Kavakeb, S, Nguyen, TT, Yang, Z and Jenkinson, I

Evolutionary fleet sizing in static and uncertain environments with shuttle transportation tasks - the case studies of container terminals

Evolutionary fleet sizing in static and uncertain environments with shuttle transportation tasks - the case studies of container terminals

Shayan Kavakeb*, Trung Thanh Nguyen*, *Member, IEEE*, Zaili Yang and Ian Jenkinson

***Abstract* -** This paper aims to identify the optimal number of vehicles in environments with shuttle transportation tasks. These environments are very common industrial settings where goods are transferred repeatedly between multiple machines by a fleet of vehicles. Typical examples of such environments are manufacturing factories, warehouses and container ports. One very important optimisation problem in these environments is the fleet sizing problem. In real-world settings, this problem is highly complex and the optimal fleet size depends on many factors such as uncertainty in travel time of vehicles, the processing time of machines and size of the buffer of goods next to machines. These factors, however, have not been fully considered previously, leaving an important gap in the current research. This paper attempts to close this gap by taking into account the aforementioned factors. An evolutionary algorithm was proposed to solve this problem under static and uncertain situations. Two container ports were selected as case studies for this research. For the static cases, the state-of-the-art CPLEX solver was considered as the benchmark. Comparison results on real-world scenarios show that in the majority of cases the proposed algorithm outperforms CPLEX in terms of solvability and processing time. For the uncertain cases, a high-fidelity simulation model was considered as the benchmark. Comparison results on real-world scenarios with uncertainty show that in most cases the proposed algorithm could provide an accurate robust fleet size. These results also show that uncertainty can have a significant impact on the optimal fleet size.

***Index Terms* -** Evolutionary computation, uncertainty, shuttle transportation, intelligent vehicles.

I. Introduction

In this paper, we propose an evolutionary algorithm (EA) to identify the optimal number of vehicles in environments with shuttle transportation tasks (ESTTs). ESTTs are representative of industrial settings where goods are transferred repeatedly between multiple pickup-and-delivery points (PDPs) by a fleet of vehicles. At each PDP there is a machine to process the goods. Once goods have been processed, they will be picked up by vehicles and transferred to another machine for further processing. Next to each machine there might be a buffer, which is a limited space designed to temporarily store goods in a queue. The purpose of the buffer is to reduce waiting time. Vehicles can drop off goods in the buffer without having to wait for the machines to become available. Machines also can place the goods in the buffer for vehicles to collect later. ESTTs are very common in industrial applications. Typical examples are manufacturing factories, warehouses and container terminals (CTs) [1].

* Co-first authors. These authors contributed equally to this work.

Corresponding author: Trung Thanh Nguyen, email address: T.T.Nguyen@ljmu.ac.uk, Tel: +44 151 231 2006

S. Kavakeb, T.T. Nguyen, Z. Yang, and I. Jenkinson are with The School of Engineering, Technology and Maritime Operations, Liverpool John Moores University, Liverpool, L3 3AF, United Kingdom; emails: S.Kavakeb@ljmu.ac.uk, T.T.Nguyen@ljmu.ac.uk, Z.Yang@ljmu.ac.uk, and I.D.Jenkinson@ljmu.ac.uk.

One very important problem in an ESTT is the fleet sizing problem (FSP), of which the aim is to identify the optimal number of vehicles to transfer goods. Having too few vehicles may decrease performance while having too many vehicles is expensive and may introduce deadlocks[1]. This problem is not trivial. In real-world cases, it is highly complex and the optimal fleet size depends on many factors such as uncertainty in travel time of vehicles, the processing time of machines and size of the buffer. These factors, however, have not been fully considered previously, leaving an important gap in the current research. This paper attempts to close this gap by proposing an EA to solve the FSP by considering the above factors. The proposed algorithm will be tested on two case studies of container ports[2]. The motivation for choosing ports as the case studies is due to their vital role in international supply chains: 90% of the world's non-bulk trade goods were carried by containers as of 2007 [2].

Specifically, the outcome of this research will help answering the following questions for the first time: 1) How to determine the optimal/robust number of vehicles in static/uncertain ESTTs, especially CTs? 2) How to analyse the impact of uncertainty on the optimal number of vehicles? 3) What is the impact of the buffer size on the optimal/robust fleet size?

The novelty of this research can be summarised as follows: First, an EA is proposed to solve the FSP in ESTTs, with better performance than existing state-of-the-art methods. Second, a new formulation for the FSP is developed so that EA components can be built upon. Third, the following EA elements are developed: a representation, a local search, two operators and an adaptive learning mechanism. Fourth, uncertainty in the FSP in CTs is taken into account and solved. Two high-fidelity simulation models are also developed to serve as the benchmark for the EA in the uncertain cases. Finally, a set of test cases is developed using realistic data from European CTs to resolve the issue of the lack of benchmarks.

The rest of this paper is structured as follows. Sec. II describes the FSP in container ports. Sec. III discusses the gap of knowledge in research on the FSP in ESTTs. Sec. IV describes the proposed EA for the static case and its different components. In Sec. V, a combination of the proposed EA with a Monte Carlo (MC) simulation to determine the robust number of vehicles under uncertainty is described. The test cases are given in Sec. VI. The experimental results in the static case are presented in Sec. VII. Experiment results in the uncertain case are described in Sec. VIII. Finally, the conclusion is provided in Sec. IX.

II. Terminologies and problem descriptions

A. Basic terminologies and an integer programming model for the FSP

Here we explain some detailed concepts of the FSP, as adopted from [3].

1) Jobs and time windows: A job is defined as the process of moving a good from one PDP to another by a vehicle. For each job a time window $[a_j, b_j]$ is associated where a_i is the release time of job i from a PDP and b_i is the latest time to start job i. The value of b_i is calculated based on the release time of subsequent jobs of job i and the size of the buffer. For example, with a buffer of size n (i.e. the capacity of n goods) the due time for job i is calculated as: $b_i = a_{i+n}$. This means that job i should start before the release time of job $i+n$ to have at least one available slot for job $i+n$ in the buffer. To determine the exact pickup time of each job from the buffer, each time window needs to be discretised into multiple intervals, each with a duration of δ. The pickup time of a job is set at the beginning of one of the intervals.

[1]A situation in which one or more involved vehicles cannot move due to a lack of competent traffic control schemes or using too many vehicles etc.
[2]These two container ports have committed to consider the result of this research to improve their operations.

2) Compatible jobs: Two jobs are compatible if they can be done consecutively by one vehicle. Jobs i and j are compatible if one vehicle can pick up job i from a pickup point P_i at a time s_i and deliver it to a delivery point D_i, and then can still travel to a pickup point P_j to pick up job j within the time window of job j. Mathematically, i and j are compatible if $s_i + t_{P_i D_i} + t_{D_i P_j} \epsilon [a_j, b_j]$, where t_{PD} is the travel time from point P to point D and $[a_j, b_j]$ is the time window of job j.

3) Problem modelling: The FSP was modelled in [3] as a graph where each node represents one of the possible pickup times for a job. This graph has a source node from which all other nodes originate and a sink at which all other nodes terminate. Nodes that are compatible, i.e. nodes whose jobs can be done by the same vehicle, are connected by arcs. A set of connected arcs going from the source to the sink is called a path. Each path represents the sequence of jobs to be done by one vehicle. The total number of paths represents the total number of vehicles. The objective is to find the minimum number of paths which start from the source node and end at the sink node subject to the following constraints: 1) each job can start only once (this means that among all the possible pickup time nodes of a job, only one should be included in one of the paths); 2) each job cannot be done by more than one vehicle (this means that each node in the graph cannot be included in more than one path).

Fig. 1 shows an example of using a graph to model the solution of one simple FSP with three jobs and two vehicles. Job 1 has two possible pickup times, represented by nodes j_{11} and j_{12}. Job 2 has one pickup time node, j_{21}. Job 3 has two possible pickup time nodes, j_{31} and j_{32}. Node s is the source and node t is the sink. This solution consists of two paths (i.e. two vehicles) in which the path of vehicle 1 passes through j_{11} and j_{32} and the path of vehicle 2 passes only through j_{21}. Using this graph, Vis et al. [3] formulated the FSP as an integer programming (IP) problem. In this research, we reproduced this IP formulation and solved it using the CPLEX solver. The results will be used as a benchmark to compare with our algorithm.

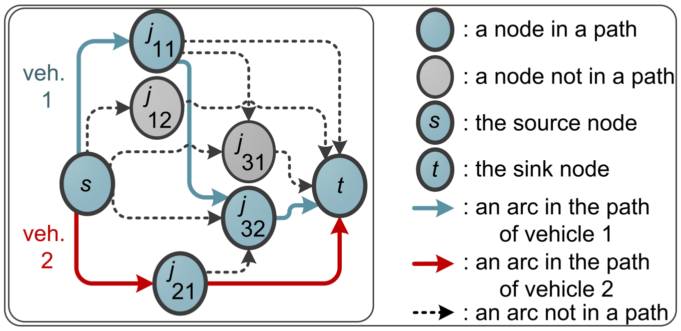


Fig. 1. The solution for an FSP problem with three jobs.

B. The FSP in container terminals (CTs)

In CTs, a fleet of vehicles is responsible for transporting containers between two areas named quay and stack areas. The quay area is the place where vessels are berthed and there are quay cranes (QCs) to discharge and load containers from/to vessels. The stack area consists of a number of stack blocks served by stack cranes (SCs). Each quay or stack area has a number of PDPs for vehicles to transfer containers. Each PDP is facilitated by one crane.

Once a vessel is berthed, the discharging and then loading operations are performed, usually separately. In the discharging phase, QCs unload containers from vessels and pass them to transfer vehicles which then transport them to the stack area.

If vehicles can pick up containers by themselves, QCs place the containers in the *buffer,* a place underneath the cranes, for vehicles to collect. Otherwise, the QCs have to wait for vehicles to come and then place the containers directly onto them. The capacity of the buffer is limited and it is required that containers should be picked up from the buffer before it gets full, i.e. there should always be at least one free slot available in the buffer. This is to minimise the expensive waiting times of QCs.

At the stack area, SCs collect containers from vehicles and place them in the stacks. After passing a container to a SC, the transfer vehicle will come back to the QCs to collect another container. The buffer for SCs is similar to the one for QCs: vehicles drop off containers in the buffer from which SCs pick them up. In this paper, it is assumed that there are always free slots available in the SC buffer for vehicles to drop off containers. After the discharging phase finishes, the loading phase will start. The loading tasks are similar to the discharging tasks but in the opposite direction. Due to such similarity, in this paper, we only consider the FSP for the discharging tasks.

III. Related literature

It should be noted that the FSP in ESTTs is very different from the fleet sizing and mixed vehicle routing problem (FS-VRP) and its variants, and hence requires a different solving approach. In the FS-VRP, the objective is to find the optimal routes for vehicles to minimise the total cost, which usually comprises the cost of routes and cost of vehicles. In contrast, the objective of the FSP in ESTTs is to assign the transportation tasks to an optimal number of vehicles to increase the total throughput of the system. In addition, the ESTTs in the FSP are different to the environments in the FS-VRP in that vehicles have to shuttle among the PDPs. Therefore, existing algorithms used for the FS-VRP may not be suitable for solving the FSP in ESTTs. In fact, so far none of the algorithms for the VRP have been applied to the FSP in ESTTs [4].

Existing approaches to solving the FSP in ESTTs can be grouped into three main categories [5]: 1) calculus-based methods, 2) stochastic and queuing network models, and 3) deterministic and meta-heuristic optimisation algorithms.

In the calculus-based approach, a series of straightforward equations and calculations are used to identify the optimal number of vehicles. This approach is mainly used to identify the optimal fleet size for flexible manufacturing systems (FMSs). For details, readers are referred to [6, 7, 8, 9, 10]. In the queuing network approach, ESTTs are modelled as queuing systems [5, 11, 12, 13] and the optimal number of vehicles is estimated using the queuing theory.

The deterministic and metaheuristic optimisation approaches model the FSP as an optimisation problem, and then solve it. An integer programming (IP) model was proposed [14] to determine the optimal number and type of vehicles in automated guided vehicle (AGV) systems. The objective function is the summation of the annualised operating cost of vehicles and the transportation cost of parts between the workstations. In [15], a two-phase algorithm was provided for the fleet sizing and routing of vehicles in the Port of Busan. In the first phase, a lower bound for the fleet size is produced by an optimisation algorithm. In the second phase, a Tabu search is applied to find the optimal routes of vehicles for the given fleet size. If all the transportation tasks cannot be done within the given makespan, the fleet size must be increased.

In [16], analytical and simulation models were provided to determine the fleet size in FMSs. Load handling, empty travelling, and waiting were considered to be different states. By analytically estimating the time that vehicles spend in these three states, an optimal fleet size was obtained and the results were validated by a simulation. A bee algorithm [17] was used to determine the optimal amount of equipment in manufacturing systems. This problem was modelled as a multi-periodic optimisation problem in which the objective is to maximise profits. Sinriech and Tanchoco [18] used goal programming to determine the optimal number of AGVs. A costs versus total throughput decision table showed different solutions with the trade-off ratios between the goals. In [19], a fleet sizing problem of AGVs in ports was modelled as a minimum flow problem. A strongly polynomial time algorithm was proposed to tackle this problem. This model, however, assumes that container release time and travel time are fixed, and there is no buffer under the crane. These assumptions may not be realistic in many CTs. This research was followed by [3], where an IP model was proposed for the fleet sizing of automatic lifting vehicles (ALVs). With ALVs, it is possible to use buffer areas underneath the cranes to reduce waiting time of cranes and vehicles. The authors solved this IP problem using the CPLEX commercial solver and validated results of the analytical model by conducting a simulation study.

To the best of our knowledge, there is no existing research that considers uncertainty in the FSP in ESTTs to offer a robust optimisation solution. However, robust optimisation is widely considered in solving other problems [20]. Among the robust optimisation research, the studies on robust scheduling and planning (RSP) are most relevant to the robust FSP in ESTTs because, in both classes of problems, the optimisation algorithms search in the space of job schedules to optimise an objective such as makespan, cost, fleet size, etc. Thus, here we review some research on RSP. Note that there are also some studies which tackle the FSP under uncertainty in other applications (e.g. military [21, 22, 23, 24, 25], maritime [26, 27, 28, 29] and rail [30, 31]), however, due to the differences between the nature of those problems and that of the FSP in ESTTs, we do not review these studies here. Shadrokh and Kianfar [32] proposed a genetic algorithm (GA) for the resource investment project scheduling problem (RIPSP). The objective is to optimise the sum of resource availability costs and tardiness penalty. Kılıç et al. [33] developed a bi-objective GA to minimise the makespan and total cost. Zuo et al. [34] developed a multi-objective immune scheduling algorithm with the optimality and robustness of schedules as the objectives. The authors used simulation to evaluate the robustness of solutions. Some papers [35, 36, 37] used the free slack as the robustness measure to identify the robust schedules. Xiong et al. [38] developed an EA to tackle flexible job-shop scheduling problems (FJSP) under the uncertainty of machine breakdowns. The authors proposed two measures to evaluate robustness based on the overlap between the machines' float time and their possible breakdown duration.

Mahdavi et al. [39] developed a real-time simulation-based decision support system for production control of the stochastic FJSP under stochastic processing time of machines. Wang and Yu [40] proposed a beam search-based heuristic algorithm to tackle the FJSP with machine availability constraints due to maintenance activities. Nguyen et al. [41] proposed genetic programming-based hyper-heuristic algorithms to identify non-dominated scheduling policies based on the makespan, normalised total weighted tardiness and mean absolute percentage error measures. Moradi et al. [42] developed a bi-objective GA to minimise simultaneously the makespan and system unavailability. A two-stage hybrid GA for the FJSP under random machine breakdowns was proposed in [43]. The authors defined a number of bi-objective measures combining the robustness and stability of schedules. Gu et al. [44] proposed a co-evolutionary quantum GA for the stochastic FJSP to optimise the expected makespan. Sevaux and Sörensen [45] modified the GA to compute robust machine schedules. New robustness measures were defined to evaluate solutions based on the robustness and distance to the baseline solutions. Xiong et al. [46] modelled capability planning problems as a multi-mode RIPSP. The makespan and cost were considered to be the two objectives.

The literature review above reveals three possible gaps in the current research. First, despite the supposed usefulness of buffers in PDPs in reducing waiting time in ESTTs, very few existing methods actually consider buffers in the FSP. Second, most of the existing deterministic/metaheuristic optimisation methods for the FSP in ESTTs do not consider the impact of uncertainty on the optimal fleet size. Third, there is a clear lack of computational intelligence techniques, particularly EAs, in solving the FSP in ESTTs. Given that EAs have been widely used to find approximated solutions for complex, non-linear, large-scale problems in both static and uncertain/dynamic cases [47, 48, 49], it is of interest to investigate how EAs can be used to address the aforementioned gaps. This will be the purpose of this paper.

IV. An evolutionary algorithm approach for static environments

We develop an EA named Fleet-sizing for Shuttle-transport environments Evolutionary Algorithms (FSEA) to address the aforementioned gaps in existing literature, namely dealing with larger-scale situations, handling uncertainty, and investigating the use of buffers in static and uncertain situations. In this section, a version of FSEA for static environments will be explained.

A. *A chromosome representation*

Given the graph representation in II-A3 (Fig. 1), each chromosome of FSEA is a solution comprising a number of paths (sequences of jobs to be done by one vehicle). Each chromosome is represented as a string of pairs $P_i =< x_i, y_i >, i = 1, ..., n$ where x_i represents the pickup time for job i; and y_i represents the chosen pickup time for the next job that will be done by the same vehicle that carries job i. Fig. 2 shows a general representation and an example of a chromosome.

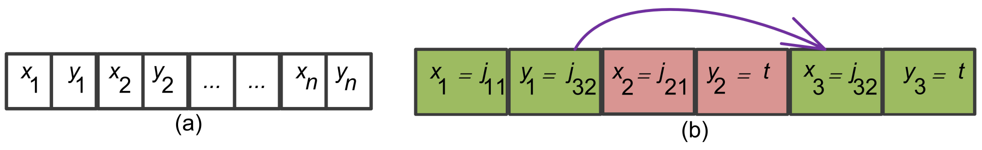

x_1 y_1 x_2 y_2 ... ... x_n y_n
(a)
$x_1 = j_{11}$ $y_1 = j_{32}$ $x_2 = j_{21}$ $y_2 = t$ $x_3 = j_{32}$ $y_3 = t$
(b)

Fig. 2. (a) a general representation of a chromosome. (b) an example of how such a chromosome can encode the paths of vehicles in the example in Fig. 1. The two pairs in green represent the path for vehicle 1 and the pair in red represents the path for vehicle 2. Job 1 is linked with job 3 because both can be picked up by vehicle 1. Both y_1 and x_3 refer to the same value, j_{32}, which is the pickup time for job 3.

B. *A new problem formulation for the FSP*

This formulation is necessary for FSEA as it is used in the population initialisation to repair infeasible solutions and in the recombination to maintain the feasibility of solutions. Let:

x_i: represents the pickup time for job i; y_i: represents the chosen pickup time for the next job to be done by the same vehicle as job i; n: number of jobs, $n \epsilon \mathbb{N}$; t: the sink node; S_i: set of nodes that correspond to the different possible pickup times of job i; C_{x_i}: set of nodes that are compatible with node x_i;

$$e(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases} \qquad a, b \epsilon \mathbb{N};$$

and $b(y_i, y_j) = \begin{cases} 1 & \text{if } i \neq j \text{ and } \exists \text{job } k \text{ so that: } y_i \epsilon S_k \text{ and } y_j \epsilon S_k, \\ 0 & \text{otherwise.} \end{cases}$

Find x_i and y_i to:

$$Min \sum e(y_i, t). \tag{1}$$

Subject to:

$$x_i \epsilon S_i,\ y_i \epsilon C_{x_i},\ 1 \leq i \leq n, \tag{2}$$

$$\sum_{j=1}^{n}\sum_{i=1}^{n} b(y_j, y_i) = 0, \tag{3}$$

$$\sum_{j=1}^{n}\sum_{i=1}^{n} e(x_j, y_i) + \sum_{i=1}^{n} e(y_i, t) = n. \tag{4}$$

The objective function (1) counts the number of times the y_i variables are set as t (the sink node). y_i being set as t means y_i is at the end of the path of one vehicle. Accordingly, the value of the objective function (1) is the total number of paths, which is equivalent to the total number of vehicles. Const. (2) defines the domain range of the x_i and y_i variables. Const. (3) ensures that a job must be in only one path (see Fig. 3). If a job j is included in more than one path, j will be presented by more than one "y" variable in the chromosome. Const. (3) prevents this from happening by ensuring that, for each job that is not the first or the last in a path, there is one and only one "y" variable corresponding to the job (see Fig. 3). Const. (4) ensures that in all paths there is no more than one node referring to one job (see Fig. 3). In other words, each job must have only one node in only one path. The following example clarifies the meaning of this constraint. Consider one path where job j is set to be transported after job i by the same vehicle. It means that y_i refers to one of the possible pickup time nodes of job j. The path is only feasible if x_j is equal to y_i, i.e. both x_j and y_i refer to the same pickup time for job j if job i is not the last job in the path. It means $e(x_j, y_i) = 1$. Job i can only be picked up once, so $\sum_{i=1}^{n} e(x_j, y_i)$ must be equal to 1. Let $m < n$ be the number of jobs that are not the last jobs in their paths, then $\sum_{j=1}^{n}\sum_{i=1}^{n} e(x_j, y_i)$ must be equal to m. Let $k = n - m$ be the number of jobs that are the last in their paths, then $\sum_{i=1}^{n} e(y_i, t)$ must be equal to k. Take the summation: $\sum_{j=1}^{n}\sum_{i=1}^{n} e(x_j, y_i) + \sum_{i=1}^{n} e(y_i, t)$, we have $\sum_{j=1}^{n}\sum_{i=1}^{n} e(x_j, y_i) + \sum_{i=1}^{n} e(y_i, t) = m + (n - m) = n$. This is Const. (4).

C. *Initialisation and repair*

At the start of the algorithm, each variable is initialised with a random value. Most probably, the initialised solutions are not feasible. To repair such infeasible solutions, we developed a repair operator. Note that Const. (2) is never violated, since individuals are initialised within the domain ranges given in this constraint.

The repair operator repairs violations of Consts. (3) and (4) as follows. First, the repair operator checks the violation of Const. (3). If it is violated, at least one job is placed in more than one path. Fig. 3 (a) shows an example of such a violation.

In order to repair individuals regarding this violation, the duplicated jobs are removed and replaced by the sink node in all but one of the violated paths. In the example in Fig. 3 (a), job 3 must be removed from one of the paths, e.g. the path of vehicle 2, and be kept in the path of vehicle 1. Consequently, this violation is repaired (Fig. 3 (c)).

The individuals must then be checked regarding the violation of Const. (4). If this constraint is violated it means that a path has two different nodes corresponding to the same job (Fig. 3 (b)). In the example in Fig. 3, two different pickup time nodes of job 3 (j_{31} and j_{32}) are in the path of vehicle 1, hence this is a violation of Const. (4). To repair the violation, the repair operator checks whether x_3 (j_{31}) in pair $< x_3, y_3 >$ of job 3 is compatible with x_1 (j_{11}) in pair $< x_1, y_1 >$. If this is the case, the repair operator changes the value of y_1 to j_{31}. Otherwise, it updates the value of y_1 with the sink node (t) to terminate the path of vehicle 1 at job 1 and starts a new path from job 3. For this type of violation, the repair operator always changes the y variables of the preceding job rather than the x variables of the violated job. This is because, if the repair operator changes the x variable of the violated job, the consecutive jobs might not be compatible and hence the rest of the path might become invalid. The pseudo-code of the repair procedure is shown in Alg. 1.

Algorithm 1 Repair(Individual $\vec{X}$)

```
1: Identify $ViolConst_3$, the set of jobs in individual $\vec{X}$ that violate Const. (3)
2: for all jobs $j \in ViolConst_3$
3:     Update all but one "$y$" variables corresponding to job $j$ with the sink node //keep one of the "$y$" variables unchanged randomly
4: Identify $ViolConst_4$, the set of triples $< x_i, y_i, x_j >$ that violate Const. (4)
5: for all triples $< x_i, y_i, x_j > \in ViolConst_4$
6:     if $x_i$ is compatible with $x_j$ then Update $y_i$ with $x_j$
7:     else Update $y_i$ with the sink node
```

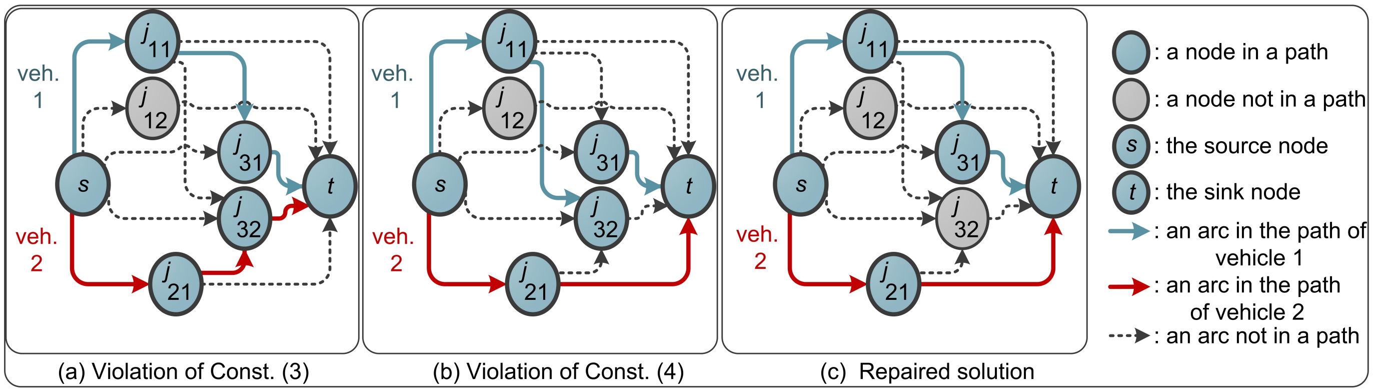


Fig. 3. Plot (a) shows an example of a solution violating Const. (3) (chromosome: [<x_1 = j_{11}, y_1 = j_{31}>, <x_2 = j_{21}, y_2 = j_{32}>, <x_3 = j_{31}, y_3 = t>]) and plot (b) shows a violation of Const. (4) where two different nodes of job 3 are placed in the path of vehicle 1 (chromosome: [<x_1 = j_{11}, y_1 = j_{32}>, <x_2 = j_{21}, y_2 = t>, <x_3 = j_{31}, y_3 = t>]). Plot (c) shows the repaired solution for these violations (chromosome: [<x_1 = j_{11}, y_1 = j_{31}>, <x_2 = j_{21}, y_2 = t>, <x_3 = j_{31}, y_3 = t>]).

D. Reproduction

We propose a reproduction method, which is a combination of a mutation operator and a local search. The mutation operator is used to create offspring and the local search is used to improve the fitness of each offspring.

1) Local search: Recall from II-A3 that (i) we model a solution as a graph containing multiple paths (job sequences) from the source to the sink and (ii) the total number of job sequences represents the total number of vehicles. Then, if we can reduce the number of job sequences in a solution, we will be able to reduce the fleet size. To reduce the number of job sequences, we can try to remove all jobs in one randomly chosen sequence (let us call it s_delete) and, if possible, insert them into the other sequences. This is equivalent to asking other vehicles to take over all the jobs originally assigned to the s_delete vehicle. If this can be done, the s_delete job sequence will disappear and the fleet size will be reduced by one (Alg. 2 and Fig. 4).

Algorithm 2 ReduceJobSequences()

```
1: for all jobs $j \in JobSeq_{s\_delete}$
2:     if job $j$ can be inserted into $JobSeq_l$ at position $k$ ($1 \le l \le FS, l \ne$ s_delete and $1 \le k \le length(JobSeq_l)$)
3:         insert job $j$ into $J_l$ at position $k$
4:         remove job $j$ from $JobSeq_{s\_delete}$
5: if all jobs are removed from $JobSeq_{s\_delete}$ //vehicle s_delete was removed from the fleet size successfully
6:     Randomly choose another vehicle to be s_delete
7: if any job is removed from $JobSeq_{s\_delete}$ then $\alpha := 0$
8: else   $\alpha := \alpha + 1$
9: adaptiveLearning()
```

where $JobSeq_l$ is the sequence of jobs for vehicle l, FS is the fleet size, α is the number of generations that $JobSeq_{s\_delete}$ has remained unchanged, and the adaptiveLearning() is our proposed learning method to help FSEA to get out of local minima (see Subsec. IV-E).

2) Mutation: There might be a local optimum situation where the local search cannot remove the chosen s_delete sequence, because there is no available place in the other sequences to further insert the remaining jobs from s_delete. We propose a mutation operator to help the algorithm escape such a situation. Instead of moving jobs from s_delete to other sequences, we try to move jobs among the other sequences, in the hope that this movement would change the structure of these sequences, which in turn would open up space into which jobs from s_delete can be slotted in (see Alg. 3 and Fig. 4).

Algorithm 3 Mutate()

```
1: Generate $m$, a random integer value //where $1 \le m \le FS$, $m \ne s\_$delete and $FS$ is the fleet size
2: for all jobs $j \in JobSeq_m$ //$JobSeq_m$ is the array of the sequences of jobs for vehicle $m$
3:     if job $j$ can be inserted into $JobSeq_l$ at position $k$ ($1 \le l \le FS, l \ne s\_$delete and $1 \le k \le length(JobSeq_l)$)
4:         insert job $j$ into $JobSeq_l$ at position $k$ and remove job $j$ from $JobSeq_m$
```

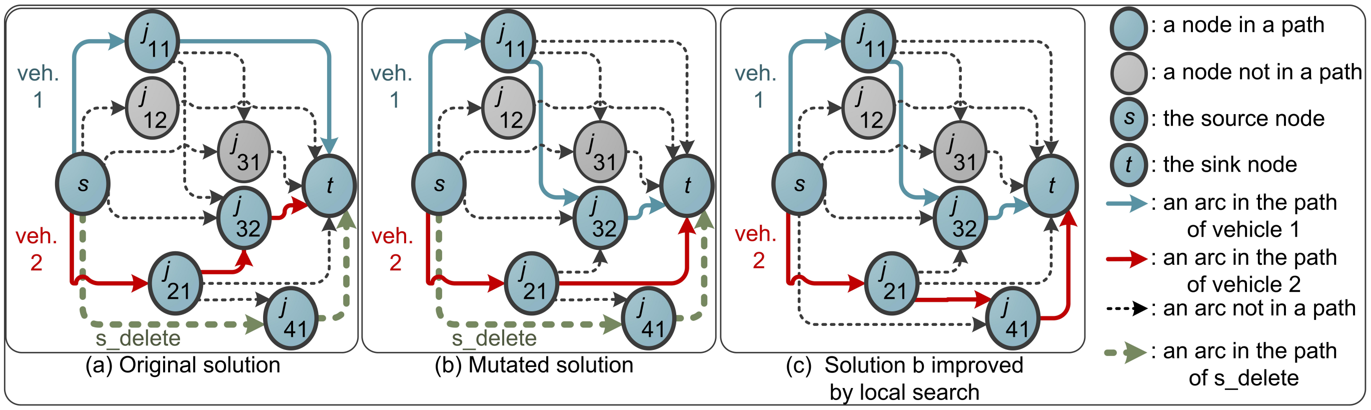


Fig. 4. In this example, if job 3 in the original solution (left plot) is moved from vehicle 2 to vehicle 1 thanks to the mutation operator (middle plot), job 4 can be moved by the local search from vehicle s_delete to vehicle 2 (right plot). As a result, we can delete vehicle s_delete and reduce the fleet size by one.

E. Adaptive learning method

As mentioned in Subsec. IV-D, the algorithm might be trapped in a local optimum where it is not possible to move the remaining jobs from a randomly chosen s_delete to any other sequences. Our experiments showed that some jobs are significantly more difficult to remove than others and they are the ones that normally remain in s_delete when the algorithm gets trapped. Worse, even if the algorithm is able to escape thanks to the mutation operator, later on it may still be likely to end up in a similar local optimum, where those most-difficult-to-remove jobs will again be the ones that remained in s_delete.

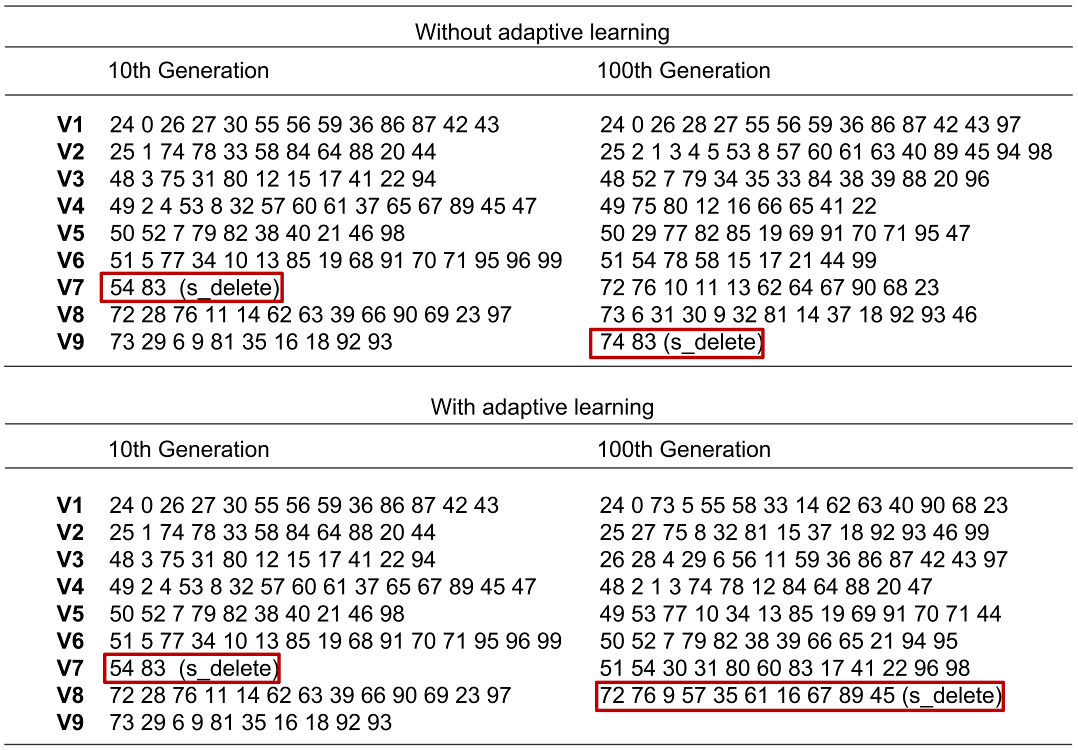
The best way to deal with this situation is to adaptively learn what are those most difficult jobs, remember them and then avoid removing any job sequence containing them when applying the local search ReduceJobSequences(). This process is similar to a Tabu search. This process is done in our proposed adaptiveLearning() procedure (Alg. 4). This procedure has two tasks: 1) identifying the most-difficult-to-remove jobs and 2) updating the index of s_delete to another job sequence (i.e. vehicle) if all the jobs of s_delete are the most-difficult-to-remove jobs.

To perform the first task, this procedure keeps track of the changes in the jobs of s_delete to identify the most-difficult-to-remove jobs. Recall from Alg. 2 that this procedure counts the number of times where s_delete remained unchanged during consecutive generations using α as the counter (Alg. 2, lines 7-9). Alg. 4 checks the value of α and, if $\alpha = \beta$ (β is the upper bound for α), Alg. 4 then considers the jobs of s_delete to be the most-difficult-to-remove jobs. In such cases it adds the jobs of s_delete to the set of most-difficult-to-remove jobs (i.e. $DifficultJobs$). The algorithm then changes the index of s_delete to another job sequence (i.e. vehicle) in the solution, because all jobs of s_delete are difficult to remove (Alg. 4, lines 1-4). Alg. 4 limits the size of $DifficultJobs$ to γ. When the size of this set reaches its limit, the algorithm randomly removes more jobs from this list until its size becomes r (a random integer value between 0 and γ) to make room for other potential jobs to become most-difficult-to-remove jobs (Alg. 4, lines 7-9). To perform the second task, Alg. 4 compares the jobs of s_delete with the elements of the $DifficultJobs$ set. If all the jobs of s_delete belong to this set, it is less likely that the jobs of s_delete can be removed; thus, this algorithm updates the index of s_delete with another job sequence, i.e. vehicle (Alg. 4, lines 5-6). An example of how the adaptive learning method helps FSEA to escape from local optima is in Fig. 5.

Algorithm 4 AdaptiveLearning()

```
1: if $\beta = \alpha$
2:     $DifficultJobs := DifficultJobs \cup JobSeq_{s\_delete}$
3:     $\alpha := 0$
4:     Update s_delete with a new random value between 1 and $FS$
5: if $JobSeq_{s\_delete} \subseteq DifficultJobs$
6:     Update s_delete with a new random value between 1 and $FS$ and $\alpha := 0$
7: if length($DifficultJobs$) $\geq \gamma$
8:     Generate $r$, a random integer value between 0 and $\gamma$
9:     Remove elements of $DifficultJobs$ randomly until its size becomes $r$
```

where $DifficultJobs$ is the list of difficult jobs, $JobSeq_{s-delete}$ is the sequence of jobs for s_delete, β is the maximum number of generations that $Jobs_{s\_delete}$ can remain unchanged, α counts the number of generations that $Jobs_{s\_delete}$ has remained unchanged, γ is the maximum size of $DifficultJobs$.



Without adaptive learning		
	10th Generation	100th Generation
V1	24 0 26 27 30 55 56 59 36 86 87 42 43	24 0 26 28 27 55 56 59 36 86 87 42 43 97
V2	25 1 74 78 33 58 84 64 88 20 44	25 2 1 3 4 5 53 8 57 60 61 63 40 89 45 94 98
V3	48 3 75 31 80 12 15 17 41 22 94	48 52 7 79 34 35 33 84 38 39 88 20 96
V4	49 2 4 53 8 32 57 60 61 37 65 67 89 45 47	49 75 80 12 16 66 65 41 22
V5	50 52 7 79 82 38 40 21 46 98	50 29 77 82 85 19 69 91 70 71 95 47
V6	51 5 77 34 10 13 85 19 68 91 70 71 95 96 99	51 54 78 58 15 17 21 44 99
V7	54 83 (s_delete)	72 76 10 11 13 62 64 67 90 68 23
V8	72 28 76 11 14 62 63 39 66 90 69 23 97	73 6 31 30 9 32 81 14 37 18 92 93 46
V9	73 29 6 9 81 35 16 18 92 93	74 83 (s_delete)

With adaptive learning		
	10th Generation	100th Generation
V1	24 0 26 27 30 55 56 59 36 86 87 42 43	24 0 73 5 55 58 33 14 62 63 40 90 68 23
V2	25 1 74 78 33 58 84 64 88 20 44	25 27 75 8 32 81 15 37 18 92 93 46 99
V3	48 3 75 31 80 12 15 17 41 22 94	26 28 4 29 6 56 11 59 36 86 87 42 43 97
V4	49 2 4 53 8 32 57 60 61 37 65 67 89 45 47	48 2 1 3 74 78 12 84 64 88 20 47
V5	50 52 7 79 82 38 40 21 46 98	49 53 77 10 34 13 85 19 69 91 70 71 44
V6	51 5 77 34 10 13 85 19 68 91 70 71 95 96 99	50 52 7 79 82 38 39 66 65 21 94 95
V7	54 83 (s_delete)	51 54 30 31 80 60 83 17 41 22 96 98
V8	72 28 76 11 14 62 63 39 66 90 69 23 97	72 76 9 57 35 61 16 67 89 45 (s_delete)
V9	73 29 6 9 81 35 16 18 92 93	

Fig. 5. Without adaptive learning FSEA got trapped in local minima with job 83 in s_delete. With adaptive learning (and remembering to avoid job 83), FSEA can get out of such local minima and hence reduce the fleet size by one at the 100th generation.

F. The pseudo-code for FSEA

So far, we have explained all the components of FSEA. Now, it is time to put together the components of FSEA and present the pseudo-code for the whole algorithm (Alg. 5). FSEA starts with the initialisation of the population (see Subsec. IV-C). It then evaluates the individuals based on the fleet size of each individual using (1). FSEA selects individuals for the next generation using the rank selection. Algs. 2 and 3 are then applied to individuals for possible improvement of their fitness. This loop will continue until the stopping criteria are met and the best found solution will be the optimal solution.

Algorithm 5 FSEA()

```
1: Initialise population $P_t$
2: Evaluate population $P_t$
3: while stopping criteria not met
4:     Select elements from $P_t$ to copy into $P_{t+1}$
5:     Mutate population $P_{t+1}$ by Mutate() (Alg. 3)
6:     Recombine population $P_{t+1}$ by ReduceJobSequences() (Alg. 2)
7:     Evaluate new population $P_{t+1}$
8:     $P_t := P_{t+1}$
9: return the best individual
```

V. Extensions of FSEA for uncertain environments

A. Uncertainty in ESTTs

We identified uncertainty in the processing time of machines and travel time of vehicles as two important types of uncertainty that have significant impacts on the optimal fleet size. The first type of uncertainty can be caused by variations in the quality of machine parts, skills of operators and so on. The second type of uncertainty can be caused by weather conditions, vehicle breakdowns, or congestions. Both types may render a static optimal fleet size ineffective. Due to that, we need to take these types of uncertainty into account by identifying a fleet size that is robust against uncertainty, so that if there is any change in the environment, the fleet size is still effective. We propose a MC approach to simulate and guide FSEA towards the most robust solutions. The proposed method will measure the quality robustness by which we can ensure the optimal fleet size is capable of performing all the transportation tasks without imposing any delay on the machines.

B. A MC simulation for the uncertainty in vehicle travel time

As explained in Sec. IV, the solutions (chromosomes) produced by FSEA contain not only the total number of vehicles but also the schedules (sequences) of jobs for each vehicle. It is possible that two individuals have the same number of vehicles but different schedules, i.e. sequences of jobs can be different for the same number of vehicles. In the static case FSEA evaluates fitness of individuals based on only the number of vehicles, regardless of the produced schedules. However, in the uncertain case such an evaluation may not be totally realistic. Different schedules for the same number of vehicles may not behave similarly under uncertainty. Some schedules may show more robustness against changes than others.

To evaluate the robustness of a schedule of vehicles, we propose a MC approach. The proposed MC evaluates the schedules of vehicles to answer the following questions: 1) How robust is the schedule of vehicles under uncertainty in travel time? 2) Is it necessary to add more vehicles to the system to reduce the possible waiting time of machines caused by this type of uncertainty? If those additional vehicles are needed, how many more should be added?

Before we explain the proposed MC, we will define some concepts: the frequency of vehicle disruptions and the time taken to resolve the disruptions. Different types of vehicle disruption can happen in ESTTs, such as breakdowns, collisions and deadlocks. The disruption rate ($\lambda(t)$) of a vehicle is defined as the frequency of disruptions until the time t. Without any loss of generality, we assume that the disruption rate is a constant value in the period of evaluation, i.e. $\lambda(t) = \lambda_0$. It is commonly assumed that the disruption of vehicles follows the exponential distribution with the parameter λ_0. Once disruptions happen to the vehicles, they will not be available until they get repaired. Mean time to repair (MTTR) is a parameter that shows the average of unavailable time. We use the MTTR in the MC to estimate the duration for which vehicles become unavailable.

The proposed MC evaluates the robustness of schedules as follows. First, for each vehicle the MC estimates the first moment and the duration of disruptions using a random exponential value with the parameter λ (the given disruption rate and a MTTR value, respectively). Let us assume that the first disruption is at time t_1. This means that the vehicle can work from time $t = 0$ until $t = t_1$. Then, the vehicle will not be available for a period equal to $MTTR$ until the time t_2, $t_2 = t_1 + MTTR$. This process is repeated until we reach the makespan - the time by which the last job has finished. Then we repeat this process of simulating disruptions for all the vehicles until they all reach the makespan. In order to prevent any delay in the system, the jobs that are uncovered due to vehicles being unavailable must be assigned to other available vehicles. The MC searches through all the available vehicles in the solution to find suitable substitutions to carry out the uncovered jobs. If such substitute vehicles are found then MC assigns the uncovered jobs to them. Otherwise, new vehicles must be added to the system to carry out the uncovered jobs. We call those vehicles *additional vehicles*. In order to produce a robust number of vehicles, the additional number of vehicles must be estimated and added to the fleet size.

In order to have an accurate estimation of the additional vehicles, the algorithm applies the above MC approach to the schedule of each individual for m_1 times and stores the fleet size of all these m_1 replications. At the end of the m_1 replications, the MC produces an average number of additional vehicles. This average number is considered to be a measure by which to evaluate the robustness of schedules. This MC approach is combined with FSEA to determine the robust number of vehicles. This combined algorithm is MC1-FSEA. In MC1-FSEA, the fitness of an individual is calculated as the number of vehicles decoded from the individual's chromosome plus the additional vehicles needed to deal with the uncertainty, as estimated by the MC. The objective function for MC1-FSEA is defined as: $MC1 = f + \frac{\sum_{i=1}^{m_1} av_i}{m_1}$, where f is calculated using Equation (1); av_i is the number of required additional vehicles to cover the uncovered jobs during disruptions in the ith MC simulation; and m_1 is the number of replications. The pseudo-code of MC1-FSEA is exactly the same as FSEA (Alg. 5) except that the objective function in step 7 (Evaluation) is replaced by $MC1 = f + \frac{\sum_{i=1}^{m_1} av_i}{m_1}$.

C. *Uncertainty in machines' processing time*

The processing time of machines can change due to many reasons, but it usually follows a probabilistic distribution [50]. The distribution is assumed to have already captured various uncertainty factors, such as mechanical faults, operator's skills, position of containers, etc. To capture this uncertainty, we develop another variance of FSEA called MC2-FSEA. MC2-FSEA uses m_2 different sets of generated processing time of machines that are estimated by the above distribution. It then applies FSEA to each set of processing times separately to determine the optimal number of vehicles. Results of runs over the m_2 sets show the impacts of the uncertainty in processing time on the optimal fleet size. The average of results of m_2 runs is considered to be the robust fleet size in regard to this type of uncertainty. The robust fleet size for MC2-FSEA is defined as: $MC2 = \frac{\sum_{i=1}^{m_2} FS_i}{m_2}$, where FS_i is the static optimal fleet size identified by FSEA for run i and m_2 is the number of runs. Note that, in this equation, if instead of FSEA we use MC1-FSEA we can produce the robust number of vehicles under both types of uncertainty. This version is called MC12-FSEA. For MC12-FSEA, the robust fleet size is defined as: $MC12 = \frac{\sum_{i=1}^{m_2} MC1_i}{m_2}$, where $MC1_i$ is the robust fleet size against the uncertainty in vehicles' travel time for run i and m_2 is the number of runs.

VI. Case studies

As mentioned earlier, we considered container ports as the case study for this research. The FSP in ports is a new problem. There have been no published test cases for this problem in ports in the literature. As a result, we created all the test cases for this research from scratch using the real-world data from two of our European port partners.

A. *Real-world test cases*

To generate the test cases, we use some real-world data such as the number of QCs, the number of containers, the size of buffers, the distances between QCs and SCs (Table I), and so on. In the two ports (let us call them port A and port B[3]), the maximum numbers of QCs on one vessel are three and two, respectively. The numbers of containers to be discharged - 100, 200, and 300 - are considered to be in line with the actual transactions in the two ports. Currently, port A has six container

[3]Due to confidentiality agreements, we cannot reveal their actual identities.

stack blocks and port B has two blocks, and we assume containers are distributed evenly between the blocks. Each stack block is facilitated with one SC. Given the actual space under or next to the cranes, the sizes of the buffers vary from 0 to 10.

Table I
DISTANCES (IN METERS) BETWEEN QCS AND SCS.

PORT A																		PORT B			
QC1						QC2						QC3						QC1		QC2	
SC1	SC2	SC3	SC4	SC5	SC6	SC1	SC2	SC3	SC4	SC5	SC6	SC1	SC2	SC3	SC4	SC5	SC6	SC1	SC2	SC1	SC2
220	257	298	467	441	428	250	287	328	497	471	458	340	377	418	587	561	648	784	795	682	693

The only assumptions that we have made (in agreement with the port partners) in generating the test cases are as follows: 1) the speeds of vehicles are constant (Subsec. VI-C); 2) the processing time of quay cranes follows the given distributions (Subsec. VI-B) and 3) no waiting time of vehicles for stack cranes has been considered, similar to [3].

B. Time windows of jobs

As mentioned earlier in Subsec. II-A1, a container has to be picked up within a certain time window, which is the duration between the release time and due time of this container. The release time of a container depends on crane's cycle time - the time it takes for a crane to complete moving a container from the ship to the quay side. To make the test cases as realistic as possible, we used the crane's cycle distribution time-table (Table II) from a real-world scenario [50] to generate container release time. For example, there is a 5% chance that the QC would take 30-40 seconds to process a container. Given the release time of a specific container, we can then calculate its due time following the calculation in Subsec. II-A1.

Table II
DISTRIBUTION OF QCS' CYCLE TIME [50].

PERCENTAGE	5%	15%	20%	19%	19%	10%	8%	3%	1%
CYCLE TIME (SEC)	30-40	40-50	50-60	60-70	70-80	70-90	90-120	120-150	150-180

C. Graph model of the FSP

We follow the approach in Subsec. II-A to create a graph model for the FSP. To do so, first we calculate all the possible pickup times for each container, which are the nodes in the graph, by dividing the time window by a discretisation unit δ. To create the test cases we consider δ to be 60 seconds. Then, among all the nodes, we identify those that are compatible with each other. To do so, we calculate the travel time of vehicles between PDPs (QCs and SCs) based on the distances between PDPs and the speeds of vehicles, which are set to be 4m/s and 2m/s for the empty and loaded vehicles respectively, based on common industrial specifications. Given the pickup and travel times, now we can calculate all the compatible nodes of a given node using the procedure in II-A2. By connecting all the compatible nodes a graph model of the FSP can be created.

VII. EXPERIMENTAL RESULTS IN STATIC ENVIRONMENTS

A. IBM ILOG CPLEX optimiser as a benchmark

To the best of our knowledge, there has been no existing EA research for the FSP in ports. As a result, we considered the integer programming (IP) model in [3], which was solved using the CPLEX solver from IBM, to be the benchmark. CPLEX is the commercial, state-of-the-art solver and mathematically it is proved that the IP model in CPLEX can reach the global optima given unlimited time and resources. We developed the source code for the IP model in [3] and then solved it by CPLEX to find global optima. We then applied FSEA to all the test cases to find its optimal solutions and compared these to CPLEX.

B. Parameter settings of FSEA and CPLEX

We conducted a series of experimental studies to determine the best values for the parameters of FSEA, which are 15, 20, and 5 for the $popSize$, γ, and β respectively, where $popSize$ is the size of the population, and γ and β are the parameters of the adaptive learning method (see Subsec. IV-E). FSEA uses the rank selection. The local search and mutation operators are applied to all individuals in each generation. FSEA stops when one of the following criteria is met first: FSEA reaches the global optima found by CPLEX (if available) or FSEA reaches its 2000th generation. In the latter case, the first time that FSEA finds the best solution will be its processing time. For CPLEX, the best estimate search for node selection and the strong branching for variable selection were considered. The relative gap tolerance was set to 0.01% of the optimal value. If CPLEX is out-of-memory due to gap tolerance, the gap tolerance will be increased to 0.1%.

C. Performance measures

We ran FSEA for 30 times and compared the results to that of CPLEX. There are four possible comparison outcomes. First, FSEA can find the same optimal fleet size as CPLEX in all runs. Second, FSEA can only find the same optimal fleet size as CPLEX in fewer than 30 runs. Third, CPLEX runs out of memory but it provides an integer lower bound with no proof of optimality. Fourth, CPLEX runs out of memory and cannot find any lower bounds. In the first case, the time to reach the global optimum is used to compare the two algorithms, using the Mann-Whitney statistical test with a significance level of 95%. In the second case, we consider CPLEX outperforms FSEA. In the third case, the algorithm with the lower objective value is considered to be better. In the case of equal objective values, the Mann-Whitney statistical test identifies the superior algorithm based on the processing time. In the fourth case, since CPLEX cannot solve the problem, obviously FSEA outperforms CPLEX.

D. Experimental results

We coded FSEA in C++. All the experiments were conducted on a Core 2 Duo CPU 2.98 GHz with 3 GB RAM. We created 165 test cases using the settings given in Sec. VI[4].

Experimental results are summarised into Tables III and IV. Table III shows that FSEA significantly outperforms CPLEX in 128 out of 165 cases. CPLEX outperforms FSEA in 31 out of 165 cases. In 6 out of 165 cases, no algorithm outperforms the other algorithm. CPLEX can only solve 56 cases (the smaller-scale ones) and it cannot solve 109 larger cases due to out-of-memory issues, whereas FSEA is able to solve all cases. This table also shows the effectiveness of the FSEA in terms of identifying the global optima. In 50 out of 56 cases, FSEA finds the global optima as found by CPLEX.

[4]Detailed experimental results are shown in [http://www.staff.ljmu.ac.uk/enrtngu1/Papers/detailed_results.png].

Table III
SUMMARY OF THE COMPARISON RESULTS BETWEEN FSEA AND CPLEX.

PORT	NO. OF QCS	NO. OF TEST CASES	BETTER ALGORITHM			SOLVABLE CASES		FSEA FOUND CPLEX'S OPTIMA
			FSEA	CPLEX	NONE	CPLEX	FSEA	
A	1	33	32	0	1	10	33	10
	2	33	26	4	3	11	33	10
	3	33	23	8	2	12	33	11
B	1	33	26	7	0	11	33	10
	2	33	21	12	0	12	33	9
Total		165	128	31	6	56	165	50

Table IV shows the performance of the two algorithms with respect to the problems' size. Generally, it can be seen that CPLEX cannot be used when the sizes of the problems increase. For example, with 100 containers and buffer sizes ≥ 5, CPLEX runs out of memory. Worse, when the number of containers increases to 200 or 300, CPLEX can only solve the problems if the size of buffer is small, i.e. ≤ 3 or 2, respectively. In contrast, FSEA can solve all larger-scale problems in a reasonable time.

Table IV
THE AVERAGE PROCESSING TIME OF CPLEX AND FSEA (IN SECONDS) TO SOLVE THE TEST CASES. NOTE THAT IN THE CASES WHERE CPLEX RUNS OUT OF MEMORY, CPLEX TIMES ARE SHOWN BY "N/A".

BUFFER SIZE	100 CONTAINERS		200 CONTAINERS		300 CONTAINERS	
	FSEA	CPLEX	FSEA	CPLEX	FSEA	CPLEX
0	14.80	1.82	80.02	14.27	93.27	49.61
1	14.96	1.913	52.83	15.02	96.18	53.46
2	8.97	7.40	144.48	62.62	405.31	N/A
3	107.36	26.19	159.67	N/A	506.91	N/A
4	52.12	60.46	492.65	N/A	914.90	N/A
5	431.67	N/A	132.28	N/A	1674.83	N/A
6	167.33	N/A	450.08	N/A	712.00	N/A
7	201.17	N/A	692.08	N/A	565.86	N/A
8	46.45	N/A	713.69	N/A	1616.15	N/A
9	10.34	N/A	847.65	N/A	1289.40	N/A
10	183.06	N/A	322.95	N/A	769.58	N/A

Based on the results it can be seen that FSEA not only has a reliable performance regarding finding global optima, but it also is able to solve the larger-scale problems where CPLEX fails[5]. FSEA was also tested on large-scale cases where there were 400 - 3000 containers. The results[6] show that FSEA managed to find optimal solutions in a reasonable length of time.

E. Optimal fleet sizes and advantages of using buffers

We also study the optimal fleet size when the buffer size of the ports under investigation changes (Fig. 6). The results show that, when the buffer size increases, the optimal fleet size decreases significantly. For instance, in port A with 100 containers, the optimal fleet size decreases significantly from 20 to 10 when the buffer size increases from 0 to 10. A similar trend is also observed in port B. One reason for this behaviour is that by increasing the size of the buffer more spaces is available in the

[5]Readers are referred to Sec. VII-F for a detailed analysis on the FSEA's operators and the impact of the buffer on the optimal fleet size.
[6]Available in http://www.staff.ljmu.ac.uk/enrtngu1/Papers/CIM_large_scales.pdf

buffer, so it takes more time for it to become full. This means that a vehicle can take more jobs before the buffer is full, and hence a smaller fleet size is needed.

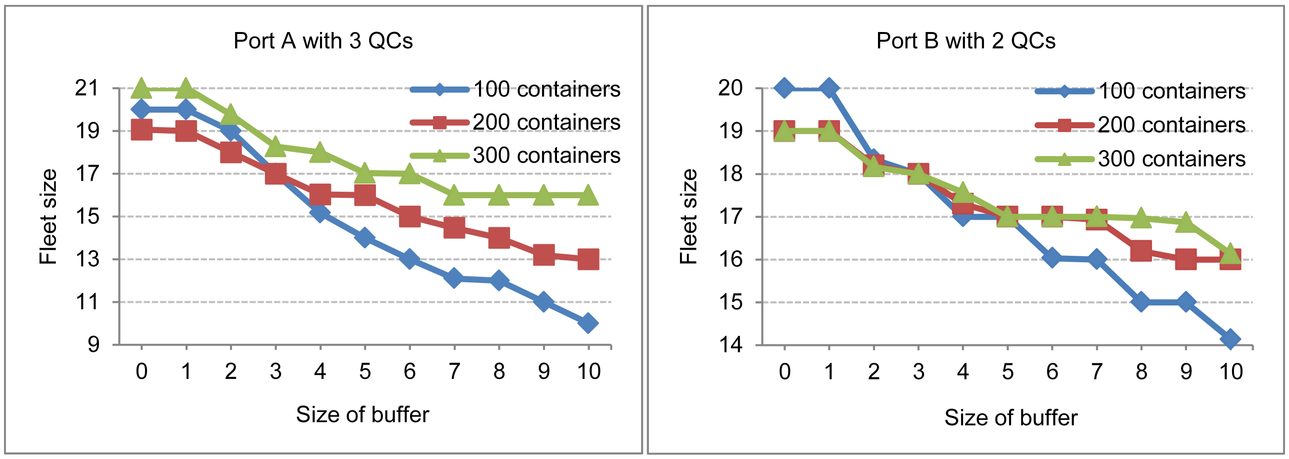


Fig. 6. The impact of using the buffer on the optimal fleet size.

F. *Why FSEA works? Analysing the contribution of different algorithmic components*

1) Analysing the impact of the adaptive local search: We developed two special versions of FSEA where the adaptive local search was replaced by a crossover and a mutation operator each. By comparing the original FSEA with these two versions, we will see how the adaptive local search approach can improve the performance over the crossover and mutation operators.

For each individual, the mutation operator randomly replaces the values of a number of variables with some random uniform values. It then repairs the mutated solution to maintain feasibility. There are some standard crossover operators in the literature, such as the one-point and two-point crossover operators. Such operators, however, do not make sense for the FSP. As a result, we defined a new relevant crossover operator. It randomly selects two individuals, and then from each individual it selects one vehicle randomly and swaps the jobs of the two selected vehicles. By such job swapping, the crossover produces two new individuals. Finally, it repairs the new individuals to maintain feasibility.

2) The impact of the adaptive learning approach: To show how significantly this approach can help FSEA, we compared an FSEA with adaptive learning with another FSEA without adaptive learning (local search + mutation only).

3) Experimental results: Because the behaviour of the various FSEA versions (original, +standard mutation, +crossover, +local search and mutation only) are similar in all test cases, to save space here we will present only two representative cases for the two ports (Fig. 7). Fig. 7 shows that FSEA with the standard mutation or crossover operators cannot find high-quality solutions (the best solution has six or seven more vehicles than that of the original FSEA). This shows the significant improvement brought by the newly proposed local search and mutation operator. Fig. 7 also shows that adaptive learning can improve the fleet size further, by at least one vehicle.

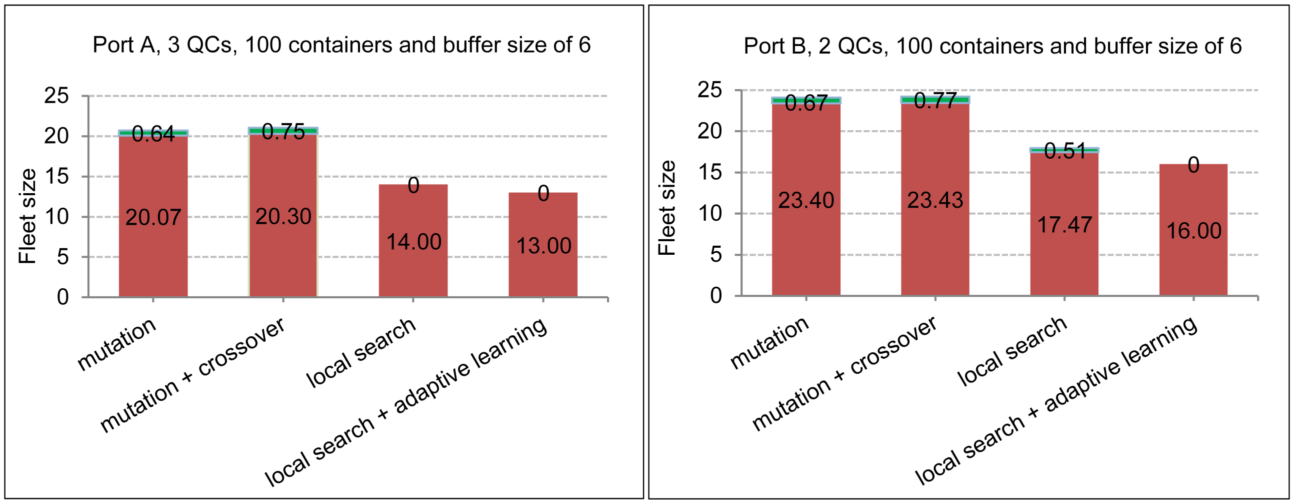


Fig. 7. Analysis on the adaptive local search approach by comparing it with the standard mutation and the proposed crossover operators. Each bar in the figure has two values inside: the lower value is the average and the upper value is the standard deviation of found solutions over 30 runs.

VIII. Experimental results in uncertain environments

A. Case studies

We use the same test cases used in Sec. VII, but with the addition of uncertainty simulated by the MC simulation as described in Sec. V. Due to the lack of space, this section only reports the results of two representative sets of test cases, one from port A and one from port B. Note that, although for the static case, in Sec. VII, we considered an IP model [3] as the benchmark, in the uncertain case it is impossible to use this IP model as the benchmark. The reason is that there has been no existing research on how to extend the IP model for the FSP to deal with uncertainty[7]. Thus, due to the lack of available benchmarks, we compare the results of MC1-FSEA and MC2-FSEA with those of FSEA to show the impact of uncertainty on the fleet size. We then develop high-fidelity simulation models to simulate the real port operations to analyse what would be a robust fleet size in the real ports. We then compare this result from the simulations with the robust fleet size suggested by MC12-FSEA. The purpose is to see if the EA can provide an accurate robust fleet size, in comparison to the high-fidelity simulations.

B. Disruption rates and the MTTRs

As recalled in Subsec. V-B, the robustness of an FSP solution depends on the disruption rates of vehicles and the MTTRs. The disruption rate and the MTTR of automated guided vehicles (AGVs) are chosen from [51, 52]. These two references provide two representative examples of AGVs under low [51] and high [52] disruption rate (called "failure rate" in these studies). In [51] the disruption rate and the MTTR are 5.0×10^{-6} (disruptions/sec) and 1620 (sec), respectively. In [52] the disruption rate and the MTTR are 1.0×10^{-3} (disruptions/sec) and 500 (sec), respectively.

[7]One way to extend the IP [3] for uncertain cases is stochastic programming. However, it has not been done before and is out of the scope of this paper.

C. Calculating the number of samples for MC simulation in MC1-FSEA

It is important to identify the appropriate number of samples (replications) for the MC simulation in MC1-FSEA. Using too many replications will make the algorithm inefficient, while using too few will make the simulation inaccurate. In this paper we use the length of confidence interval [53, Chapter 3] to determine the best number of replications in MC1-FSEA.

We conducted a series of pilot experiments to measure the length of confidence intervals for a different number of replications (Fig. 8 is an example). Fig. 8 shows that, at 100 replications, MC1-FSEA achieves a length of confidence interval of less than 0.3, which is satisfactory. Thus, in this research we considered 100 replications for MC1-FSEA.

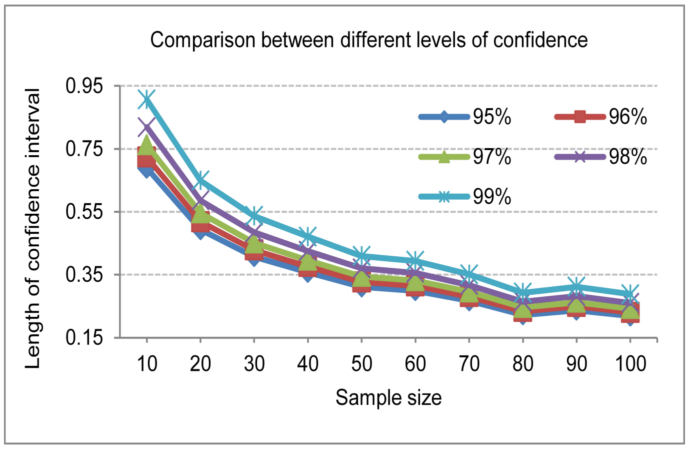


Fig. 8. Length of confidence intervals in different levels of confidence, 95-99% for port A with 100 containers, buffer size of six.

D. Comparison of the results between static and uncertain cases

Results of the proposed algorithms for the static case, the uncertainty in travel time case, the uncertainty in processing time case, and the uncertainty in the travel and processing time case are shown in Table V. All the algorithm settings are the same as in Subsec. VII-B. The stopping criterion for the algorithms in the experiments is considered to be 100 generations. Regarding the case of the uncertainty in processing time, the sample size for MC simulation is set to be 30, i.e. algorithms like MC2-FSEA and MC12-FSEA will run 30 times with 30 different processing times. The fleet size achieved after these 30 runs will then be averaged to calculate the final robust fleet size. The values for the uncertainty in machine processing time are the average of 30 runs including the standard deviations. The values after the $\pm$ sign are the standard deviation.

We can see in Table V that, if the disruption rate is low, uncertainty in travel time will not have a significant impact on the optimal number of vehicles. One reason for such a behaviour is that, since the disruption rate is low, at any time there is likely to be no disruption, so there is no need for additional vehicles. Even if there is a rare disruption, it is likely to be isolated, so there might be a chance that the current fleet will be able to deal with it without the need for additional vehicles. On the contrary, when the disruption rate is high, disruptions are likely to occur more often and on a more global scale. In such a case, the current fleet might not have enough available vehicles to cover all the possible failed vehicles and hence additional vehicles are likely to be required. Table V also shows that the optimal fleet size under processing time uncertainty is very

Table V
A COMPARISON OF THE OPTIMAL FLEET SIZE IN DIFFERENT STATIC AND UNCERTAIN SCENARIOS. THE VALUES FOR MC2-FSEA AND MC12-FSEA ARE AVERAGED FLEET SIZES ± STANDARD DEVIATIONS.

PORT	NUMBER OF CONTAINERS	STATIC (FSEA)	UNCERTAINTY IN PROCESSING TIME (MC2-FSEA)	UNCERTAINTY IN TRAVEL TIME (MC1-FSEA)		UNCERTAINTY IN PROCESSING AND TRAVEL TIME (MC12-FSEA)	
				LOW	HIGH	LOW	HIGH
A	100	10	10.53 ± 0.49	10.20	12.90	10.46 ± 0.49	12.86 ± 0.53
	200	14	15.10 ± 0.53	14.30	18.77	15.00 ± 0.67	18.61 ± 0.58
	300	16	16.30 ± 0.64	16.20	20.05	16.20 ± 0.47	20.10 ± 0.59
B	100	15	15.83 ± 0.45	15.07	20.22	15.59 ± 0.60	20.87 ± 0.75
	200	17	17.13 ± 0.42	17.27	23.57	17.27 ± 0.65	23.46 ± 0.64
	300	17	17.70 ± 0.52	17.24	24.84	17.88 ± 0.55	24.44 ± 0.47

similar to the optimal fleet size in the static case. This indicates that uncertainty in machines' processing time seems not to have a significant impact on the fleet size. This result suggests that, at least for the two studied ports, perhaps we can ignore the uncertainty of processing time to be able to provide an optimal fleet size solution faster.

E. Simulation with high fidelity to validate the robust fleet sizes

In this subsection, we evaluate the effectiveness of MC12-FSEA using a simulation approach. The reason for considering only MC12-FSEA is that this algorithm considers both types of uncertainty, and hence is the most general form of FSEAs. To evaluate the effectiveness of MC12-FSEA, we simulated ports A and B with high fidelity to simulate exactly their real operations under uncertainty when using different fleet sizes. To do so, we used a simulation framework in [54] to develop simulations. In the developed simulation models, we attempted to reproduce the real-world operations, which are the same as the 12 test cases in Table V with the same properties (e.g. distances, speeds of vehicles, number of containers, etc). In addition, the simulations incorporate the same uncertainty (travel time and machine time) as considered by MC12-FSEA.

The simulation models were run under different fleet sizes (from 8 - 30) to identify the total discharging time of vessels for each fleet size. These simulation results helped to show the impact of changing the fleet size on the port performance and, more importantly, to identify an optimal fleet size which is robust against changes. Because the simulation models have been validated by the data from the port partners, the robust fleet size observed by the simulation is the most accurate we can get. Hence, comparing the robust fleet sizes found by simulation with the robust fleet sizes by MC12-FSEA can validate the effectiveness of MC12-FSEA. Each simulation was replicated 10 times and the average results are reported.

Note that, although a high-fidelity simulation can provide the most accurate robust fleet size, it is port-specific. This means that for each different port a new simulation needs to be developed. This is a very time-consuming process. In addition, a simulation model generally takes significantly more time to run than an optimisation algorithm like MC12-FSEA. These two disadvantages makes an EA like MC12-FSEA (if it can be proved to be effective) a much better alternative than a simulation model, because we only need to develop the EA once to apply it to all different ports.

Fig. 9 shows the results of the simulation experiments. As can be seen, in all of the cases for a certain range of fleet sizes the discharging time is almost unchanged. The smallest fleet size in this range can be considered to be the optimal robust fleet size, because (1) if we decrease the fleet size further, the discharging time will increase, and (2) if we increase the fleet size, we will not get any further improvement. In Fig. 9, the optimal robust fleet sizes found by simulations are squared in red and the optimal robust solutions found by MC12-FSEA are circled in black. Note that the fleet sizes found by MC12-FSEA are converted to integers by adding up the average fleet sizes and standard deviations, then rounding them to the closest integer.

As can be seen in Fig. 9, in most cases the robust fleet sizes found by MC12-FSEA are very close to the optimal robust fleet sizes found by the simulation. In 8/12 scenarios the robust solutions found by the two approaches are identical. In the other four scenarios (three from port A under high disruptions), EA underestimates the fleet size, but the differences are not significant: between one and four vehicles and the difference in discharging time is less than 20 minutes. The possible reason for the EA to underestimate in the scenarios of port A under high disruptions is that these are the most complex scenarios. There might be some other factors in these scenarios that the EA model has not considered. The results prove that MC12-FSEA is able to accurately estimate the optimal robust fleet size in the majority of cases.

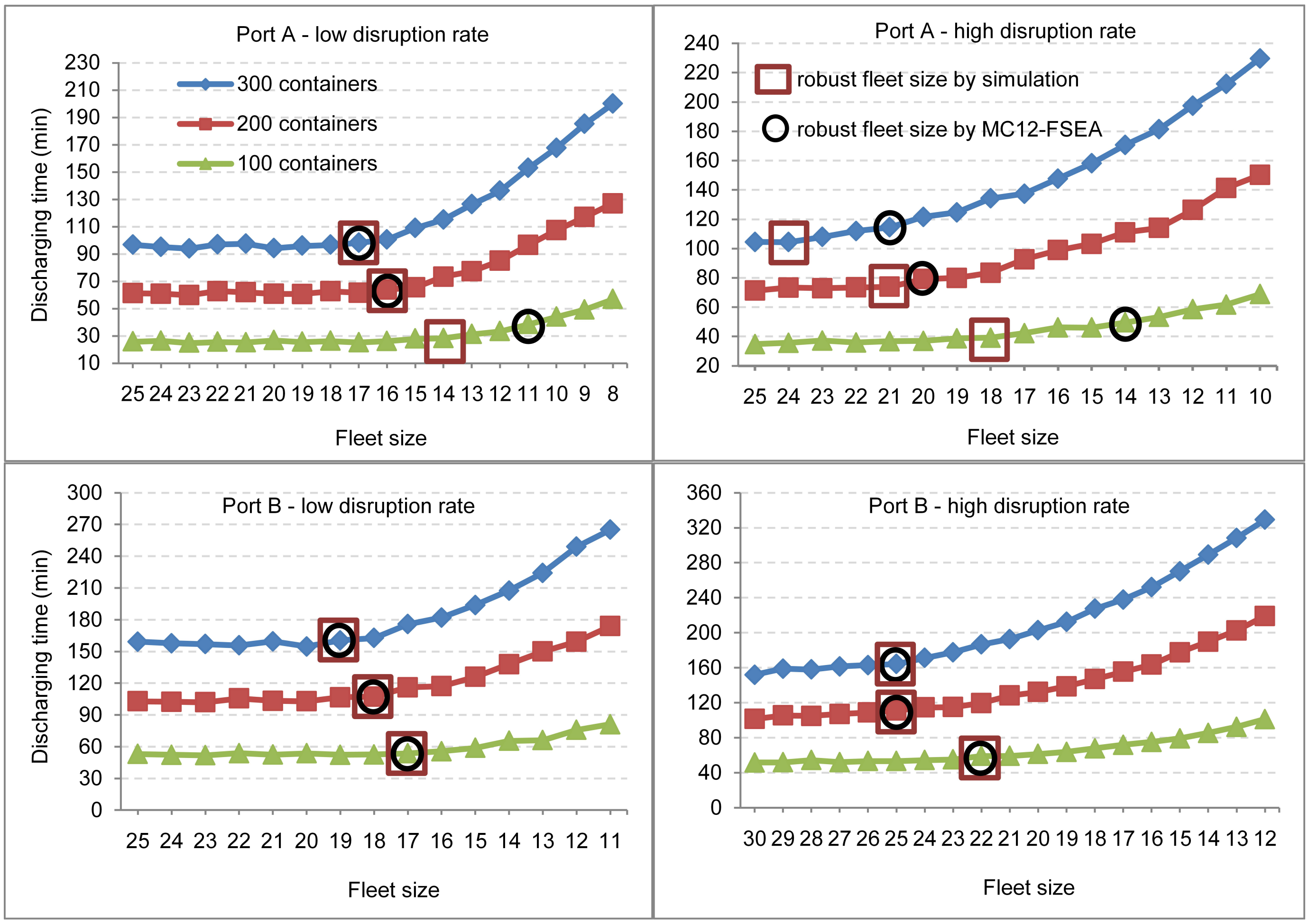


Fig. 9. Average discharging time of vessels in simulation for the instances in Table V. The robust fleet sizes found by MC12-FSEA for each instance are shown as black circles and the optimal fleet sizes found by simulation are shown as red squares.

IX. Conclusion

In this research, we have developed an evolutionary algorithm able to identify the suitable number of vehicles in environments with shuttle transportation tasks (ESTTs), in both static and uncertain situations. We considered the travel time of vehicles and the processing time of machines to be the main sources of uncertainty. In the static case we compared the results of our algorithm, FSEA, with that of the commercial, state-of-the-art solver CPLEX. The results showed that FSEA was significantly better than CPLEX in most cases and especially in all large-scale cases. FSEA also has another advantage over CPLEX: while CPLEX cannot solve the FSP with uncertainty elements, FSEA is able to solve the uncertain problems to find the robust solutions. In the tested uncertain cases, results showed that the effect of uncertainty on the optimal number of vehicles is significant when the disruption rate is high. The results also show that the proposed algorithm is able to provide the optimal robust solution in most of the uncertain cases, as validated by high-fidelity simulation models.

Acknowledgements

This work was supported by a European project, Intelligent Transportation for Dynamic Environment (InTraDE), a seed-corn funding grant by the Chartered Institute of Logistics and Transport, a British Council UK-ASEAN Partnership grant and a British Council Newton Institutional Links grant. The authors are grateful to Kay McGinley, Roisin Murray and Aurélien Bertelle for their helpful information.

References

[1] I. F. A. Vis, "Survey of research in the design and control of automated guided vehicle systems," *Eur. J. Oper. Res.*, vol. 170, no. 3, pp. 677–709, 2006.

[2] C. Ebeling, "Evolution of a box," *Invent. Technol.*, vol. 23, no. 4, pp. 8–9, 2009.

[3] I. F. A. Vis, R. M. B. M. De Koster, and M. W. P. Savelsbergh, "Minimum vehicle fleet size under time-window constraints at a container terminal," *Transp. Sci.*, vol. 39, no. 2, pp. 249–260, 2005.

[4] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "Heuristics for multi-attribute vehicle routing problems: A survey and synthesis," *Eur. J. Oper. Res.*, vol. 231, no. 1, pp. 1–21, 2013.

[5] F. F. Choobineh, A. Asef-Vaziri, and X. Huang, "Fleet sizing of automated guided vehicles: A linear programming approach based on closed queuing networks," *Int. J. Prod. Res.*, vol. 50, no. 12, pp. 3222–3235, 2012.

[6] R. Arifin and P. J. Egbelu, "Determination of vehicle requirements in automated guided vehicle systems: A statistical approach," *Prod. Plan. Control*, vol. 11, no. 3, pp. 258–270, 2000.

[7] P. Egbelu, "The use of non-simulation approaches in estimating vehicle requirements in an automated guided vehicle based transport system," *Mater. Flow*, vol. 4, no. 1, pp. 17–32, 1987.

[8] M. Ji and J. Xia, "Analysis of vehicle requirements in a general automated guided vehicle system based transportation system," *Comput. Ind. Eng.*, vol. 59, no. 4, pp. 544–551, 2010.

[9] B. Mahadevan and T. T. Narendran, "Estimation of number of AGVs for an FMS: An analytical model," *Int. J. Prod. Res.*, vol. 31, no. 7, pp. 1655–1670, 1993.

[10] K. R. Fitzgerald, "How to estimate the number of AGVs you need," *Mod. Mater. Handl.*, vol. 10, p. 79, 1985.

[11] M. E. Johnson, "Modelling empty vehicle traffic in AGVS design," *Int. J. Prod. Res.*, vol. 39, no. 12, pp. 2615–2633, 2001.

[12] C. J. Malmborg, "A model for the design of zone control automated guided vehicle systems," *Int. J. Prod. Res.*, vol. 28, no. 10, pp. 1741–1758, 1990.

[13] J. Tanchoco, P. Egbelu, and F. Taghaboni, "Determination of the total number of vehicles in an AGV-based material transport system," *Mater. Flow*, vol. 4, no. 1-2, pp. 33–51, 1987.

[14] R. Kasilingam, "Mathematical modeling of the AGVS capacity requirements planning problem," *Eng. Cost. Prod. Econ.*, vol. 21, no. 2, pp. 171 – 175, 1991.

[15] P. H. Koo, W. S. Lee, and D. W. Jang, "Fleet sizing and vehicle routing for container transportation in a static environment," *OR Spectrum*, vol. 26, no. 2, pp. 193–209, 2004.

[16] S. Rajotia, K. Shanker, and J. Batra, "Determination of optimal AGV fleet size for an FMS," *Int. J. Prod. Res.*, vol. 36, no. 5, pp. 1177–1198, 1998.

[17] H. R. Sayarshad, "Using bees algorithm for material handling equipment planning in manufacturing systems," *Int. J. Adv. Manuf. Technol.*, vol. 48, pp. 1009–1018, 2010.

[18] D. Sinriech and J. M. A. Tanchoco, "An economic model for determining AGV fleet size," *Int. J. Prod. Res.*, vol. 30, no. 6, pp. 1255–1268, 1992.

[19] I. F. A. Vis, R. M. B. M. De Koster, K. J. Roodbergen, and L. W. P. Peeters, “Determination of the number of automated guided vehicles required at a semi-automated container terminal,” *J. Oper. Res. Soc.*, vol. 52, no. 4, pp. 409–417, 2001.

[20] V. Gabrel, C. Murat, and A. Thiele, “Recent advances in robust optimization: An overview,” *Eur. J. Oper. Res.*, vol. 235, no. 3, pp. 471–483, 2014.

[21] D. Wojtaszek and S. Wesolkowski, “Military fleet mix computation and analysis,” *IEEE Comput. Intell. Mag.*, vol. 7, no. 3, pp. 53–61, Aug. 2012.

[22] H. A. Abbass, S. Alam, and A. Bender, “Application notes: MEBRA: Multiobjective evolutionary-based risk assessment,” *IEEE Comput. Intell. Mag.*, vol. 4, no. 3, pp. 29–36, Aug. 2009.

[23] H. A. Abbass, A. Bender, S. Gaidow, and P. Whitbread, “Computational red teaming: Past, present and future,” *IEEE Comput. Intell. Mag.*, vol. 6, no. 1, pp. 30–42, Feb. 2011.

[24] H. A. Abbass, A. Bender, H. H. Dam, S. Baker, J. Whitacre, and R. Sarker, “Computational scenario-based capability planning,” in *Proc. 10th Annu. Conf. Genet. Evol. Comput.* ACM, 2008, pp. 1437–1444.

[25] S. Baker, A. Bender, H. Abbass, and R. Sarker, “A scenario-based evolutionary scheduling approach for assessing future supply chain fleet capabilities,” in *Evol. Sched.* Springer, 2007, pp. 485–511.

[26] J. F. Alvarez, P. Tsilingiris, E. S. Engebrethsen, and N. M. Kakalis, “Robust fleet sizing and deployment for industrial and independent bulk ocean shipping companies,” *INFOR Inf. Syst. Oper. Res.*, vol. 49, no. 2, pp. 93–107, 2011.

[27] Q. Meng and T. Wang, “A chance constrained programming model for short-term liner ship fleet planning problems,” *Marit. Policy Manag.*, vol. 37, no. 4, pp. 329–346, 2010.

[28] Q. Meng, T. Wang, and S. Wang, “Short-term liner ship fleet planning with container transshipment and uncertain container shipment demand,” *Eur. J. Oper. Res.*, vol. 223, no. 1, pp. 96–105, 2012.

[29] G. Pantuso, K. Fagerholt, and L. M. Hvattum, “A survey on maritime fleet size and mix problems,” *Eur. J. Oper. Res.*, vol. 235, no. 2, pp. 341–349, 2014.

[30] M. Milenković and N. Bojović, “A fuzzy random model for rail freight car fleet sizing problem,” *Transport. Res. C-Emer.*, vol. 33, pp. 107–133, 2013.

[31] H. R. Sayarshad and R. Tavakkoli-Moghaddam, “Solving a multi periodic stochastic model of the rail–car fleet sizing by two-stage optimization formulation,” *Appl. Math. Model.*, vol. 34, no. 5, pp. 1164–1174, 2010.

[32] S. Shadrokh and F. Kianfar, “A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty,” *Eur. J. Oper. Res.*, vol. 181, no. 1, pp. 86–101, 2007.

[33] M. Kılıç, G. Ulusoy, and F. S. Şerifoğlu, “A bi-objective genetic algorithm approach to risk mitigation in project scheduling,” *Int. J. Prod. Res.*, vol. 112, no. 1, pp. 202–216, 2008.

[34] X. Zuo, H. Mo, and J. Wu, “A robust scheduling method based on a multi-objective immune algorithm,” *Inform. Sciences*, vol. 179, no. 19, pp. 3359–3369, 2009.

[35] M. A. Al-Fawzan and M. Haouari, “A bi-objective model for robust resource-constrained project scheduling,” *Int. J. Prod. Econ.*, vol. 96, no. 2, pp. 175–187, 2005.

[36] P. Kobylański and D. Kuchta, “A note on the paper by MA Al-fawzan and M. Haouari about a bi-objective problem for robust resource-constrained project scheduling,” *Int. J. Prod. Res.*, vol. 107, no. 2, pp. 496–501, 2007.

[37] H. Chtourou and M. Haouari, “A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling,” *Comput. Ind. Eng.*, vol. 55, no. 1, pp. 183–194, 2008.

[38] J. Xiong, L. Xing, and Y. Chen, “Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns,” *Int. J. Prod. Res.*, vol. 141, no. 1, pp. 112–126, 2013.

[39] I. Mahdavi, B. Shirazi, and M. Solimanpur, “Development of a simulation-based decision support system for controlling stochastic flexible job shop manufacturing systems,” *Simul. Model. Pract. Theory*, vol. 18, no. 6, pp. 768–786, 2010.

[40] S. Wang and J. Yu, “An effective heuristic for flexible job-shop scheduling problem with maintenance activities,” *Comput. Ind. Eng.*, vol. 59, no. 3, pp. 436–447, 2010.

[41] S. Nguyen, M. Zhang, M. Johnston, and K. Tan, “Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming,” *IEEE Trans. Evol. Comput.*, vol. 18, pp. 193–208, Feb. 2014.

[42] E. Moradi, S. F. Ghomi, and M. Zandieh, “Bi-objective optimization research on integrated fixed time interval preventive maintenance and production for scheduling flexible job-shop problem,” *Expert Syst. Appl.*, vol. 38, no. 6, pp. 7169–7178, 2011.

[43] N. Al-Hinai and T. El-Mekkawy, “Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm,” *Int. J. Prod. Res.*, vol. 132, no. 2, pp. 279–291, 2011.

[44] J. Gu, M. Gu, C. Cao, and X. Gu, “A novel competitive co-evolutionary quantum genetic algorithm for stochastic job shop scheduling problem,” *Comput. Oper. Res.*, vol. 37, no. 5, pp. 927–937, 2010.

[45] M. Sevaux and K. Sörensen, “A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates,” *J. Belg. Fren. Ital. Oper. Res. Soc.*, vol. 2, no. 2, pp. 129–147, 2004.

[46] J. Xiong, K. Yang, J. Liu, Q. Zhao, and Y. Chen, “A two-stage preference-based evolutionary multi-objective approach for capability

planning problems," *Knowl-Based Syst.*, vol. 31, pp. 128–139, 2012.

[47] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *IEEE Trans. Evol. Comput.*, vol. 9, pp. 303–317, June 2005.

[48] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, 2012.

[49] S. Yang, Y. Jiang, and T. T. Nguyen, "Metaheuristics for dynamic combinatorial optimization problems," *IMA J. Manag. Math.*, vol. 24, no. 4, pp. 451–480, 2013.

[50] H. Celen, R. Slegtenhorst, R. T. van der Ham, A. Nagel, R. de Vos Burchart, J. Berg, J. van den Evers, D. Lindeijer, R. Dekker, and P. Meersmans, "Famas newcon," Tech. U. of Delft Erasmus U. Rotterdam, Tech. Rep., 1997.

[51] S. Hoshino and J. Ota, "Design of an automated transportation system in a seaport container terminal for the reliability of operating robots," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2007, pp. 4259–4264.

[52] B. Farling, C. Mosier, and F. Mahmoodi, "Analysis of automated guided vehicle configurations in flexible manufacturing systems," *Int. J. Prod. Res.*, vol. 39, no. 18, pp. 4239–4260, 2001.

[53] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method*. Wiley, 2011.

[54] S. Kavakeb, T. T. Nguyen, C. Ly, Z. Yang, and I. Jenkinson, "A discrete-event simulation framework for container terminals," *Unpublished*, 2015.