A near-optimal algorithm for constraint test ordering in automated stowage planning

Lee, Zhuo Qi; Fan, Rui; Hsu, Wen-Jing

2018

Lee, Z. Q., Fan, R., & Hsu, W.- J. (2018). A near-optimal algorithm for constraint test ordering in automated stowage planning. IEEE Transactions on Automation Science and Engineering, 15(3), 1298-1308. doi:10.1109/TASE.2017.2779470

<https://hdl.handle.net/10356/106121>

<https://doi.org/10.1109/TASE.2017.2779470>

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The published version is available at:
<https://doi.org/10.1109/TASE.2017.2779470>

Downloaded on 26 Apr 2023 23:19:18 SGT

A Near-optimal Algorithm for Constraint Test Ordering in Automated Stowage Planning

Zhuo Qi Lee, Rui Fan, and Wen-Jing Hsu

Abstract—The container stowage planning problem is known to be NP-hard and heuristic algorithms have been proposed. Conventionally, the efficiency of the stowage planning algorithms are improved by pruning or reducing the search space. We observe that constraint evaluation is the core of most algorithms. In addition, the order at which the constraints are evaluated can have significant impact on the efficiency of the constraint evaluation engine. We propose Random Sample Model (RSM) and Sequential Sample Model (SSM) for analysis of the problem. We present and evaluate seven strategies in optimizing the constraint evaluation engine. We show how to achieve the optimal constraint ordering with respect to RSM and SSM respectively. However, the optimal ordering for SSM requires perfect information about the states of the constraint tests which is impractical. We present an alternative strategy and show empirically that its efficiency is close to the optimal. Experiments show that, compared to a naïve ordering, an average of 2.74 times speed up in the evaluation engine can be achieved.

Note to Practitioners: **Abstract**—Automated stowage planning has become a trend as the number of mega-scale containerships being deployed has increased drastically over the past decade. Due to the nature of the problem, it is impractical to expect the best stowage plan within a limited amount of time, and hence many heuristics have been devised to reduce the solution space. Many heuristics algorithms share a common core - repeatedly selecting a stowage slot and a container (selection mechanism) and evaluate whether all of constraints are satisfied. To improve the efficiency, most studies aim to reduce the number of location-container pairs being considered. We consider an alternative approach which improves the efficiency of the constraint evaluation engine. Specifically, we show how to improve the efficiency by strategically reordering the sequence in which the constraints are evaluated. With the improvements on the efficiency, the stowage algorithm is one step closer to being able to generate solutions in real time. This may enable the shipping companies to take last-minute shipment order and react to changes in demands quickly.

Index Terms—Automation; Testing; Logistics; Optimization methods; Markov Processes.

I. INTRODUCTION

The task of stowage planning is to assign containers stowage locations on the containership such that the number of containers that can be transported between ports is maximized without violating the safety and stability constraints while minimizing the operational cost. In addition, a good stowage plan also balances the workload of the quay cranes when

loading or unloading the containers at the ports. With the increase in the volume of international trade and economies of scale, mega-scale containerships are built to meet the demand of the market. These mega-scale containerships induce big challenges to the port-related logistics and supply chain. Thus, much research has been conducted to improve the efficiency of the port and operations [1–16].

With the trend to transit from relatively smaller containerships (capacity of a few thousand Twenty-foot Equivalent Units or TEUs) to large scale containerships (more than 15000 TEUs capacity), the stowage planning process has become exceedingly taxing. It may take experienced human planners several hours of effort to devise a feasible stowage plan. Moreover, shipping lines face an increasingly daunting challenge to hire and retain sufficiently qualified and experienced planners.

In view of the challenges above, automated stowage plan generation has become a trend. Fast automated stowage plan generation also enables the shipping company to take last-minute customer orders for shipment. This capability may be offered as a premium service to customers with urgent requirements. In addition, fast stowage plan generation may also enable a feedback loop to the tactical planning phase, in which the port rotations are decided [5]. This allows the shipping company to react and adapt to changes in demands and increase the efficiency of resource utilization.

In this paper, we present an improvement, in terms of plan generation efficiency, over an existing stowage algorithm whose performance, including both the quality of solution and execution time, has been investigated empirically and reported in [7–10]. For the sake of completeness, we briefly describe the overall approach of the algorithm in Section II.

Our contributions. We show how to improve the efficiency of a class of stowage planning algorithm by optimizing the performance of the constraint evaluation engine. Specifically, we:

- observe the existence of a recurring pattern in the outputs of consecutive rounds of constraint tests;
- present a new state-based model to incorporate the above observations;
- derive the theoretically optimum constraint ordering strategy w.r.t. the new constraint test model;
- show how to achieve near-optimal efficiency in the constraint evaluation engine without assuming perfect knowledge about the states of the constraint tests.

Fig. 1. Cross section view of a containership. The space on the ship is divided into discrete locations where the x-axis direction in the figure corresponds to the bays and the y-axis direction corresponds to the tiers. Bays for stacking 40 ft containers are indexed with even numbers while bays for stacking 20 ft containers are indexed with odd numbers. Two consecutive 20 ft bays form a single 40 ft bay.

Paper organization. In Section II, we briefly introduce relevant concepts about container stowage. Then we review literature on container stowage problem and highlight the role played by a constraint evaluation engine. We also briefly review recent development in constraint test ordering optimization. In Section III, we present the Random Sample Model (RSM), which is widely used in the literature for optimizing the constraint test ordering. We show how to refine RSM using the new observations and propose the State-based Sequential Sample Model (SSM). We present both static and dynamic strategies for reordering the constraint tests. We also analyse the optimal efficiencies with respect to RSM and SSM. In Section IV, we present empirical evaluations of the proposed strategies. We comment on the efficiency of the proposed strategies as well as the general applicability of the approach in other domains in Section V and conclude the paper in Section VI.

II. PRELIMINARY

We briefly introduce some of the main considerations in generating a stowage plan. Fig. 1 shows the cross section view of a containership. The space on the ship is divided into discrete locations where the x-axis direction in Fig. 1 corresponds to the bays and the y-axis direction corresponds to the tiers. Bays for stacking 40 ft containers are indexed with even numbers while bays for stacking 20 ft containers are indexed with odd numbers. Two consecutive 20 ft bays form a single 40 ft bay. The locations are divided into above-deck and below-deck locations by hatch covers. Fig 2 shows a slice of a 40 ft bay. The x-axis direction in Fig. 2 corresponds to rows which are indexed starting from the middle. The y-axis direction corresponds to the tiers.

A. Stowage Constraints and Considerations

A feasible stowage plan is an assignment of locations to the containers such that the containership can transport the goods safely between the ports. The safety requirements can be categorized into three main types: (i) dangerous goods (DGs) constraints; (ii) stacking constraints; (iii) balance and weight distribution constraints [8, 11].

Fig. 2. A slice of 40 ft bay (two consecutive 20 ft bays) from the containership. The x-axis direction in the figure corresponds to rows which are indexed starting from the middle. The y-axis direction corresponds to the tiers. Hatch covers divide the locations into above- and below-deck.

DGs need to be handled with care, for instance, DG containers which contain flammable objects must not be placed close to the heat sources. In addition, some DGs may emit hazardous gas and must not be placed near to the living quarters or close to other DG which may react to the emitted gas. The stacking constraints specify whether different types of containers can be stacked on one another. In addition, the total weight and height of a stack cannot exceed a specified limit. In the balance and weight distribution constraints, the forces applied onto the ship structure (such as the buoyancy of the vessel and weights of the containers) are considered. The containers have to be distributed such that the ship can maintain balance during the voyage while ensuring that the differences in weight distribution exert minimal stress on the ship structure (such as shear force and bending moments). In the case where the balance and weight distribution cannot be satisfied by swapping the locations of containers, the ballast tanks are filled with water to achieve balance.

However, a feasible stowage plan does not guarantee profit as there are other operational costs such as fuel and handling cost. The fuel cost is affected by the weight distribution and the amount of ballast water used. On the other hand, the handling cost is affected by the stacking pattern. Stowing a container destined for a further port on top of another container which is destined for a nearer port is said to be an *overstow*. Overstows induce additional handling cost as the containers can only be accessed when all the containers that are stacked on top are removed. The containerships are also charged port dues for the duration of stay at a port. Part of the contributing factor to the port stay duration is the workload distribution between the bays [9]. The cost incurred for staying at a port for an extended period of time is especially significant for megavessels [9, 17].

B. Related Works - Stowage Algorithms

The container stowage optimization problem, which is also known as the Master Bay Plan Problem (MBPP), was shown to be NP-hard in [12]. Since it is computationally challenging to obtain an optimal solution, many studies are conducted to devise a *near optimal* solution, rather than the *optimum*, to the problem. The existing approaches for solving the problem can mainly be divided into the following three categories: (i) mixed

Fig. 3. Flow chart of typical stowage algorithm. The container-location pair selection mechanism and the constraint evaluation engine form the core of most stowage algorithms.

integer programming (MIP) based, e.g., [6, 13, 14, 18, 19]; (ii) meta-heuristics based, e.g., [7, 14, 15], and (iii) heuristics based, e.g., [9, 14, 16, 20]. Due to practical considerations, a 10-minute time bound is usually imposed [21] on automated stowage planning algorithms. While most of the algorithms above provide good solutions to MBPP, some may not be applicable to real world usage due to either the implicit simplifying assumptions made [8], or the long period of time required to generate a good feasible solution [15].

Heuristics based algorithms are devised by observing the patterns and rules-of-thumb utilised by the human planners. Although these approaches usually do not offer theoretical guarantees on the quality of the solution, they can generate multiple feasible solutions quickly and the pattern of solutions generated can be easily modified by the human planners [10].

Fig. 3 shows the flow chart of a typical stowage algorithm. The algorithm proceeds iteratively. At each iteration, a pair of container and stowage location is selected. Then, the selected container-location pair is passed into the *constraint evaluation engine*. The constraint evaluation engine returns true if the container satisfies all the required constraints at the specified location, and false otherwise. The process is repeated until there is no container or location remaining for consideration. As shown in the figure, there are two options for improving the efficiency of the stowage algorithm: (i) reduce the number of container-location pairs being considered; (ii) reduce the time spent in evaluating constraints. We consider the largely un-explored option (ii) in this study. As a result, the techniques presented in this paper may be used in conjunction with other optimization methods to achieve an even greater efficiency, as long as the resultant framework needs to evaluate more than one test sequentially at any stage of the algorithm.

C. Related Works - Constraint Test Ordering

A container-location pair is evaluated against an ordered set of tests in a sequential manner. Each test is composed of a set of related criteria. For instance, a test could be checking the stack weight limit, or checking the segregation among the DGs. The constraint evaluation returns false if the container-location pair is rejected by at least one of the tests.

The order in which the tests are evaluated could have significant impact on the total costs incurred. For instance, suppose

that we are given three tests (ϕ_1, ϕ_2, ϕ_3) with evaluation costs of 1000, 100, and 10 respectively. Assume that for the current iteration, both ϕ_1 and ϕ_2 return “True” while ϕ_3 return “False”. If we evaluate the sequence in the given order of (ϕ_1, ϕ_2, ϕ_3) , then the cost incurred would be 1110. However, if we evaluate ϕ_3 first, then the cost incurred is 10, which is significantly less than the default order. The main idea is to place a violating test to the front, so that a majority of the tests can be skipped, and hence saving computational time. Of course, we may not know in advance which tests would return “False” and hence the problem is usually studied with a probabilistic framework.

In fact, the constraint test optimization has been studied under different themes. In [22], Garey studied the case where the tests follow certain precedence requirements, i.e., some tests may have to be conducted before some other tests. The tests ordering problem with precedence requirements was shown to be NP-hard in [23]. On the other hand, Berend et al. [23] also considered the case where a test is one-sided perfect, i.e., the test may erroneously reject a sample (false negative) with certain probability, but not the other way round. In [24], Berend et al. studied algorithms for tests with extreme dependencies, i.e. disjoint tests and dominating tests. For the case of disjoint tests, each test may cover different aspects of the sample. For the case of dominating tests, a test may cover a subset of another test. In [25], Ünlüyurt presented a survey of the test order optimization problem.

Test order optimization has applications in diagnostic based processes like cargoes inspections [2, 26, 27], regression testing for software engineering [28], and the design of screening procedures [22]. It has also been applied in the machine learning domain for improving the reaction time of classification algorithms [29].

Overall, the existing studies focused on the case where the test samples arrive at the tester in a random order. In the following, we show that when applied in the context of container stowage problem, the samples typically arrive at the tester following certain patterns. Then, we show how to utilize such knowledge to reorder the tests for improving the efficiency of the constraint evaluation engine.

III. CONSTRAINT ORDERING

A. Preliminaries

In the following, we formally define the terms and formulate the problem of constraint test optimization.

Definition 1. A constraint test, denoted by ϕ_i , checks whether a given container-location pair satisfies certain criteria. The constraint test may evaluate to “True” if the criteria are not violated by the container-location pair, or “False” if otherwise.

Definition 2. The evaluation cost, denoted by c_i , of a constraint test ϕ_i is the computational time incurred for obtaining the result of the constraint test on a given container-location pair.

TABLE I
EXAMPLES OF CONSTRAINT TESTS

Constraint Names	Avg Cost (nanosecs)	Probability of False	Description
DGConstraint	4552.879	0.097657	Checks if the DG segregation and stowage rules are observed.
Overstow	1559.011	0.109025	Checks if a container destined for a further port is stowed on top of another container which is destined for a nearer port.
StackLimit	392.4948	0.088814	Checks if the sum of weight of the current stack exceeds the stack weight limit.
CrossHatchOverstow	243.4047	0.147427	Checks if allocating the container at the current slot on deck causes the overstow problem across a hatch cover.
...	...	...	...

The evaluation cost of a constraint test may be obtained by taking the difference between the timestamps before and after executing the code segment which corresponds to the constraint test. While the evaluation cost of a constraint test may be different between the “True” and “False” cases, for the benefit of analysis, we assume the same cost for both cases. Extending to different test costs according to the evaluation result is straightforward [30].

Definition 3. A constraint test sequence, denoted by $\Phi = (\phi_1, \phi_2, \phi_3, \dots, \phi_N)$, is an ordered set of N constraint tests. The result of evaluating a constraint test sequence is “True” if all of the tests evaluate to “True”, and it is “False” if otherwise.

In this paper, we assume that the tests in a constraint test sequence are mutually independent, i.e., the result of one test ϕ_i does not affect the result of another test ϕ_j , for $i \neq j$. This is because each constraint test is responsible for different aspect of the stowage planning problem. In the case where one test is related to another, it makes sense to group the related tests together as a single test as updating a single shared variable may be more computationally efficient.

Definition 4. A strategy produces one or more orderings of the constraint test sequence Φ according to certain rules. An ordering produced by strategy X at time t is denoted by $\Phi^X(t) = (\phi_1^X(t), \phi_2^X(t), \phi_3^X(t), \dots, \phi_N^X(t))$.

Definition 5. The cost of evaluating an ordering of the sequence produced by a strategy X at time t , $\Phi^X(t)$, on a container-location pair s , denoted by $C(\Phi^X(t), s)$, is the cumulative cost accrued from applying the first constraint test to the first test that evaluates to “False”, or the sum of evaluation costs of all constraint tests if all of the tests evaluate to “True”, i.e.

$$C(\Phi^X(t), s) = \begin{cases} \sum_{i=1}^k c_i^X(t) & \text{if } \phi_k^X \text{ evaluates to “False” and} \\ & \phi_i^X \text{ evaluates to “True” } \forall i < k \\ \sum_{i=1}^N c_i^X(t) & \text{otherwise} \end{cases} \quad (1)$$

where $c_i^X(t)$ is the cost of the test at index i in the ordering produced by strategy X at time t .

The goal of constraint test optimization is to order the constraint tests such that the overall computational cost of evaluating a set of container-location pairs against the constraint

Fig. 4. Schematic of Random Sample Model. A constraint test evaluates to “True” or “False” probabilistically. The constraint test evaluation is oblivious of the container allocation process.

test sequence is minimized, i.e.,

$$\min \sum_{t=1}^{|S|} C(\Phi(t), s_t) \quad (2)$$

where $s_t \in S$ and S denotes the set of all container-location pairs considered in the stowage planning engine.

B. Static Ordering Strategies

Since we may not know in advance which tests will be violated, a probabilistic framework is adopted to model the constraint tests. Conventionally, the constraint order optimization problem is studied by considering that a randomly selected sample is evaluated at the constraint evaluation engine. A sample in this context refers to a container-location pair. As such, no differentiation can be made between the samples, and hence only a single fixed ordering of the constraint test sequence is considered. This is captured in the *Random Sample Model (RSM)* as presented in the following. Fig. 4 shows the schematic of RSM.

Definition 6. Random Sample Model describes a scenario in which a randomly selected sample is evaluated by the constraint evaluation engine. Under this model, a constraint test ϕ_i may evaluate to “True” with probability $p(\phi_i)$, and to “False” with probability $1 - p(\phi_i) = f(\phi_i)$.

Table I shows some examples of the constraint tests. The set of constraint tests in the stowage planning algorithm studied in this paper comprises 27 constraint tests. As shown in the table, the constraint tests have different average costs and probabilities of returning “False”.

In the following, we present strategies for ordering the constraint tests for optimizing the efficiency of the constraint evaluation engine with respect to RSM. Since each strategy produces a single fixed ordering, we drop the time parameter t in this section.

1) *Lowest Cost First (LC) Strategy*: By code profiling, we may collect the cost of evaluating the constraint tests. Intuitively, one may order the constraint tests sequence by sorting the sequence in non-decreasing order of costs as follows:

$$\Phi^{LC} = (\phi_1^{LC}, \phi_2^{LC}, \phi_3^{LC}, \dots, \phi_N^{LC})$$

such that $\text{cost}(\phi_i^{LC}) \leq \text{cost}(\phi_j^{LC})$ for $i \leq j$.

However, a particular constraint test with low cost may also evaluate to “True” very frequently and thus is contributing to the evaluation cost most of the time.

2) *Highest Probability First (HP) Strategy*: We may also collect data about how frequently a test returns “False”. Recall that the constraint evaluation engine stops at the first “False” encountered. Hence, to know the true violation frequency of a test in the data collection phase, we need to modify the constraint evaluation engine slightly such that it doesn’t stop the evaluation procedure after encountering a “False” result.

Under RSM, the samples (container-location pairs) are drawn randomly, and hence we may interpret the constraint violation frequencies as probabilities. With this information, we can order the constraint test sequence in non-increasing order of “False” probability as follows:

$$\Phi^{HP} = (\phi_1^{HP}, \phi_2^{HP}, \phi_3^{HP}, \dots, \phi_N^{HP})$$

such that $f(\phi_i^{HP}) \geq f(\phi_j^{HP})$ for $i \leq j$, where $f(\phi_i^{HP}) = 1 - p(\phi_i^{HP})$ denotes the probability of constraint test at index i to return “False”, and $p(\phi_i^{HP})$ denotes the probability of the constraint test to return “True”.

Under RSM, while the HP strategy guarantees that the least expected number of tests is evaluated, the strategy does not consider the cost of each evaluation. A constraint test with high $f(\cdot)$ may also incur a high cost of evaluation.

3) *Cost-effective First (CE) Strategy*: As seen above, both LC and HP cannot guarantee the cost optimality of the constraint test sequence. We show how the optimal ordering can be obtained with respect to RSM. With RSM, we may calculate the expected cost of evaluating a sequence as follows:

Definition 7. The expected cost of evaluating a sequence with respect to a strategy X is given by

$$E[C(\Phi^X)] = \lim_{|S| \rightarrow \infty} \frac{\sum_{s \in S} C(\Phi^X, s)}{|S|} \quad (3)$$

$$= c_1^X + p_1^X c_2^X + p_1^X p_2^X c_3^X + p_1^X p_2^X p_3^X c_4^X + \dots + p_1^X p_2^X \dots p_{N-1}^X c_N^X \quad (4)$$

where c_i^X is the cost of the constraint test at index i as dictated by strategy X , and p_i^X is the probability of the constraint test at index i as dictated by strategy X returning “True”.

By combining the insights gained from both LC and HP strategies, we can order the constraint test sequence in non-increasing (or simply decreasing) order of cost-effectiveness, which is defined as follows:

Definition 8. The cost effectiveness of a constraint test is given by:

$$\theta_i = \frac{f(\phi_i)}{\text{cost}(\phi_i)} \quad (5)$$

which is the ratio of “False” probability to the cost of evaluating the test.

In fact, it was shown in [23] that, ordering the constraint tests in non-increasing order of cost-effectiveness achieves the lowest expected evaluation cost among all possible permutations w.r.t. RSM. We state and show a short proof of the proposition [22, 31] because it will be extended in Proposition 2:

Proposition 1. Suppose that we are given a constraint test sequence $\Phi = \{\phi_1, \phi_2, \dots, \phi_N\}$ that follows RSM where a test ϕ_i returns “False” with probability $f(\phi_i)$ and takes $\text{cost}(\phi_i)$ to evaluate. If the sequence is sorted such that $\Phi^{CE} = (\phi_1^{CE}, \phi_2^{CE}, \dots, \phi_N^{CE})$, where $\theta_i^{CE} \geq \theta_j^{CE}$ for $i \leq j$, then $E[C(\Phi^{CE})] \leq E[C(\Phi^X)]$ for any strategy X .

Proof. For contradiction, suppose Φ^{CE} is not the optimum ordering. Then there exists some i in the optimum strategy Y such that $\theta_i^Y < \theta_{i+1}^Y$ and we have $E[C(\Phi^Y)] \leq E[C(\Phi^X)]$ for any strategy X . By Eq.(4), the expected cost of Φ^Y is given by

$$E[C(\Phi^Y)] = c_1^Y + p_1^Y c_2^Y + p_1^Y p_2^Y c_3^Y + \dots + p_1^Y p_2^Y \dots p_{i-1}^Y c_i^Y + p_1^Y p_2^Y \dots p_i^Y c_{i+1}^Y + \dots + p_1^Y p_2^Y \dots p_{N-1}^Y c_N^Y \quad (6)$$

Next, we swap ϕ_i^Y and ϕ_{i+1}^Y and denote the resulting sequence as Φ^Z . The expected cost of Φ^Z expressed using the order of strategy Y is given by:

$$E[C(\Phi^Z)] = c_1^Y + p_1^Y c_2^Y + p_1^Y p_2^Y c_3^Y + \dots + p_1^Y p_2^Y \dots p_{i-1}^Y c_{i+1}^Y + p_1^Y p_2^Y \dots p_i^Y c_i^Y + \dots + p_1^Y p_2^Y \dots p_{N-1}^Y c_N^Y \quad (7)$$

By subtracting Eq.(6) from 7, we get

$$\begin{aligned} E[C(\Phi^Z)] - E[C(\Phi^Y)] &= p_1^Y p_2^Y \dots p_{i-1}^Y c_{i+1}^Y + p_1^Y p_2^Y \dots p_{i-1}^Y p_i^Y c_i^Y \\ &\quad - p_1^Y p_2^Y \dots p_{i-1}^Y c_i^Y - p_1^Y p_2^Y \dots p_i^Y c_{i+1}^Y \\ &= p_1^Y p_2^Y \dots p_{i-1}^Y c_i^Y (p_{i+1}^Y - 1) \\ &\quad + p_1^Y p_2^Y \dots p_{i-1}^Y c_{i+1}^Y (1 - p_i^Y) \\ &= p_1^Y p_2^Y \dots p_{i-1}^Y [f_i^Y c_{i+1}^Y - f_{i+1}^Y c_i^Y] \end{aligned}$$

Since $\theta_i^Y < \theta_{i+1}^Y$, we have $\frac{f_i^Y}{c_i^Y} < \frac{f_{i+1}^Y}{c_{i+1}^Y}$, thus $f_i^Y c_{i+1}^Y - f_{i+1}^Y c_i^Y < 0$. Therefore $E[C(\Phi^Z)] < E[C(\Phi^Y)]$. This means that, by swapping an adjacent pair of tests from Y with $\theta_i^Y < \theta_{i+1}^Y$, the expected time can be decreased. Thus, strategy Y is not optimal, a contradiction. Hence, the optimum strategy must order the tests in non-increasing order of cost-effectiveness. $\square$

C. Dynamic Ordering Strategies

The strategies in the previous subsection assumed that the samples arrive at the evaluation engine in a random order. However, in practice we observed certain recurring patterns in the test evaluation result, as shown in Fig. 5.

Fig. 5. An example of recurring pattern for evaluating multiple samples against a single constraint test from real historical log of constraint evaluation. A “True” result is usually followed by a series of “True” before turning into a series of “False” and vice versa.

The ‘recurring’ pattern can be explained by considering, for example, the stack weight limit constraint test. The test checks if adding the container to the stack at which candidate location resides will exceed the specified stack weight limit. As the stowage planning algorithm plans on a stack-by-stack basis, when the stack is low, the test result is usually “True”. This means that it is safe to place the container at the candidate location without violating the stack weight limit. Testing additional containers at the same location is likely to yield the same “True” result, which leads to a chain of “True”. However, as more containers are allocated onto the stack, it becomes much more likely to violate the stack weight limit. Once the limit is exceeded, the test evaluates to “False” until the stowage algorithm moves on to another stack. This leads to the observed repeating pattern of a long chain of “True”s followed by another chain of “False”s.

With this observation, we see that the test samples are indeed not independent of each other. In fact, a test sample (container-location pair) is related to its previous sample since most stowage planning algorithms allocate containers in a localized manner, with consecutive allocations close to each other. Thus, it is likely that the current test result is dependent on the previous test result. This indicates that it may not be appropriate to assume that the test samples are selected randomly (RSM). Hence, we consider reordering the constraint test sequence dynamically. Since the ordering of the test sequence can change as opposed to the static ordering strategies described in the previous section, we add a time parameter to the notation. The constraint test sequence ordered by using strategy X at time t is then denoted by $\Phi^X(t) = (\phi_1^X(t), \phi_2^X(t), \dots, \phi_N^X(t))$.

As a result, the constraint evaluation engine adopts a two-phase process for each iteration, namely an *evaluation phase* and a *re-ordering phase*. The evaluation process starts at time 0, with an initial ordering of the constraint test sequence $\Phi^X(0)$, where X denotes the name of the strategy used in ordering the test. One possible way is to initialize the test sequence using the CE strategy, i.e., $\Phi^X(0) = \Phi^{CE}$. At any time step t , the constraint evaluation engine receives a sample and starts the evaluation phase, which checks through the constraint test sequence $\Phi^X(t)$ for violation. In the re-ordering phase, also at time step t , the constraint evaluation engine re-orders the constraint test sequence based on the observation obtained at the current round, and produce a new ordering $\Phi^X(t+1)$. The process is repeated until the stowage planning task is

completed.

1) Recent First (RF) Strategy: An obvious strategy is to move the recently violated constraint test to the front of the sequence at each time step. This is because a test that returned “False” previously is likely to return “False” on the next sample. The RF strategy orders the constraint test sequence in the following manner:

$$\Phi^{RF}(t+1) = (\phi_k^{RF}(t), \phi_1^{RF}(t), \phi_2^{RF}(t), \dots, \phi_{k-1}^{RF}(t), \phi_{k+1}^{RF}(t), \dots, \phi_N^{RF}(t))$$

where k is the index of the first test that evaluates to “False” at time t .

However, similar to the HP strategy, a recently violated constraint test could be very costly to evaluate. In addition, repeatedly moving constraint tests to the front may destroy any initial ordering of the sequence very quickly. Such behaviour could be detrimental to a sequence which is initialized according to some other strategies such as HP, LC, or CE.

2) Recent First with Restriction (RFR) Strategy: With the observations above, we can refine the RF strategy such that the recently violated constraint test is moved half-way to the front as follows:

$$\Phi^{RFR}(t+1) = (\phi_1^{RFR}(t), \phi_2^{RFR}(t), \dots, \phi_{\lfloor k/2 \rfloor - 1}^{RFR}(t), \phi_k^{RFR}(t), \phi_{\lfloor k/2 \rfloor}^{RFR}(t), \phi_{\lfloor k/2 \rfloor + 1}^{RFR}(t), \dots, \phi_{k-1}^{RFR}(t), \phi_{k+1}^{RFR}(t), \dots, \phi_N^{RFR}(t))$$

We give an example scenario in the following. Say, in the evaluation phase, the constraint tests at index 1 to 9 evaluates to “True” and the constraint test at index 10 evaluates to “False”. Then, in the re-ordering phase, constraint test 10 is inserted to index 5, pushing the original tests at 5 to 9 to position 6 to 10 respectively.

The benefit of this strategy is that, only a test that is violated more frequently can be moved to the front of the sequence. This emulates the HP strategy dynamically. In addition, the RFR strategy does not require any information about the costs and probabilities of the constraint tests.

3) State-based Sequential Sample Model and State-based (SB) Strategy: Similar to the LC and HP strategies, the RF and RFR strategies do not offer guarantees on the expected evaluation cost. Recall the observation that the test results exhibit certain ‘recurring’ pattern, in which the current test result is related to the previous test result. This suggests a remarkable resemblance to the behaviour of a Markov Chain, i.e., the next state is dependent on the current state. Thus, we propose the State-based Sequential Sample Model (SSM) as an alternative to RSM, to model the test behaviour, and to analyse the expected evaluation cost.

Definition 9. State-based Sequential Sample Model describes a scenario in which the samples from an instance of stowage planning are evaluated sequentially by the constraint evaluation engine. A constraint test ϕ_i is modelled by a Markov Chain with two states, namely T_i and F_i . Only one state can

Fig. 6. Schematic of State-based Sequential Sample Model. The constraint test has inner states composed of T_i and F_i . When the test is in the T_i state, it will always return “True”. Similarly, it returns “False” when it is in the F_i state. After evaluating each sample, the test may change state between T_i and F_i with the specified probabilities $p_{TT_i}, p_{TF_i}, p_{FT_i}, p_{FF_i}$.

be active at a time. The evaluation result of the constraint test is determined by the currently active state. If the state is T_i , then the evaluation result will be “True”, and similarly F_i state for “False”. After evaluating each sample, the state may change as follows:

$$\text{state}(\phi_i, t+1) = \begin{cases} T_i & \text{with prob } p_{TT_i} \text{ if state}(\phi_i, t) = T_i \\ F_i & \text{with prob } p_{TF_i} \text{ if state}(\phi_i, t) = T_i \\ T_i & \text{with prob } p_{FT_i} \text{ if state}(\phi_i, t) = F_i \\ F_i & \text{with prob } p_{FF_i} \text{ if state}(\phi_i, t) = F_i \end{cases} \quad (8)$$

Fig. 6 shows the schematic of SSM. The state transition mechanism in SSM is for modelling the test pattern for a constraint test $\phi_i \in \Phi$. For instance, in the stack weight limit example, moving from bottom of the stack to the top of the stack may correspond to a state change from T_i to F_i . Similarly, a state change from F_i to T_i corresponds to the instant when the stowage algorithm moves to an empty stack. The transitions between the states are modelled using probabilities to capture the fact that we do not assume any additional knowledge of, or control over, the behaviour of the stowage planning algorithm.

The purpose of SSM is twofold: it first captures the dependencies between consecutive test samples at a constraint test ϕ_i , and secondly support the estimation of the “False” probability in the next time step. Recall that, in SSM, if the state of a constraint test ϕ_i is known, then the test’s output can be known deterministically. Conversely, if we know the test result, then we can know the current state of the constraint test. With the knowledge of the current state, we may then infer the probability of the constraint test’s state being in the F_i state through the state-transition probability. Precisely, we have the following:

$$\hat{f}(\phi_i, t) = \begin{cases} p_{TF_i} & \text{if state}(\phi_i, t) = T_i \\ p_{FF_i} & \text{otherwise} \end{cases} \quad (9)$$

where $\hat{f}(\phi_i, t)$ denotes the probability of constraint test ϕ_i evaluating to “False” at time t .

With the sequential sample model, we can get a more precise estimation of “False” probabilities, and thus enable the ordering of constraint tests dynamically based on their dynamic cost-effectiveness.

Definition 10. The dynamic cost-effectiveness of a constraint test ϕ_i at time t is given by

$$\theta_i(t) = \frac{\hat{f}(\phi_i, t)}{\text{cost}(\phi_i)} \quad (10)$$

where $\hat{f}(\phi_i, t)$ denotes the probability of constraint test ϕ_i evaluating to “False” at time t , and is defined in Eq.(9).

At any time t , the SB strategy orders the constraint tests in non-increasing (or decreasing) order of dynamic cost-effectiveness:

$$\Phi^{SB}(t) = (\phi_1^{SB}(t), \phi_2^{SB}(t), \phi_3^{SB}(t), \dots, \phi_N^{SB}(t))$$

where $\theta_i(t) \geq \theta_j(t)$ for $i < j$.

We show that SB strategy is optimum with respect to SSM.

Proposition 2. Suppose that we are given a constraint test sequence $\Phi = \{\phi_1, \phi_2, \dots, \phi_N\}$ that follows SSM. Also assume that at time step t , the test results of the tests ϕ_i , for $i \in [1, N]$, are known. If the sequence is ordered dynamically using the SB strategy at time t after the evaluation phase, then $E[C(\Phi^{SB}(t+1))] \leq E[C(\Phi^X(t+1))]$ for any strategy X at time $t+1$.

Proof. Omitted. The same technique used in the proof of Proposition 1 can be applied directly. $\square$

Proposition 3. Suppose that we are given a constraint test sequence $\Phi = \{\phi_1, \phi_2, \dots, \phi_N\}$ that follows SSM. If the sequence is ordered dynamically using the SB strategy at all times, then $E[C(\Phi^{SB})] \leq E[C(\Phi^X)]$ for any strategy X .

Proof. The expected cost of a dynamic strategy Y can be calculated by:

$$E[C(\Phi^Y)] = \lim_{M \rightarrow \infty} \sum_{t=1}^M E[C(\Phi^Y(t))] \quad (11)$$

Since the expectation is a linear summation of the expected costs of the different sequences at each time step, we can treat the sequence at each time step independently. As shown previously in Proposition 2, ordering the sequence in non-increasing order of cost-effectiveness, at any time step t' , results in the minimum expected cost. Hence, the sum of expected costs across all times t is also the minimum. $\square$

4) State-estimation-based (SEB) Strategy: While it is desirable to use the SB strategy since it guarantees optimum expected evaluation cost w.r.t. SSM, it is not always possible to know the real state of the constraint tests. This is because the constraint evaluation engine always stops evaluating the rest of the tests as soon as a “False” test is found, skipping over all subsequent tests in the sequence. Since the real state of a test can be determined only after evaluating the test, we do not know the true states of the tests which were skipped. Fig. 7 shows an example.

In order to tackle this problem, we revise the constraint evaluation engine as follows. To evaluate a given sample (container-location pair), the evaluation engine will proceed

Fig. 7. An example of the states of the constraint tests. The evaluation stops at ϕ_i because the first “False” result has been seen. The states of the remaining tests cannot be determined as they are not evaluated.

as usual and start evaluating the sequence of tests until the first “False” is encountered. After that, instead of stopping immediately, the evaluation engine continues and uses a low cost operation to estimate the “False” probabilities of the remaining tests in the sequence. The “False” probabilities for the next time step are calculated as follows:

$$\tilde{f}(\phi_i, t+1) = \begin{cases} p_{TF_i} & \text{if } \phi_i \text{ returned “True” at } t \\ p_{FF_i} & \text{if } \phi_i \text{ returned “False” at } t \\ \tilde{f}(\phi_i, t)p_{FF_i} + (1 - \tilde{f}(\phi_i, t))p_{TF_i} & \text{if } \phi_i \text{ is not evaluated at } t \end{cases} \quad (12)$$

We note that if a constraint test is not evaluated consecutively for a sufficiently large number of rounds, $\tilde{f}(\phi_i, t)$ converges to $f(\phi_i)$. This also means that, if we lose track of the state of a constraint test, the constraint test model falls back to RSM. In the following, we prove the convergence of $\tilde{f}(\phi_i, t)$.

Proposition 4. Suppose that we are given the state transitions in the following form,

$$\begin{bmatrix} \tilde{p}(\phi_i, t+1) \\ \tilde{f}(\phi_i, t+1) \end{bmatrix} = \begin{bmatrix} p_{TT_i} & p_{FT_i} \\ p_{TF_i} & p_{FF_i} \end{bmatrix} \begin{bmatrix} \tilde{p}(\phi_i, t) \\ \tilde{f}(\phi_i, t) \end{bmatrix} \quad (13)$$

,where $\tilde{p}(\phi_i, t) = 1 - \tilde{f}(\phi_i, t)$. For sufficiently large τ , $\tilde{f}(\phi_i, \tau)$ converges to certain value f_i . In addition, f_i is related to the transition probabilities p_{TT_i} , p_{TF_i} , p_{FT_i} , and p_{FF_i} by

$$f_i = \frac{p_{TF_i}}{p_{TF_i} + p_{FT_i}} \quad (14)$$

where $p_i = 1 - f_i$, $p_{TT_i} = 1 - p_{TF_i}$, and $p_{FF_i} = 1 - p_{FT_i}$.

Proof. If $\tilde{f}(\phi_i, \tau)$ converges, then $\tilde{f}(\phi_i, \tau+1) = \tilde{f}(\phi_i, \tau)$. Then, from Eq.(13), we have the following system of linear equations:

$$\begin{cases} p_{TT_i} \cdot \tilde{p}(\phi_i, \tau) + p_{FT_i} \cdot \tilde{f}(\phi_i, \tau) = \tilde{p}(\phi_i, \tau) & (15) \\ p_{TF_i} \cdot \tilde{p}(\phi_i, \tau) + p_{FF_i} \cdot \tilde{f}(\phi_i, \tau) = \tilde{f}(\phi_i, \tau) & (16) \\ \tilde{p}(\phi_i, \tau) + \tilde{f}(\phi_i, \tau) = 1 & (17) \end{cases}$$

Using Eq.(15) and (16), we get

$$\tilde{p}(\phi_i, \tau) \left(p_{TF_i} + (p_{FF_i} - 1) \frac{1 - p_{TT_i}}{p_{FT_i}} \right) = 0 \quad (18)$$

For the convergence to be valid, we need $p_{TF_i} + (p_{FF_i} - 1) \frac{1 - p_{TT_i}}{p_{FT_i}} = 0$, which is always true by definition. Then, by solving the equation system, we get $p_i = \frac{p_{TF_i}}{p_{TF_i} + p_{FT_i}}$, and $f_i = \frac{p_{TF_i}}{p_{TF_i} + p_{FT_i}}$. $\square$

With the changes in the “False” probabilities, the estimated dynamic cost-effectiveness for a constraint test in SEB is then given by

$$\tilde{\theta}_i^{SEB}(t) = \frac{\tilde{f}(\phi_i^{SEB}, t)}{\text{cost}(\phi_i^{SEB})} \quad (19)$$

Similar to the SB strategy, the constraint test sequence is always sorted according to the estimated dynamic cost-effectiveness. Algorithm 1 describes the revised constraint evaluation engine for the SEB strategy. In Line 9, the algorithm updates the $\tilde{f}(\phi_i^{SEB})$ of the constraint tests that are not evaluated in the current round by using Eq.(12).

Algorithm 1: SEB constraint evaluation engine.

Data: Constraint test sequence $\Phi^{SEB}(t)$, container r , location l
Result: TRUE if all constraints are satisfied, FALSE if at least one constraint is violated.

```

1  $i \leftarrow 0$  ;
2  $violated \leftarrow \text{FALSE}$  ;
3 while ( $i < \text{numConstraint}$ )  $\wedge$  ( $\text{!}violated$ ) do
4    $result \leftarrow \text{evaluate}(\phi_i^{SEB}(t), r, l)$  ;
5    $i \leftarrow i + 1$  ;
6   if  $result = \text{FALSE}$  then
7      $violated \leftarrow \text{TRUE}$  ;
8 while  $i < \text{numConstraint}$  do
9    $\text{update}(\phi_i^{SEB}(t))$  ;
10   $i \leftarrow i + 1$  ;
11  $\Phi^{SEB}(t+1) \leftarrow \text{sort } \Phi^{SEB}(t) \text{ in non-increasing order of } \tilde{\theta}_i^{SEB}(t)$  ;
12 return  $violated$  ;
```

IV. EXPERIMENTS

A. Experiment Setup

We conducted experiments on data obtained from real stowage planning instances to compare the effectiveness of the re-ordering strategies described in Section III. Fig. 8 shows the schematic of the experiments. The experiments consist of a data collection phase and a strategy evaluation phase. In the data collection phase, for each container-location pair evaluated by the constraint evaluation engine, we recorded the computational costs¹ and the outcome of each constraint test without stopping at the first “False”. The collected data is then divided into training set and evaluation set respectively. The training set and evaluation set are each composed of real data from 244 different instances of stowage planning problem. Each instance is composed of iterations of constraint evaluations. In the strategy evaluation phase, the training set is used to obtain the model parameters p_i , f_i , cost , and the transition probabilities. The parameters are used for constructing the test case generator as well as input for the strategies to reorder the test sequence. By using the historical log of all of the results and computational costs, we can replay the results to simulate

Fig. 8. Design of experiments to evaluate the effectiveness of different strategies in reducing the evaluation costs.

the constraint evaluation engine and compare the effectiveness of the strategies in reducing the computational costs.

B. Speedup Comparisons

The simulation results are compared in Fig. 9. We compare the speedup achieved by the proposed strategies with respect to a baseline strategy that randomly permutes the test sequence only once at the beginning of the entire experiment (denoted as **RND**). Both the baseline strategy (RND) and the theoretically optimum strategy (SB) for SSM are drawn in dotted lines. Static strategies are shown in dot-dash lines and dynamic strategies are shown in solid lines. As shown in the figure, the SB and SEB strategies are consistently the best while RF and RFR (almost overlapped) comes next. The dynamic reordering strategies consistently outperform the static strategies.

We compare the overall speed up achieved by the strategies in Fig. 10. The speed up achieved by a strategy X is defined as the total cost incurred by the RND strategy divided by the total cost incurred by the strategy X . The total costs are the sum of the evaluation costs in all instances, including both the training and evaluation set. We note that, as the dynamic strategies also introduces logic overhead after evaluating each sample, we also compare the speed up achieved when taking the overhead into consideration. For SB and SEB strategies, the overhead is significant because of the sorting operation introduced at each

Fig. 9. Comparison of total cost incurred (normalized w.r.t. RND) in the constraint evaluation engine by the reordering strategies. The RND strategy randomly permutes the test sequence once at the beginning. The column labelled 'Generated' are the accumulated costs for evaluating 10,000,000 iterations of constraint tests whose results are generated by using the parameters learnt from the training set. The SB and SEB strategies are consistently the best while RF and RFR (almost overlapped) strategies comes next.

Fig. 10. Comparison of speed up achieved by the strategies. Dark grey bars represent the speed up achieved by each strategy without considering the logic overhead while the light grey bars represent the speed up with logic overhead introduced. Without considering the logic overhead introduced by the dynamic strategies, SB and SEB achieves the best speed ups. With the logic overhead, SB is still the best strategy while SEB is ranked slightly behind RF and RFR.

step. Without considering the logic overhead introduced by the dynamic strategies, SB and SEB achieves the best speed ups at 2.98 and 2.74 speed up respectively. With the logic overhead, SB is still the best strategy while SEB is ranked slightly behind RF and RFR strategies, at 2.17, 2.23, and 2.26 times speed up respectively.

C. Correlation Between Costs and Probabilities

From Fig. 9, RF and RFR seem to have efficiency comparable to that of SEB. This may suggest that the RF and RFR strategies are superior as they do not need any information on the tests' costs and probabilities. However, we note that this may not be true in general. This is because in our test cases, the constraint tests that are frequently violated (high "False" probabilities) typically also have low costs, which means that these tests are highly cost-effective. In other words, the costs and the "False" probabilities of the tests, p_{FF_i} , are negatively correlated. As shown in Proposition 3, placing cost-effective tests at the front leads to low expected overall costs, and hence RF and RFR are very effective when costs and p_{FF_i} 's are negatively correlated. If the costs and p_{FF_i} 's are positively correlated, then RF and RFR will have worse performance.

¹The experiments are written in Java language and conducted on HP Z400 Workstation with Intel(R) Xeon(R) CPU W3565 @3.20GHz (8CPUs); 12288MB RAM; 64-bit Windows 7.

Fig. 11. Comparison of total cost incurred by varying the correlation between p_{FF_i} and $\text{cost}(i)$. The costs and f_i 's are fixed while p_{TF_i} are generated according to Eq.(14). Static strategies have constant total cost as they are not dependent on p_{FF_i} . SB and SEB consistently have the lowest total costs except for the last entry. RF and RFR strategies have the total costs growing with respect to the cost-to- p_{FF_i} correlation.

To verify the impact of cost-to- p_{FF_i} correlation on the performance of RF and RFR, we can construct test cases with controlled cost-to- p_{FF_i} correlation. In order to make the generated test sequences to 'resemble' the training set for comparability, we retain the costs and f_i 's parameters of the tests. By using the Cholesky decomposition and the result from Proposition 4, we construct test cases where the cost-to- p_{FF_i} correlation is targeted to lie at the range from -0.9 to 0.9 . After generating the p_{FF_i} parameters, we generate p_{TF_i} correspondingly by using Eq.(14). The results are plotted in Fig. 11.

As shown in Fig. 11, RF and RFR strategies show a general trend that the total costs increase with respect to the cost-to- p_{FF_i} correlation. Specifically, for negative correlations, RF and RFR are among the best strategies; for positive correlations, RF and RFR are among the worst strategies.

D. Performance of SEB in Individual Instances

While the overall speedup obtained by SEB is substantial, it is possible that the strategy could perform badly in individual instances, and only achieves the observed speedup through an aggregated or batched execution. Thus, we also investigate the number of instances in which the performance of SEB is better, when compared to the other strategies. Here, we consider three main strategies, namely RF, CE, and RND, as they are representative of the respective categories (dynamic, static, and baseline respectively). The results are tabulated in Table II. Each row shows the comparison result of SEB w.r.t. a strategy. The number in each cell gives the number of instances in which SEB is better when compared to the other strategy. The result shows that SEB outperforms RF by a small margin, and outperforms CE as well as RND in almost all of the instances.

V. DISCUSSIONS

Given the prevalence of multicore processors and GPUs today, another way to improve the efficiency of the constraint eval-

TABLE II
THE NUMBER OF INSTANCES IN WHICH SEB RESULTED IN LOWER TOTAL COSTS WHEN COMPARED TO OTHER STRATEGIES.

Strategy Comparison	Training Set	Evaluation Set
SEB vs. RF	127 (52.0%)	134 (55.0%)
SEB vs. CE	237 (97.1%)	241 (98.8%)
SEB vs. RND	242 (99.2%)	241 (99.2%)

uation engine may be to run the constraint tests in parallel. However, there may often be cases where the number of tests to be conducted is larger than the available parallelism. In such cases, we need to run multiple constraint tests at each processor. Thus, we still need to determine the order of evaluation at each processor, and hence our proposed approach is relevant.

The dynamic approach utilizes the observation that the container-location pairs being considered by the stowage algorithm follow certain pattern. This follows from the fact that the stowage solution (or partial solution) is built incrementally by the algorithm. In other words, the next partial solution is dependent on the current partial solution. As a result, we used first order Markov Chain to formulate the constraint tests' behaviour.

We conjecture that the proposed strategies may work for other types of algorithms that share a similar behavior, i.e., the partial solution is built incrementally while ensuring that multiple constraints are satisfied. The formulation of costs and probabilities then follows directly as presented in Section III. Our proposed dynamic approach is less likely to be useful in situations where constraint evaluations do not follow any patterns. Nevertheless, we have demonstrated the success of our technique on the important problem of stowage planning.

VI. CONCLUSION

We have presented two models for the constraint test optimization problem, namely Random Sample Model (RSM) and State-based Sequential Sample Model (SSM). For each model, we considered several strategies to achieve the optimal performance. For SSM, as it is unrealistic to assume full knowledge about the states of the constraint tests, we showed how to modify the optimal strategy SB to produce a near optimal strategy SEB. We showed empirically that SEB performs nearly as well as SB and achieves a speed up of 2.74, as compared to the random ordering strategy. For RF and RFR, we also show that they are sensitive to the cost-to- p_{FF_i} correlation. Specifically, with positive correlation, RF and RFR strategies become less effective.

By improving the efficiency of the constraint evaluation engine, we also improve the efficiency of the stowage planning algorithm. As a result, we may now generate a good feasible stowage plan with comparable quality to that of the human planners in around 3 minutes time. For the detail of the stowage algorithm, please refer to [8, 10, 11, 32].

We focused on describing the constraint test optimization in the theme of automated stowage planning problem. However,

the technique described can be applied to problems in other domains as well. Most heuristics for solving combinatorial optimization problem explore the search space in a certain manner, which may result in the recurring constraint violation patterns as observed in the stowage algorithm we considered, and thus our results can be applied directly.

Lastly, the evaluation costs need not necessarily be computational cost. For instance, to check if a cargo contains illegal substances, a series of tests (x-ray detectors, gamma-ray detectors, various sensors, etc) can be conducted [2, 26, 27]. Each test incurs a different cost and has a different probability of success. The test reordering strategies may be adopted to such scenario to reduce the time and monetary cost to detect violating substance.

ACKNOWLEDGMENT

The authors gratefully acknowledge the grants from the NOL Fellowship programme as well as the co-funding from the Singapore Maritime Institute (SMI), and also the editors and anonymous reviewers for their constructive feedbacks, which have helped to improve the paper.

REFERENCES

- [1] W. Zhu, H. Qin, A. Lim, and H. Zhang, "Iterative deepening a* algorithms for the container relocation problem," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, pp. 710–722, Oct 2012.
- [2] E. A. Elsayed, C. M. Young, M. Xie, H. Zhang, and Y. Zhu, "Port-of-entry inspection: Sensor deployment policy optimization," *IEEE Transactions on Automation Science and Engineering*, vol. 6, no. 2, pp. 265–276, April 2009.
- [3] B. Cai, S. Huang, D. Liu, S. Yuan, G. Dissanayake, H. Lau, and D. Pagac, "Multiobjective optimization for autonomous straddle carrier scheduling at automated container terminals," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 3, pp. 711–725, July 2013.
- [4] L. Zhen, L. H. Lee, E. P. Chew, D. F. Chang, and Z. X. Xu, "A comparative study on two types of automated container terminal systems," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 1, pp. 56–69, Jan 2012.
- [5] X. Liu, H.-Q. Ye, and X.-M. Yuan, "Tactical planning models for managing container flow and ship deployment," *Maritime Policy & Management*, vol. 38, no. 5, pp. 487–508, 2011.
- [6] D. Ambrosino, A. Sciomachen, and E. Tanfani, "Stowing a containership: the master bay plan problem," *Transportation Research Part A: Policy and Practice*, vol. 38, no. 2, pp. 81 – 99, 2004.
- [7] F. Liu, M. Y. H. Low, W. Hsu, S. Huang, M. Zeng, and C. Win, "Randomized algorithm with tabu search for multi-objective optimization of large containership stowage plans," in *Computational Logistics*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, vol. 6971, pp. 256–272.
- [8] M. Low, M. Zeng, W. Hsu, S. Y. Huang, F. Liu, and C. A. Win, "Improving safety and stability of large containerships in automated stowage planning," *Systems Journal, IEEE*, vol. 5, no. 1, pp. 50–60, March 2011.
- [9] X. Xiao, M. Y. H. Low, F. Liu, S. Y. Huang, W. J. Hsu, and Z. Li, "An efficient block-based heuristic method for stowage planning of large containerships with crane split consideration," in *The International Conference on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.
- [10] C. A. Win, M. Y. H. Low, S. Y. Huang, W.-J. Hsu, F. Liu, and M. Zeng, "Visualization and simulation tool for automated stowage plan generation system," in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol II (IMECS 2010)*, 2010.
- [11] M. Zeng, M. Low, W. Hsu, S. Y. Huang, F. Liu, and C. A. Win, "Automated stowage planning for large containerships with improved safety and stability," in *Simulation Conference (WSC), Proceedings of the 2010 Winter*, Dec 2010, pp. 1976–1989.
- [12] M. Avriel, M. Penn, and N. Shpirer, "Container ship stowage problem: complexity and connection to the coloring of circle graphs," *Discrete Applied Mathematics*, vol. 103, no. 13, pp. 271 – 279, 2000.
- [13] D. Ambrosino, A. Sciomachen, and E. Tanfani, "A decomposition heuristics for the container ship stowage problem," *Journal of Heuristics*, vol. 12, no. 3, pp. 211–233, 2006.
- [14] D. Ambrosino, D. Anghinolfi, M. Paolucci, and A. Sciomachen, "A new three-step heuristic for the master bay plan problem," *Maritime Economics and Logistics*, vol. 11, no. 1, pp. 98–120, 2009.
- [15] —, "An experimental comparison of different heuristics for the master bay plan problem," in *Experimental Algorithms*, ser. Lecture Notes in Computer Science, P. Festa, Ed. Springer Berlin Heidelberg, 2010, vol. 6049, pp. 314–325.
- [16] L. Cruz-Reyes, P. Hernandez H., P. Melin, H. Fraire H., and J. Mar O., "Constructive algorithm for a benchmark in ship stowage planning," in *Recent Advances on Hybrid Intelligent Systems*, ser. Studies in Computational Intelligence, O. Castillo, P. Melin, and J. Kacprzyk, Eds. Springer Berlin Heidelberg, 2013, vol. 451, pp. 393–408.
- [17] "Singapore port marine notice," The Maritime and Port Authority of Singapore (MPA), April 2013, accesible at: "http://www.mpa.gov.sg/sites/circulars_and_notices/

- pdfs/port_marine_notices/pn13-47.pdf" (Accessed 25 March 2015).
- [18] D. Ambrosino, M. Paolucci, and A. Sciomachen, "Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem," *Flexible Services and Manufacturing Journal*, vol. 27, no. 2, pp. 263–284, 2015.
 - [19] F. Parreo, D. Pacino, and R. Alvarez-Valdes, "A GRASP algorithm for the container stowage slot planning problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 94, pp. 141 – 157, 2016.
 - [20] D. Ding and M. C. Chou, "Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts," *European Journal of Operational Research*, vol. 246, no. 1, pp. 242 – 249, 2015.
 - [21] D. Pacino, A. Delgado, R. M. Jensen, and T. Bebbington, "Fast generation of near-optimal plans for eco-efficient stowage of large container vessels," in *Computational Logistics: Second International Conference, ICCL 2011, Hamburg, Germany, September 19-22, 2011. Proceedings*. Springer Berlin Heidelberg, 2011, pp. 286–301.
 - [22] M. Garey, "Optimal task sequencing with precedence constraints," *Discrete Mathematics*, vol. 4, no. 1, pp. 37 – 56, 1973.
 - [23] D. Berend, R. Brafman, S. Cohen, S. Shimony, and S. Zucker, "Optimal ordering of independent tests with precedence constraints," *Discrete Applied Mathematics*, vol. 162, no. 0, pp. 115 – 127, 2014.
 - [24] D. Berend, S. Cohen, S. E. Shimony, and S. Zucker, *Modelling, Computation and Optimization in Information Systems and Management Sciences: Proceedings of the 3rd International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences - MCO 2015 - Part I*. Cham: Springer International Publishing, 2015, ch. Optimal Ordering of Tests with Extreme Dependencies, pp. 81–92.
 - [25] U. Tonguç, "Sequential testing of complex systems: a review," *Discrete Applied Mathematics*, vol. 142, no. 13, pp. 189 – 205, 2004, boolean and Pseudo-Boolean Functions.
 - [26] E. Boros, E. Elsayed, P. Kantor, F. Roberts, and M. Xie, *Intelligence and Security Informatics: Techniques and Applications*. Springer Berlin Heidelberg, 2008, ch. Optimization Problems for Port-of-Entry Detection Systems, pp. 319–335.
 - [27] C. M. Young, M. Li, Y. Zhu, M. Xie, E. A. Elsayed, and T. Asamov, "Multiobjective optimization of a port-of-entry inspection policy," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 392–400, April 2010.
 - [28] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, Oct 2001.
 - [29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–511–I–518 vol.1.
 - [30] Z. Q. Lee, R. Fan, and W.-J. Hsu, "Towards real-time automated stowage planning - optimizing constraint test ordering," in *Computational Logistics: 7th International Conference, ICCL 2016, Lisbon, Portugal, September 7-9, 2016, Proceedings*, pp. 175–189.
 - [31] B. Alidaee, "Optimal ordering policy of a sequencing model," *Journal of Optimization Theory and Applications*, vol. 83, no. 1, pp. 199–205.
 - [32] Z. Q. Lee, R. Fan, and W.-J. Hsu, "Optimizing constraint test ordering for efficient automated stowage planning," in *Computational Logistics - 6th International Conference, ICCL 2015 Delft, The Netherlands, September 23-25, 2015. Proceedings*, 2015, pp. 343–357.
- Zhuo Qi Lee** received both the B.Eng. and Ph.D degrees in Computer Science from Nanyang Technological University, Singapore, in 2010 and 2015, respectively. He is currently a Research Fellow in School of Computer Science and Engineering, Nanyang Technological University. His research interests include computational logistics and machine learning. He was a member of the NTU Port and Shipping Automation Research Team (PASAR) which won a commendation award from the Next Generation Container Port (NGCP) Challenge co-organized by SMI and MPA in Singapore.
- Rui Fan** joined ShanghaiTech University as an associate professor, PI in January 2017. Prior to this he was an assistant professor at Nanyang Technological University, Singapore. He received his BSc in computer science and mathematics from Caltech in 2000, and his PhD in computer science from MIT in 2008. He was a postdoc at the University of Toronto in 2008, and a Viterbi postdoc at the Technion, Israel in 2010. His PhD dissertation received MIT's Sprowl's Award for best computer science theses. He also received two best student paper awards at the ACM Principles of Distributed Computing (PODC) conference. His research interests span a number of topics in parallel and distributed computing. He is especially interested in the design of efficient algorithms and algorithmic solutions to problems on large datasets. Applications areas which interest him include machine learning, energy efficient computing, and managing large scale systems. He also works on distributed algorithms and synchronization related issues, as well as computational lower bounds.
- Wen-Jing Hsu** is presently with the School of Computer Science and Engineering, Nanyang Technological University. At NTU, he has initiated and completed several large scale externally funded research projects. He has authored or co-authored many papers on parallel and distributed processing; among them four papers were nominated for Best Paper Awards (with two awarded). He has published a book *Anatomy of HCTS-a High-capacity Container Handling System for Mega Vessels*. He was previously Director of the Center of Advanced Information Systems, and Deputy Director of Financial Engineering at NTU. He also held positions on the faculty of Michigan State University, USA, and was a Visiting Scientist at the IBM T. J. Watson Research Center, Yorktown Heights, and the IBM Palo Alto Scientific Center.