, HS4, HS6, PND and PAR. The remaining algorithms were not sufficiently accurate over all the test cases. Based on their ranking using the PI metric, the algorithms PND and PAR were dropped from the selected algorithms. In general, the deterministic algorithms performed better than their non-deterministic counterparts in improving the solution quality.

The non-deterministic algorithms yielded a speed-up to cut-off of around 4.5x on 8 processors and 10x on 32 processors. These values were around twice the speed-ups obtained by the deterministic algorithms. If solution repeatability is a critical, the deterministic algorithms are a clear choice. Otherwise, using a non-deterministic algorithm can greatly decrease the wall-clock time without compromising accuracy.

None of the parallel meta-heuristics defined in this paper use knowledge of the network structure or constraints define for the LNG-IRP. They can easily be used in any application that involved local neighborhood search. Considerable avenues for future research exist within the context of meta-heuristics presented in this paper. We can embed new local search operators as defined in (Shao et al. (2013)). Other communication strategies viz. ring communication (Gropp et al. (1999)), peer-to-peer communication can be investigated for incorporating diversification in the heuristics.

Finally, we introduce several performance metrics for characterizing the accuracy and the speed-up achieved by these parallel neighborhood search algorithms. We also describe a procedure to rank several algorithms. These metrics can be broadly adapted by the research community for investigating parallelization of heuristics for optimization problems.

Acknowledgments

The authors wish to thank Bora Tarhan for suggestions and corrections to the manuscript, ExxonMobil Upstream Research Company for access to high-performance computing resources.

References

Andersson, H., M. Christiansen, K. Fagerholt. 2012. Transportation planning and inventory management in the lng supply chain. *Energy Systems* **3** 427–439.

Barr, R. S., B. L. Hickman. 1993. Reporting computational experiments with parallel algorithms: Issues, measures, and expert's opinions. *ORSA Journal of Computing* **5** 2–18.

Christiansen, M., K. Fagerholt. 2009. Maritime inventory routing problems. encyclopedia of optimization. springer 1947–1955.

Christiansen, M., K. Fagerholt, B. Nygreen, D. Ronen. 2007. Maritime transportation. handbooks in operations research and management science.elsevier **14** 189–284.

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

Crainic, T. G., M. Gendreau, J.-Y. Potvin. 2006. Parallel tabu search. parallel metaheuristics. john wiley & sons 298–313.

Crainic, T. G., M. Toulouse. 2009. Parallel meta-heuristics. Tech. Rep. 2009-22, CIRRELT.

Crainic, T. G., M. Toulouse, M. Gendreau. 1997. Towards a taxonomy of parallel tabu search algorithms. *INFORMS Journal on Computing* **9** 61–72.

Fodstad, M., K. T. Uggen, F. Romo, A.-G. Lium, G. Stremersch, S. Hecq. 2008. Profit maximization in the lng-value chain by combining market prices and ship routing. in proceedings of apiems 2008 the 9th asia pacific industrial engineering & management systems conference 2782–2793.

Fodstad, M., K. T. Uggen, F. Romo, A.-G. Lium, G. Stremersch, S. Hecq. 2011. Lngscheduler: A rich model for coordinating vessel routing, inventories and trade in the liquefied natural gas supply chain. *Journal of Energy Markets* **3** 31–64.

Furman, K. C., J.-H. Song, G. R. Kocis, M. K. McDonald, P. H. Warrick. 2011. Feedstock routing in the exxonmobil downstream sector. *Interfaces* **41** 149–163.

Goel, V., K. C. Furman, J.-H. Song, A. S. El-Bakry. 2012. Large neighborhood search for lng inventory routing. *Journal of Heuristics* 1–28.

Gronhaug, R., M. Christiansen. 2009. Supply chain optimization for the liquefied natural gas business. innovation in distribution logistics: Lecture notes in economics and mathematical systems. elsevier **619** 195–218.

Gronhaug, R., M. Christiansen, G. Desaulniers, J. Desrosiers. 2012. A branch-and-price method for a liquefied natural gas inventory routing problem. *Transportation Science* **44** 400–415.

Gropp, W., E. Lusk, R. Thakur. 1999. Using mpi-2: Advanced features of the message-passing interface. mit press. cambridge, ma .

Halvorsen-Weare, E. E., K. Fagerholt. 2010. Routing and scheduling in a liquefied natural gas shipping problem with inventory and berth considerations. *Annals of Operations Research* .

Jackson, R.H.F., P.T. Boggs, S.G.Nash, S. Powell. 1990. Report of the ad hoc committee to revise the guidelines for resporting computational experiments in mathematical programming. *Mathematical Programming* **49** 413–425.

Janson, S., D. Merkle, M. Middendorf. 2006. Parallel ant colony algorithms. parallel metaheuristics. john wiley & sons 171–201.

Jin, J., T. G. Crainic, A. Lokketangen. 2010. A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. Tech. Rep. 2010-54, CIRRELT.

Jin, J., T. G. Crainic, A. Lokketangen. 2012. A cooperative parallel metaheuristics for capacitated vehicle routing problems. Tech. Rep. 2012-46, CIRRELT.

M.D.Tusiana, G.Shearer. 2007. Lng: A nontechnical guide. pennwell .

Moreno-Perez, J. A., P. Hansen, M. Mladenovic. 2006. Parallel variable neighborhood search. parallel metaheuristics. john wiley & sons 247–266.

Perron, L. 2002. Practical parallelism in constraint programming. in proceedings of cp-ai-or 261–276.

Perron, L. 2003. Fast restart policies and large neighborhood search. in proceedings of cp-ai-or .

Rakke, J. G., M. Stalhane, C. R. Moe, M. Christiansen, H. Andersson, K. Fagerholt, I. Norstad. 2011. A rolling horizon heuristic for creating a liquefied natural gas annual delivery program. *Transportation Research Part C: Emerging Technologies* **19** 896–911.

Shao, Y., K. C. Furman, V. Goel, S. Hoda. 2013. Bound improvement for lng inventory routing. *submitted to Journal of Transportation Science* .

Song, J.-H., K. C. Furman. 2013. A maritime inventory routing problem: Practical approach. *Computers & Operations Research* **40** 657–665.

U.S.DOE. 2011. International energy outlook 2011: Us energy information administration Http://www.eia.gov/forecasts/ieo/nat_gas.cfm.

Algorithm 3 Parallel Non-Deterministic Meta-heuristic (N, $\mathcal{A}$, x_0)

procedure MANAGER(N, $\mathcal{A}$, x_0) ▷ Called in processor with rank 0
  $P_{\text{sent}}, P_{\text{done}} = 0, \hat{x} = x_0$
  for i=1 to $N-1$ **do**
    Send $(P_{\text{sent}}, x_0, \text{true})$ to processor with rank i
    $P_{\text{sent}} = P_{\text{sent}} + 1$
  end for
  repeat
    if any processor has sent its solution to manager **then**
      $P_{\text{done}} = P_{\text{done}} + 1$
      Get the solution x_{worker} and processor rank n_{worker}
      if objective value($\hat{x}$) > objective value(x_{worker}) **then**
        $\hat{x} = x_{\text{worker}}$
      end if
      if $P_{sent} < P$ **then**
        Send $(P_{\text{sent}}, \hat{x}, \text{true})$ to processor with rank n_{worker}
      else
        Send $(P_{\text{sent}}, \hat{x}, \text{false})$ to processor with rank n_{worker}
      end if
    end if
  until $P_{done} \geq P$
  Return $\hat{x}$
end procedure
procedure WORKER($\mathcal{A}$) ▷ Called in processors with rank 1 through $N-1$
  Get (i, x, flag) from manager (processor rank 0)
  if flag **then**
    $\hat{x}$ = Solve(i,$\mathcal{A}$,x)
  else
    Exit worker algorithm
  end if
  Send $\hat{x}$ to manager
end procedure

Algorithm 4 Parallel Acceptance-Rejection Meta-heuristic (N, $\mathcal{A}$, x_0)

procedure WORKER($\mathcal{A}$) ▷ $\hat{x}$ is locally stored solution

Get (i, x,flag) from manager (processor rank 0)

if flag **then**

O_m and O_w are objective values corresponding to x and $\hat{x}$.

Calculate $Q_{ar} = O_m/(O_m + O_w)$.

π = sample from Uniform[0,1] distribution

if $\pi < Q_{ar}$ **then**

$\hat{x} = x$

end if

$\hat{x}$ = Solve(i,$\mathcal{A}$,$\hat{x}$)

Send $\hat{x}$ to the manager and store locally.

else

Exit worker algorithm

end if

end procedure

Algorithm 5 Parallel Store K Solutions Algorithm (N, $\mathcal{A}$, K, x_0)

procedure MANAGER(N, $\mathcal{A}$, x_0) ▷ A pool of K best solutions is stored

$P_{\text{sent}}, P_{\text{done}} = 0, \hat{x} = x_0$

for i=1 to $N-1$ **do**

Send $(P_{\text{sent}}, x_0, \text{true})$ to processor with rank i

$P_{\text{sent}} = P_{\text{sent}} + 1$

end for

repeat

if any processor has sent its solution to manager **then**

$P_{\text{done}} = P_{\text{done}} + 1$

Get the solution x_{worker} and processor rank n_{worker}

if objective value(x_{worker}) is lesser than at least one solution in the pool **then**

Set the worst solution in the pool to x_{worker}

end if

if $P_{sent} < P$ **then**

π = Sample from Uniform[0,1] distribution.

O_j = objective value of j-th solution in the solution pool

Define $Q(j)$ as $\sum_{j=0}^{j} O_j^{-1} / \sum_{j=0}^{K-1} O_j^{-1}$ with $Q(-1) = 0$.

$\hat{x}$ = k-th solution in the pool if $Q(k-1) \leq \pi < Q(k)$

Send $(P_{\text{sent}}, \hat{x}, \text{true})$ to processor with rank n_{worker}

else

Send $(P_{\text{sent}}, \hat{x}, \text{false})$ to processor with rank n_{worker}

end if

end if

until $P_{done} \geq P$

Return $\hat{x}$

end procedure