Cooperative Double-Layer Genetic Programming Hyper-Heuristic for Online Container Terminal Truck Dispatching

Xinan Chen, Ruibin Bai*, Rong Qu, and Haibo Dong

***Abstract*—In a marine container terminal, truck dispatching is a crucial problem that impacts on the operation efficiency of the whole port. Traditionally, this problem is formulated as an offline optimisation problem, whose solutions are, however, impractical for most real-world scenarios primarily because of the uncertainties of dynamic events in both yard operations and seaside loading–unloading operations. These solutions are either unattractive or infeasible to execute. Herein, for more intelligent handling of these uncertainties and dynamics, a novel cooperative double-layer genetic programming hyper-heuristic (CD-GPHH) is proposed to tackle this challenging online optimisation problem. In this new CD-GPHH, a novel scenario genetic programming (GP) approach is added on top of a traditional GP method that chooses among different GP heuristics for different scenarios to facilitate optimised truck dispatching. In contrast to traditional arithmetic GP (AGP) and GP with logic operators (LGP) which only evolve on one population, our CD-GPHH method separates the scenario and the calculation into two populations, which improved the quality of solutions in multi-scenario problems while reducing the search space. Experimental results show that our CD-GPHH dominates AGP and LGP in solving a multi-scenario function fitting problem as well as a truck dispatching problem in container terminal.**

***Index Terms*—hyper-heuristic, genetic programming, cooperative algorithm, online truck dispatching, container port**

I. Introduction

INTERNATIONAL maritime transportation has seen a significant growth and will continue growing [1]. Such continual growth has stretched (and in some cases, overwhelmed) the capacity of seaport container terminals. However, because of geographical or resource limitations, many terminals cannot quickly expand in size or upgrade their equipment to satisfy the increasing demand. Consequently, container terminals are under pressure to become more efficient, and intelligent algorithms that optimise the use of container terminal resources are promising solutions. In improving the operations efficiency at terminals, many port companies have chosen to start by tackling the crucial truck dispatching problem which connects the seaside operations closely with activities at the yard areas.

Although many studies have successfully developed offline optimisation approaches to solve the truck dispatching problem, these offline optimisation strategies adopt simplified models and are inapplicable to the actual port environment. In particular, most of these models overlooked many uncertain factors that are at play in marine container terminals: crane operators and truck drivers have distinctive operating habits and operating times, ships may not dock on time, and equipment may break down. In our field and simulated tests, although the offline optimisation strategies like integer programming, genetic algorithm and A* search algorithm can achieve good results for the first few tasks, the static model processes subsequent tasks in a fairly arbitrary manner when the operation is disrupted by unknown events halfway through the process. Therefore, considering the practical needs of port companies, this paper proposed a novel evolutionary optimisation method that can fully handle the uncertainties in a real-world setting.

In one of our previous studies [2], experts were first consulted, and their experience was subsequently used to manually crafted heuristics to guide online dispatching, helping the port in our study to significantly reduce the intensive work of truck dispatchers. Despite these desirable results of our manually designed heuristics, considerable expertise is required to construct these heuristics, which only account for some but not all problem scenarios. Every time the container terminal changes its equipment, route scheduling, or task scheduling strategy, these heuristics must be refined by experts, demanding extensive extra time and cost. In addition, due to humans cognitive limitations, experts can only explore heuristics over a very small fraction of the overall search space, which is far from realising the full potential of heuristic algorithms. A data-driven genetic programming (GP)-based heuristic method was investigated to generate heuristics [3] from previous operation data. Despite the superiority of this GP heuristic algorithm compared with manually designed heuristics (see the test result in Section V), we found that arithmetic genetic programming (AGP) without logic operators, such as '$>$', '$<$', and 'IF-ELSE', cannot fully express stochastic multi-scenario problems with a discontinuous solution space. Unfortunately, many problems, such as truck dispatching in terminals, typify these stochastic multi-scenario problems which are featured with many scenarios across different periods of time.

In the truck dispatching problem, various problem parameters/features have different degrees of influence in different scenarios. For example, the average quay crane load time and average quay crane unload time have more influence in scenarios dominated by loading and unloading operations,

This work is supported by the National Natural Science Foundation of China (Grant No.72071116) and Ningbo Municipal Bureau of Science and Technology (Grant No. 2019B10026). (Corresponding author: Ruibin Bai.)

The authors are with the School of Computer Science, University of Nottingham Ningbo China & Nottingham Ningbo China Beacons of Excellence Research and Innovation Institute (email: Xinan.Chen@nottingham.edu.cn; Ruibin.Bai@nottingham.edu.cn; Nick.Dong@nottingham.edu.cn).

Rong Qu is with School of Computer Science, University of Nottingham, UK (email: Rong.Qu@nottingham.ac.uk).

respectively. An AGP method would struggle to deal with such cases. This can be illustrated, as an example, by a piecewise-linear function in (1), where the value of x decides the scenarios and the corresponding results. To handle such problems, researchers usually choose the GP with logic operators (LGP). However, directly adding more operators considerably increases the size of the search space [4], making the search time impractically long, and resulting unsatisfactory solutions due to poor convergence quality. As shown in Fig. 10 and Fig. 11, both AGP and LGP converge to poor local optima instead of the true global optimum after 300 generations. Although LGP performs better than AGP thanks to the logic operators, unfortunately they introduce a significantly bigger search space and poor convergence in some scenarios even with the ability to fully express the function.

$$y = \begin{cases} x & (x < 1) \\ x^2 & (1 \leq x < 3) \\ x^3 & (3 \leq x < 5) \\ x^4 & (5 \leq x < 7) \\ x^5 & (7 \leq x) \end{cases} \quad (1)$$

In this research, we investigate a hierarchical GP encoding structure that can take utmost advantage of logic operators in GP, while at the same time avoid the exponential growth in search space. The proposed algorithm can efficiently handle the complexities and dynamics of real-life seaport terminal truck scheduling. Inspired by research on cooperative coevolution GP [5], [6] and novel GP representations [7], [8], [9], we introduce a cooperative double-layer GP Hyper-heuristic (CD-GPHH) to divide scenario grouping and dispatch ordering into two different subpopulations, in an attempt to improve readability and enhance solution quality for complex dynamic vehicle scheduling problems. In this proposed method, GP individuals are separated into two cooperative layers: a high-scenario layer and a normal-calculation layer, each of which with independent mutation and crossover strategies. Individuals in the high-scenario layer will decide which normal-calculation individual to employ for generating solutions in a specific scenario.

The goal of this study is to develop an effective GP approach to automatically evolve high quality dynamic truck dispatching heuristics to support port companies make decisions in real time (a trained GP expression can generate a dispatch decision almost instantaneously), thus to cope with challenges of multi-scenarios of uncertainties. We tested and compared the AGP, LGP, and our proposed CD-GPHH for both a simple function fitting problem, expressed in (1), as well as a real-world marine container terminal truck dispatching problem. Our work makes two major contributions. First, we demonstrate the effectiveness of using GPHH in solving a real-world online combinatorial optimisation problem faced in many large container ports. Second, we propose a novel bi-level solution framework that can explicitly exploit the structures of scenario switches in many real-world problems and hence improve upon the previously proposed GPHH.

The rest of the paper is organised as follows. Section II provides some background of the research and a review of the relevant literature. Section III details on the model and mathematical formulation of the dynamic truck dispatching problem for marine container terminals. Section IV describes the details of the proposed algorithm. Section V presents the experimental design, results and analysis. Finally, Section VI concludes the paper and suggests future research directions.

II. Background and Literature Review

A. Marine Container Terminal Optimisation

As illustrated in Fig. 1, a marine container terminal can be divided into three major parts: the berth area, yard area, and entry–exit area (the example terminal contains five berths). A fixed number of quay cranes (QCs) deployed along berths for loading (and unloading) containers onto (and from) vessels. A single rail is built along the berth line to enable QCs moving between different berths, but they cannot cross each other. The yard is a temporary container storage area and is often divided into similar-sized yard blocks. Each yard block is identified by a unique ID (e.g. 55, 5H at the center of the yard area) and has 1-2 yard cranes (YCs) or similar type of equipment to complete loading and unloading tasks. Both QCs and YCs can only operate unit sized containers each time hence queues are possible at each operation point. Typically, berths are built in deep sea water areas and connected to yards by a small road network consisting of bridges across the shallow water regions, road segments and intersections. Trucks (or more specifically inner trucks) are used to transport containers between QCs and YCs via this road network and strict traffic regulations are enforced to ensure safety and relieve congestion. The entry–exit area are gates that control the external truck visits.

In most cases, a container terminal primarily aims to improve the efficiency of the sea-side (i.e. QCs) operations so that vessel service times can be minimized and the overall turnover can be maximized. However, because of the interrelations between sea-side and land-side operations, often both QCs, YCs and trucks are considered as schedulable equipment for optimization. The resulting problem becomes extremely challenging due to the following factors: First, the problem is often in large scale caused by the number of containers to be handled as well as the number of QCs, YCs and trucks involved; Second, The problem is highly non-linear because all of the schedulable equipment (QCs, YCs, trucks) can handle unit sized containers only each time and queues and waiting becomes unavoidable; Last but least, uncertainties at different stages of operations (processing times by QCs, YCs, and trucks) are common and predefined plans based on deterministic models would mostly fail.

Considerable recent research has focused on the efficiency of QCs, YCs, and trucks to help port companies cope with soaring demand. Paul et al. [10], Kim and Park [11], Kaveshgar et al. [12], and other researchers [13], [14], [15] have developed different optimisation methods to reduce the makespan of QCs. According to their results, QC makespan can be reduced by adjusting the number, position, or operation sequence of QCs. However, improvements in these studies are extremely dependent on sufficient truck supplies. If truck supply is insufficient for the QCs, the QCs must stop and wait

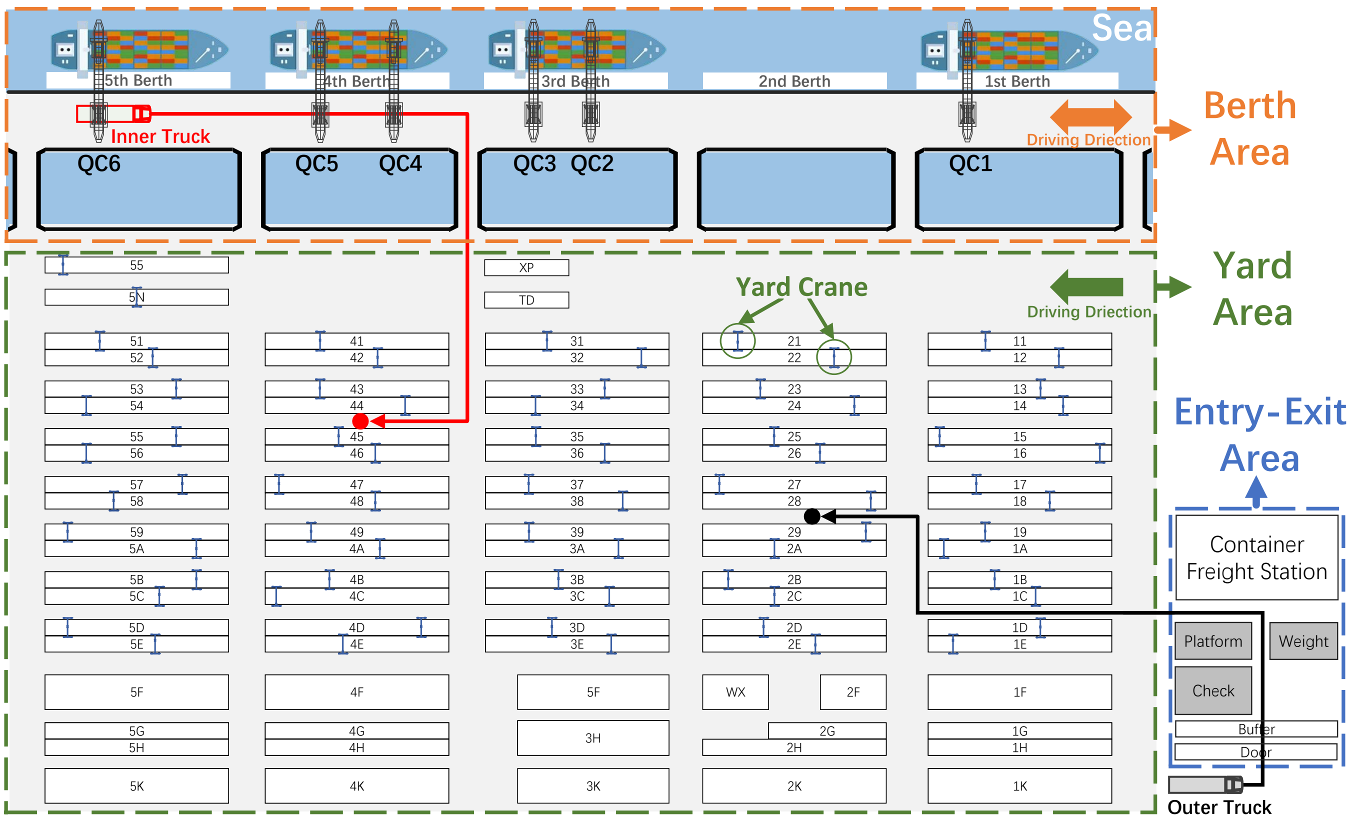


Fig. 1. A Bird-view of a Marine Container Terminal (Ningbo Meishan Port)

for available trucks, which greatly decreases QC efficiency. Furthermore, after Lai et al. [16] provided an overview of several YC schedule modules, Zhang et al. [17], Chen et al. [18], and other researchers [19], [20], [21] discovered that the moving distance and travel time of YCs can be decreased through shuffling the task sequence and changing the container storage location. These studies on QCs and YCs have been useful but limited to the yard or berth area, and most of these studies have assumed an infinite supplies of trucks, which is unrealistic and unhelpful for ports in reducing ship dock time. Subsequently, researchers have found that to further improve the port's turnover efficiency, QCs and YCs must be jointly optimised, with trucks as the links that connect all terminal equipment.

In contrast to the optimisation of QCs and YCs, the optimisation of truck dispatching is required for not only increasing local equipment efficiency in the berth or yard area but also improving the efficiency of the entire port. This is because almost all other equipment in the container terminal must interact with trucks. Many methods have been investigated to tackle the problem, including classical integer programming [22], min–max nonlinear integer programming [23], greedy algorithms [24], novel heuristic algorithms [22] and genetic algorithms [25], [26], [27]. Most studies reported good performance measured by reduction in ship dock time, empty-truck travel distance or the overall truck travel distance.

In real-world marine container terminals, QCs or YCs have different operating times when operating different containers, task sequences may be switched, and trucks do not travel at a constant speed. Because of the influence of such real-world stochasticity, offline methods, such as integer programming, require additional time to recalculate dispatching plans whenever the environment changes; otherwise, the results could become inapplicable. Some of the successful offline solutions such as genetic algorithms [28], metaheuristics [29], simulation-based optimisation [30], can tolerate uncertainty to a limited degree. When the uncertainties become higher, it is very difficult to adapt these algorithms easily [31].

Consequently, a previous study [2] attempted to resolve these issues and proposed an `online` heuristic-based truck dispatching method which uses heuristics to dispatch a task to each idle truck dynamically according to the real-time values of uncertain variables as well as the states of other key features of the port. Like many greedy algorithms, this heuristic method provides relatively good solutions but has no guarantee in terms of optimality [32]. Another drawback of this manual heuristic from domain experts is its lack of adaptivity in multi-scenario problems, leading to considerable research efforts in developing hyper-heuristics that perform well across different scenarios and problem domains.

B. *Hyper-heuristic*

Hyper-heuristic was initially proposed as a general purpose, fast-prototype search methodology. It uses `heuristics to select or generate heuristics` to, in turn, resolve a wide range of problems with acceptable solution quality [33], [34]. Hyper-heuristics differ from metaheuristics in that the

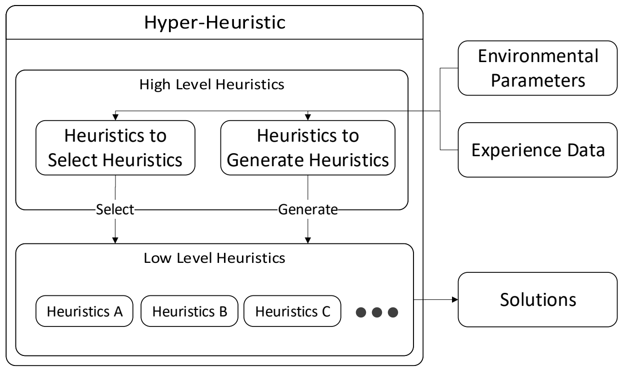


Fig. 2. Hyper-heuristic Framework

search in hyper-heuristics is applied to the space of heuristics, instead of the space of solutions. The underlying assumption of hyper-heuristics is that the heuristic space is less problem dependent than the solution space is. Thus, a more general search method can be developed by searching the solution space indirectly through the heuristic space. Although the No Free Lunch theorem [35] states that it is impossible to develop a truly general-purpose, universal optimisation method for all problems and instances, an adaptive method suitable for problems that share distinct characteristics and structures can be developed. This underpins hyper-heuristics, which use different learning mechanisms (both online and offline) to improve the generality of an algorithm to different problems and scenarios.

As illustrated in Fig. 2, a hyper-heuristic adopts a two-level structure, comprising a high-level heuristic and a set of low-level heuristics that operate on solutions. The high-level heuristic does not interact with problems directly; instead, it either adaptively selects from a set of predefined heuristics to solve the problems at hand or learn to generate new heuristics for the problem. The high-level heuristic typically utilises experience data collected during the problem solving to assist in selecting or generating appropriate low-level heuristics. Studies have reported that hyper-heuristics perform better than their counterparts when solving multiple complex problems, such as educational timetabling [36], [37], two-dimensional strip packing [38], [39], multiobjective release planning [40], [41], vehicle routing [42], [43]. Very few studies have used hyper-heuristics in solving truck dispatching problem in marine container terminals. Furthermore, most of the existing studies have mainly focused on selective hyper-heuristics. Generative hyper-heuristics that support multi-scenario dynamics are less reported in the literature.

C. GP

First proposed by Fogel et al. [44], GP is an evolutionary computation method that evolves a population of programs (often encoded as GP trees) through an evolutionary process (selection, crossover, mutation, and replacement). GP has been used in solving many engineering and optimisation problems. Su et al. [45] indicated that, compared with other methods like decision tree, logistic regression, support vector machine, and artificial neural networks, GP has three major advantages when solving optimisation problems. Firstly, GP has flexible representations; secondly, GP has powerful search mechanisms; lastly, GP generated heuristics are partially interpretable and very efficient in execution which enhances their applicability in practice. Because of these advantages, in this study, GP is considered as a strong candidate solution method for real-world problems.

However, for more complex multi-scenario real-world problems, simple GP generated heuristics cannot meet the requirements. These heuristics only adapt to part of the situations, but achieve unsatisfactory results in general. Therefore, some researches introduced logic operators in GP and proposed to use genetic programming hyper heuristic (GPHH) to select and generate heuristics simultaneously. Like project scheduling [46], [47], combinatorial bi-level optimization [48], [49], and resource allocation [50], [51], GPHH has demonstrated its outstanding ability in tackling intricate multi-scenario real-world problems which encouraged us to apply it in this online container terminal truck dispatching problem.

Traditionally, a GP tree uses a function operator in every tree node and an operand in every internal terminal node. This tree structure makes it fairly easy to encode, evolve, and evaluate mathematical expressions. Meanwhile, GP algorithms with non-tree structures have also been successfully developed, such as linear GP [52], gene expression programming [53], stack-based GP [54]. These aforementioned variety of GP representations are typically efficient in executing genetic operators and have performed well in various problems. However, tree-based GP provides better visualisation, which can enhance comprehensibility because the tree structure can readily represent the logic of decision-making in complex scenarios and can be translated into the form of a logic tree to interact effectively with real-life decision-maker (by, for example, port operators). Our proposed CD-GPHH method adopts a similar tree-based solution structure but, additionally, we explicitly separated decision-making into two levels: those that broadly characterize the scenario patterns and those that provide utility guidance. Section IV details our double-layer GPHH structure.

III. Problem Description and Formulation

Formally, the problem in this paper is defined as follows. An abstract container terminal is represented as a directed graph: $G = (A, C)$, where $C = Q \cup Y$ is the set of operation nodes (or points of work operations for all tasks), Q and Y being the sets of all QCs and YCs, respectively. A is the set of direct driving connections between different nodes. Let d be the truck depot where all trucks depart from at the beginning of the operation and return to once all tasks are completed. The set $V = \{v_1, v_2, v_3, \ldots, v_m\}$ represents the set of m trucks available for assignment. A function $\tau(x, y)$ maps two different operation points $x, y \in C (x \neq y)$ to the travel time from x to y, while respecting the traffic rules of the actual terminal road network.

The following constraints must be satisfied. First, the containers are consolidated into unit-sized `tasks`, each of which can be handled by QC, YCs and trucks in one operation. In practice, a task consists of two small containers (i.e. twenty-foot equivalent unit, TEU), or one large container (forty-foot equivalent unit). Each task has exactly three operations in sequence. For import containers, the three operations are unloading from vessel to a truck by QC, transport from sea-side to yard by the truck and then stacking to the designated yard block by a YC. At any time, an equipment (QC, YC or Truck) can only handle one task. When the required equipment is busy, the operation must wait. For export containers, the three operations include YC loading a container from yard to truck, truck transporting the container from yard to berth and finally QC loading it onto the vessel. Second, each QC will execute one type of operations only (i.e. load or unload but not both) for any given vessel. Third, the tasks are given in the form of a set of work instruction lists $W = \{IL_1, IL_2, IL_3, \dots, IL_{|Q|}\}$, where IL_q is the tasks associated with quay crane $q \in Q$. For loading-only QCs, the completion of the tasks must follow the same order specified in their corresponding instruction lists, while for unloading QCs, the execution order of the tasks can have a maximum sn deviation from their original position in the order. In this paper, we set $sn = 3$ as suggested by our collaborator.

Denote $W = \{w_1, w_2, w_3, \dots, w_n\}$ be the set of all tasks that must be completed, where n is the total number of tasks. The container count of task w_i is denoted as $size_i$ (i.e. for a merged task with two small containers, $size_i = 2$. otherwise $size_i = 1$). The source and destination nodes for each w_i are denoted by a_i and b_i, respectively, and $a_i, b_i \in C$. Denote s_i be the start time of task w_i at its source node and e_i be its completion time at the destination node. Denote sot_i and eot_i be the operation time of w_i at source and destination nodes, respectively, and their sum as ot_i. The operation times at both QCs and YCs are assumed stochastic and are drawn from estimated probability distributions based on the historical data.

To model the problem formally, the assignments of tasks to trucks are defined by the following binary variable in (2):

$$\alpha(v_i, w_j) = \begin{cases} 1 & w_j \text{ is assigned to } v_i \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The following auxiliary variable is defined to indicate whether w_k is serviced immediately after task w_j by truck v_i.

$$\beta(v_i, w_j, w_k) = \begin{cases} 1 & w_k \text{ is visited right after } w_j \text{ by } v_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The order of tasks belonging to a same crane $c_i \in C$ is described by (4).

$$\gamma(c_i, w_j, w_k) = \begin{cases} 1 & w_k \text{ is followed by } w_j \text{ in } c_i \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The main goal in a truck dispatching problem for container terminals is to increase the profit of the port company by improving turnover and reducing the waiting time of ships. A variety of metrics can be used to measure the extent to which this goal is achieved. We adopt the objective to be *units per hour*, which calculates the total number of containers processed per hour by all QCs in the terminal. This is because most port companies use this as the primary indicator to compare their operation efficiency against their competitors. Note that units-per-hour metric is equivalent to the makespan used in many scheduling problems when the set of tasks are fixed. Given these definitions, our truck dispatching problem can be modelled as follows:

$$\max\left(\frac{\sum_{i=1}^{n} size_i}{\max E - \min S}\right) \tag{5}$$

$$\sum_{i=1}^{m} \alpha(v_i, w_j) = 1 \quad \forall w_j \in W \tag{6}$$

$$\sum_{i=1}^{m}\sum_{k=1}^{n} \beta(v_i, w_j, w_k) \le 1 \quad \forall w_j \in W \tag{7}$$

$$\begin{gathered} \sum_{i=\min(x,y)}^{\max(x,y)} \sum_{j=\min(x,y)}^{\max(x,y)} \gamma(a_x, w_i, w_j) \le sn \\ \{x, y | \gamma(a_x, w_x, w_y) = 1, \forall x \ne y, a_x = a_y \in Q\} \end{gathered} \tag{8}$$

$$\begin{gathered} \begin{cases} \sum_{i=1}^{|C|} \sum_{j=\min(x,y)}^{\max(x,y)} \sum_{k=\min(x,y)}^{\max(x,y)} \gamma(c_i, w_j, w_k) = 1 \\ y > x \end{cases} \\ \{x, y | \sum_{i=1}^{|C|} \gamma(c_i, w_x, w_y) = 1, a_x = a_y \notin Q \text{ or } a_x \ne a_y\} \end{gathered} \tag{9}$$

$$s_i = \max \begin{cases} \sum_{j=1}^{n}\sum_{k=1}^{m} \beta(v_k, w_j, w_i) \cdot (t(b_j, a_i) + e_j) \\ t(d, a_i) \cdot (1 - \sum_{j=1}^{n}\sum_{k=1}^{m} \beta(v_k, w_j, w_i)) \end{cases} \tag{10}$$

$$e_i = \max \begin{cases} \max(s_i, \sum_{j=1}^{n} \gamma(a_i, w_j, w_i) \cdot e_j) + \tau(a_i, b_i) + ot_i \\ \sum_{j=1}^{n} \gamma(b_i, w_j, w_i) \cdot e_j + eot_i \end{cases} \tag{11}$$

The objective defined in (5) is the average production speed per unit time (hour), where $\max E$ and $\min S$ are the end time of the last completed task and the start time of the first initialised task, respectively. The constraint expressed in (6) ensures that each task is assigned exactly to one truck, and the constraint expressed in (7) ensures that each task is followed by a maximum one other task or by nothing if it is the last task for the truck. For each crane, due to the rules governing container terminal transportation, the constraints in (8) and (9) ensure that tasks of the same crane cannot start until its preceding task is completed, with the exception of the unload tasks in QCs, for which the operational sequence can be swapped between the $sn = 3$ neighbouring tasks. The constraints in (10) and (11) compute the tasks' start times and end times and ensure that tasks will initialise crane operation after the crane operations of the previous tasks are completed.

Studies have reported that the truck dispatching problem for marine container terminals is NP-hard because it can be reduced to the vehicle routing problem [55]; in other words, the computational time required to search for the optimal solution increases exponentially with the size of the problem. Although previous studies have used various metaheuristics to tackle this problem, these metaheuristics have rested on the assumption of perfectly predictable crane operation times ot_i for all tasks in W prior to the problem solving, which, as discussed earlier, are highly stochastic and impossible to fully predict in advance.

Consequently, we defined this problem as an online optimisation problem and used GP-based generative hyper-heuristics to solve it. We discuss this in the next section.

IV. Methodologies

We open this section by briefly introducing a dynamic truck dispatching system in a real-world marine container terminal, describing how it interacts with optimisation algorithms. Subsequently, we describe four distinct heuristic methods which are designed to solve this problem. Our overall research addresses this challenging problem, and we have implemented manually crafted heuristics and the AGP, which were detailed in our previous study [3], as well as the LGP and CD-GPHH, which is the focus of the present study.

To effectively cope with the dynamically changing business environment and various uncertainties, most existing truck dispatching systems in practice adopt dynamic dispatching methods that must respond requests within a few seconds. Dynamic dispatching contains two essential parts: a dynamic dispatching system and a dispatch algorithm. By communicating with the terminal operating system, the dynamic dispatching system can obtain the real-time status of the port, and, through cooperating with the dispatch algorithm, assigns each truck to the most appropriate task according to the real-time status of the port. The dispatching algorithm plays a crucial role in the operation of the whole system. The choice of the algorithm greatly affects the performance of the dynamic dispatching system. In our previous study, we demonstrated the superiority of the GP algorithm over manual heuristics [3]. However, in practical tests with a port company, we found that the basic GP algorithm fails to meet practical demands, especially when dealing with constant changes of problem scenarios (inbound dominated, outbound dominated, inbound/outbound balanced, etc.). The proposed double-layer encoding scheme within a GP based hyper-heuristic framework explicitly separates the scenario grouping from the truck dispatching to enhance the performance of the data-driven GP with the incorporation of logic operators, meanwhile avoid dramatic increase in the size of the search space. The individuals in the scenario grouping layer and the truck dispatching layer share the same fitness but evolve independently with separate crossover and mutation operators.

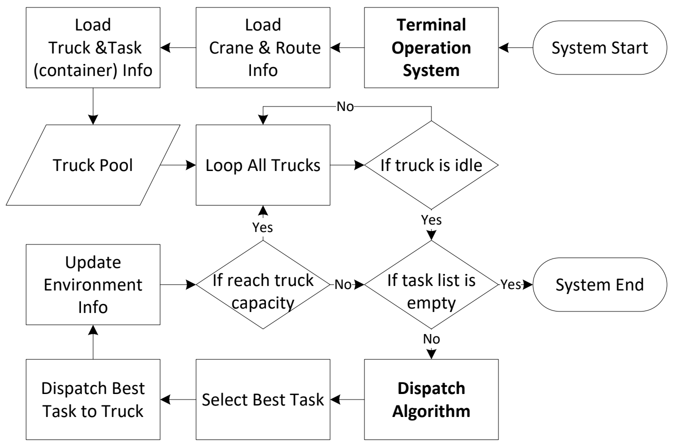


Fig. 3. Dynamic dispatching system flow chart

A. Terminal Dynamic Truck Dispatching System

The terminal dynamic truck dispatching system that interacts with the dynamic dispatch algorithm and the terminal operating system (TOS). Unlike static dispatching, dynamic dispatching does not generate schedules for all tasks in advance. Rather, the system constantly interacts with the port environment by sending out task assignments to idle trucks in real time and conducts real-time monitoring of the changes in key environmental parameters, such as vehicle distribution, crane operation conditions, and vehicle queuing status. Per the dispatching workflow illustrated in Fig. 3, this system has a circular process flow where environmental information is updated, truck are dispatched, and environmental information is updated again. In doing so, the system can feed real-time information to heuristic algorithms (such as traditional heuristic shown in Table V and the proposed GP heuristics) to select proper tasks and support dispatch decision-making. In each loop, this system dispatches the single most appropriate task to an idle truck based on the recommendation of the dispatch algorithm module, if there remains unfinished tasks. If all tasks have been completed, the system sends out no instruction and waits for further tasks.

B. Manually crafted heuristics

At present, many container terminals still recruit coordinators to manually optimise dispatching schemes and adjust the number of trucks assigned to each work queue. Relying on the experience of coordinators, most terminals can still maintain a relatively high operating efficiency to fulfil market demands. This indicates that skilled operators have, over the years, developed operational experience and rules that can help the port achieve effective truck dispatching. Therefore, through communication, surveys, and questionnaires, we summarised the experience of these skilled operators into a manually crafted heuristic and applied it to real-world truck dispatching in marine container terminals, detailed in Algorithm 1 as a baseline for evaluating the performance of our current CD-GPHH.

This manually crafted heuristic algorithm use several user parameters that are set based on coordinators' experience:

Algorithm 1 Manually Crafted Heuristic Algorithm

```
Require: Parameters $parameter$, Travel Time $t$
  function $heuristic(QC, truck)$
    if $crane\_truck\_num < desired\_trucks$ then
      $score \leftarrow travel\_time * (truck\_num - prority)$
    else
      $score \leftarrow travel\_time * desired\_trucks$
    end if
    if $truck\_num \geq truck\_limit$ then
      $score \leftarrow score + 200000$
    end if
    return $score$
  end function
```

$desired\_trucks$ (the most suitable truck number for the QC), $priority$ (the priority of the QC), $truck\_limit$ (the maximum number of trucks for a QC), along with other observed variables in real-time: $truck\_num$ (the number of trucks working for the QC), and $travel\_time$ (the travel time from the current truck to the first (few) task's source node of each QC). These features are then used to calculate a score for each available QC to reflect the preferences of each QC. The algorithm dispatches the idled trucks to the most preferred QCs in a decision tree-like fashion (see Algorithm 1).

C. AGP

Manually crafted heuristics usually focus on the most common scenarios and tend to dispatch trucks evenly among QCs, while neglect other key parameters such as the number of remaining tasks, the QCs' and YCs' operation time, and the queues of trucks. The previous data-driven genetic programming heuristic method in [3] was proposed to address of these issues. The main idea was to use GP to evolve a heuristic that shares a similar structure with the manual heuristic but its parameters are trained automatically based on a large real-world data set.

Algorithm 2 AGP, LGP, and CD-GPHH Evolution Algorithm

```
Require: Initial Parameters $initial$
  $p \leftarrow NewPopulation$
  $p.initial\_individuals(initial.population\_size)$
  $generation \leftarrow 0$
  while $generation < initial.max\_generation$ do
    $p.calculate\_fitness()$
    $p.penalize\_long\_individuals()$
    $next\_generation \leftarrow NewPopulation$
    while $next\_generation.size() < p.size()$ do
      Insert an $individual$ to $next\_generation$ by
      Crossover, Mutation, or Reproduction in $p$
    end while
    $p \leftarrow next\_generation$
    $generation \leftarrow generation + 1$
  end while
```

Algorithm 2 presents the main steps and components for AGP, LGP and CD-GPHH. An initial population is first created

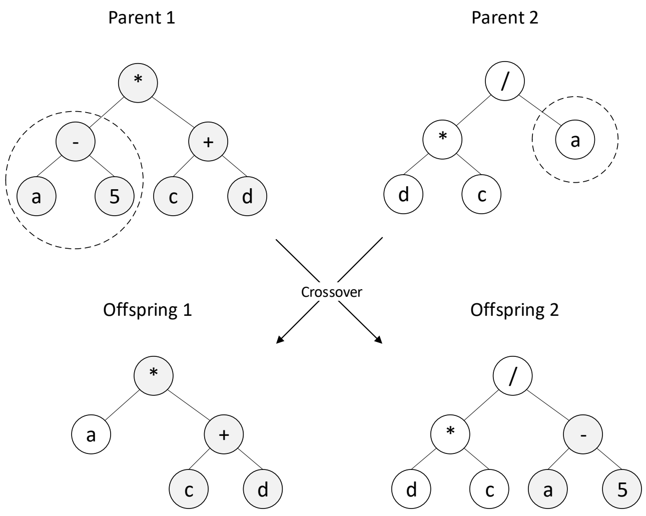


Fig. 4. Crossover operation in AGP & LGP

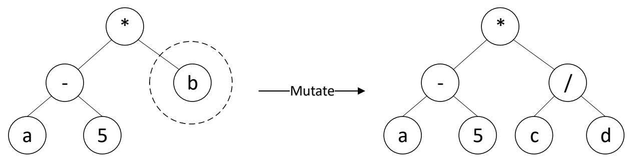


Fig. 5. Mutation operation in AGP & LGP

in according to the preset population size. In the evolution process, the fitness of each individual is calculated according to the objective in (5), plus a penalty term for oversized GP trees. Subsequently, a new generation of population is generated, and a random genetic operator between crossover, mutation, and reproduction will be chosen to generate offsprings of the new population. In this study, we adopted a non-elitist tournament selection method to increase the diversity. The evolutionary process is repeated until the maximum number of generations is reached. The following subsections present the technical details of these components.

1) Crossover: The crossover operation takes two parental individuals selected through `tournament selection` and produces two offsprings using a `single point crossover` operation. An example is illustrated in Fig. 4, where a subtree of parent 2 is combined with parent 1 to generate offspring 1, whereas offspring 2 is resulted from a merge of subtree 1 into parent 2.

2) Mutation: The mutation operation takes one individual as input and generates a new offspring by a slight modification. A mutation point is randomly selected to grow a new randomly generated subtree that keeps the whole tree within the depth limitation, as illustrated in Fig. 5.

3) Depth Restriction: To avoid the bloating problem, researchers usually set a depth limit to the GP trees and discard or prune any result that exceeds the limits. We used two approaches in this study. The first method introduces a penalty term into the fitness function to penalise individuals that are too deep or have too many nodes. The second method sets a maximum depth of subtrees for crossover and mutation operations to generate resulting offsprings within the required

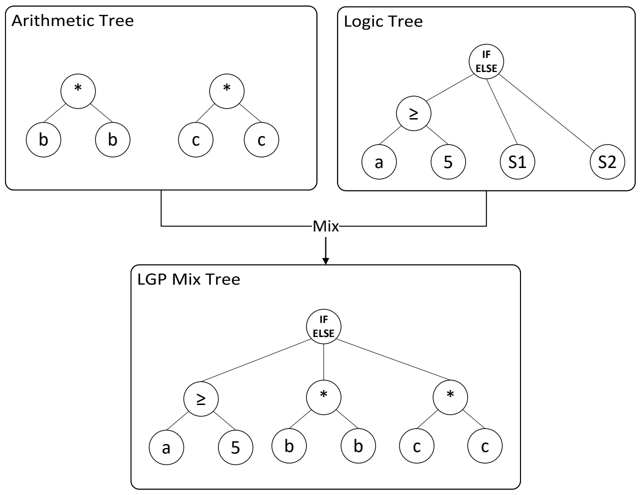


Fig. 6. LGP structure

depth limit.

D. LGP

Although AGP can address the shortcomings of manually crafted heuristics by evolving parameterised heuristics from historical data, the resulting solution can be extremely complex and ineffective when handling problems involving multiple scenarios (i.e. when the distribution of random variables changes over time). For such problems, the use of some discrete utility functions is more elegant and efficient. LGP combining logic expressions with algorithmic trees presented in the previous section, different subtrees can be generated for different scenarios, which improves the performance of the algorithm and produces results that are adaptive to complex multiscenario problems. This ability to adapt to different scenarios using a combination of the logic tree with arithmetic trees follows the general framework underlying hyper-heuristics [56], and the approach is thus also named as a GPHH.

LGP trees with a positive probability of generating several logic trees that sit at the top to select a number of arithmetic trees at the bottom (Fig. 6). LGP shares similar crossover and mutation operations with AGP but has additional operators designed for the logic tree, as shown in Table I. Moreover, a loosely-typed GP is used in LGP, which allows arithmetic and logical operations being combined freely. When performing logical calculations, numbers greater than 0 are treated as logically true, and vice versa as logically false. In this way, after the introduction of the "IF-ELSE" operator, different subtrees can be selected for calculation through the logical values of the previous decision tree. Therefore, some multiscenario problems difficult to be encoded by a single depth-constrained arithmetic tree can now be more effectively addressed in LGP. For example, with the assist of "IF-ELSE" operator and comparison operators, we can evolve a simple LGP tree to represent the discontinuous function in (1) fairly easily.

TABLE I
GP OPERATORS

Name	Label	Description	Algorithm
add	+	Add operation	AGP, LGP, CD-GPHH
sub	-	Minus operation	AGP, LGP, CD-GPHH
multiply	*	Multiplication operation	AGP, LGP, CD-GPHH
divide	/	Division operation (protected)	AGP, LGP, CD-GPHH
greater or equal	>=	Greater than or equal to	LGP, CD-GPHH
less or equal	<=	Less than or equal to	LGP, CD-GPHH
IF ELSE	if_else	Conditional operator (if-else)	LGP
and	&	Logic AND	LGP, CD-GPHH (only in real-world problem)
or	|	Logic OR	LGP, CD-GPHH (only in real-world problem)
max	max	Return the maximum value	LGP, CD-GPHH (only in real-world problem)
min	min	Return the minimum value	LGP, CD-GPHH (only in real-world problem)

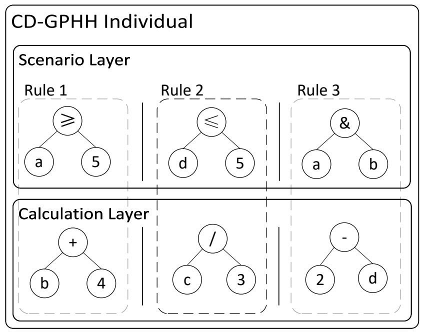


Fig. 7. CD-GPHH individual structure

E. CD-GPHH

Although LGP has shown promising performance in handling complex discontinuous functions and adapting to multiscenario problems, these abilities tend to be unreliable and only a few individuals in LGP population possess correct structure with multiscenario abilities. This is because that introducing logic operators greatly increases the search space where it is extremely difficult for LGP to converge to a good solution within a reasonable computational time. In our previous experiments, LGP can produce some effective results most of time, but lacks required reliability and consistency demanded in industrial operations. Furthermore, even the depth of the GP tree is limited, the interpretability of the resulting solutions is unsatisfactory because the logic expressions are mixed with the arithmetic terms in a LGP tree without a clear structure. When we tested the LGP solutions in a real-world container terminal in the Ningbo Port, they were not generally accepted by port operators, who believed the results too difficult to understand and thus unsuitable for practical use.

In practical problems, performance metrics are typically separable. Take the truck dispatching problem for ports as an example, where operators usually consider different scheduling strategies based on, for example, task categories, yard conditions, and number of trucks available for dispatch, etc. In fact, these factors are high-level scenarios that are generally not directly involved in the formulation of task ranking rules; they instead play a role in the selection of different policies. Some researchers exploit these decision variables by grammar-based LGP to generate individuals with scenario distinguish ability [7], [57], [58], [59]. However, because the grammar-based GP only adds grammar filtering to the normal LGP, and not explicitly separate the scenario selection from the calculation layer. As a result, these factors do not always appear in scenario layer to play its decisive role in scenario selection without presetting. Instead, they are often embedded in the calculation layer which can cause unreliable performance.

Consequently, to reduce the size of the search space and to improve performance, it is necessary to separate scenario information from dispatch rules, leading to our proposed CD-GPHH method which reduces the search space size by separating arithmetic trees and scenario trees in two different layers. The concept underlying CD-GPHH is to evolve the scenario grouping trees and truck dispatch trees concurrently but at two different layers. More specifically, each individual in CD-GPHH has a scenario layer and a calculation layer (Fig. 7). The scenario layer contains logic trees for scenario clustering purposes, and the calculation layer includes arithmetic trees that share similar structures as those in the AGP method. Each logic tree in the scenario layer is bound to an arithmetic tree from the calculation layer and grouped as a rule. In the truck dispatching problem for ports, when a tree in the scenario layer evaluates to `true` (greater than zero), then the corresponding tree in the calculation layer is invoked to compute utility scores for different truck–task assignments. Notably, thanks to this layered structure, our CD-GPHH is also more comprehensible, while enhances the quality of the resulting solutions.

Note that, in CD-GPHH, trees in both the scenario layer and calculation layer are bound into `rules` for easier computation during implementation. The algorithm first operates on a specific $rule$ before processing the two trees inside each rule. Moreover, since the number of scenarios (number of rules) was introduced as a hyper-parameter in CD-GPHH, we extend the traditional mutation operation in GP to enable it to learn and adjust automatically. These three operations are detailed as follows:

1) Crossover: The crossover operation takes two individuals (parents) as inputs. As illustrated in Fig. 8, one random rule of each parent are then selected and single point crossover operations are applied to both layers independently, resulting in two logic trees and two arithmetic trees that can form four different rules. When the four rules are inserted back to the two parents to replace the previously selected rules, eight new offspring are created. However, to maintain the diversity of the population and prevent the same pair of parents from producing too many offspring, two randomly selected offspring are retained in the next generation.

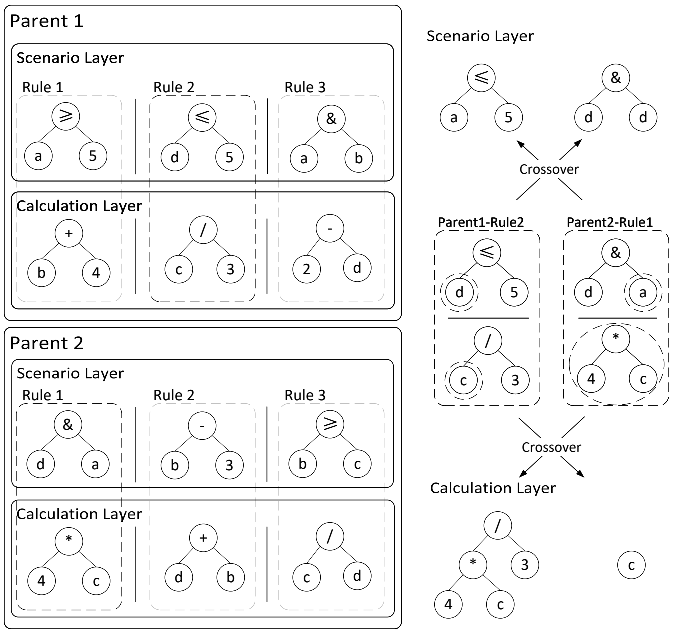


Fig. 8. CD-GPHH crossover operation

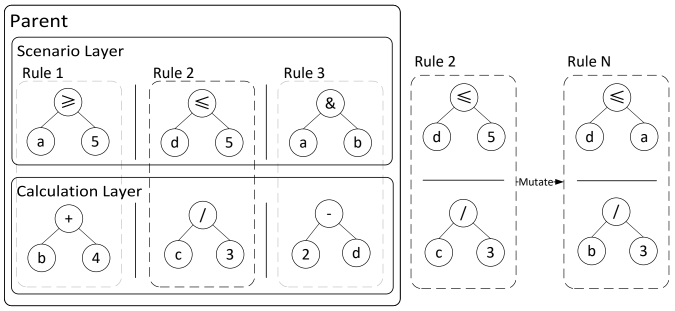


Fig. 9. CD-GPHH mutate operation

2) Mutation: The standard mutation in CD-GPHH is applied to one parent selected by tournament. A rule in the parent is randomly selected for mutation. For example, in Fig. 9 is rule 2 selected for mutation. Either the scenario layer tree or the calculation layer tree or both are modified to form a new rule (same as AGP). This new rule is then inserted into a random location of the parent to generate a new individual (hence this new individual has one more new rule than the parent). Meanwhile, to maintain diversity and dynamically adjust the number of rules, between zero to two randomly chosen rules in this new individual are subject to removal. Finally, if the total number of rules in this new individual exceeds the rule count limit, another randomly selected rule will be removed.

3) Solution Decoding in CD-GPHH: In CD-GPHH, decoding a solution involves testing all logic sub-trees (except the last) in the scenario layer in sequence, until a sub-tree evaluates to `true`. The corresponding calculation tree will be chosen to estimate the performance of different truck–task assignments. If no logic tree evaluates to `true`, without evaluating the last logic tree, the last (default) calculation tree will be used.

Notably, the increase in the search space of AGP was not large. This is because AGP has no logic operators and not many complex combinations of logic and arithmetic operators in AGP trees. CD-GPHH achieved tolerable performance by separating each rule into a scenario layer and calculation layer. This example illustrates the effectiveness of the proposed solution structure in our CD-GPHH algorithm with respect to reducing the search space while retaining the scenario matching capability of LGP.

V. Experiments and Results

We evaluated the performance of our proposed CD-GPHH against a conventional AGP as well as a LGP for solving multiscenario problems. We first compared their performance for the simple multi-scenario function fitting problem described in Section I. Subsequently, we conducted extensive experiments for the real-life truck dispatching problem for marine container terminals. For the truck dispatching problem, we also compared our CD-GPHH against the traditional heuristic method used in real-world and a manually crafted heuristic reported in [2]. Since parameter tuning is not the key focus of this paper, we just ~~follow~~ the common settings~~ and~~ all three GP-based algorithms~~ were run with the same settings~~, as listed in Table II. The number of rules in CD-GPHH was set to 1-10~~ in the experiments~~.

TABLE II
GP Initialisation Parameters

Population Size	1024
Max Generation	300
Crossover Rate	0.6
Mutation Rate	0.3
Reproduction Rate	0.1
Tree Initialization Method	Ramped half-and-half
Selection Method	Tournament Size 7
Tree Depth Restriction	10
Long Individual Penalize Scale	0.0001
Rule Amount Restriction (CD-GPHH only)	1-10

TABLE III
AGP, LGP, and CD-GPHH Test Results on Problem (1)

		AGP	LGP	CD-GPHH
Standard Deviation	**Min**	**288.46**	**4.68**	**2.02×10^{-12}**
	Mean	447.21	248.24	0.042
	Max	680.65	940.08	0.22

A. Simple Multi-scenario Function Fitting Problem

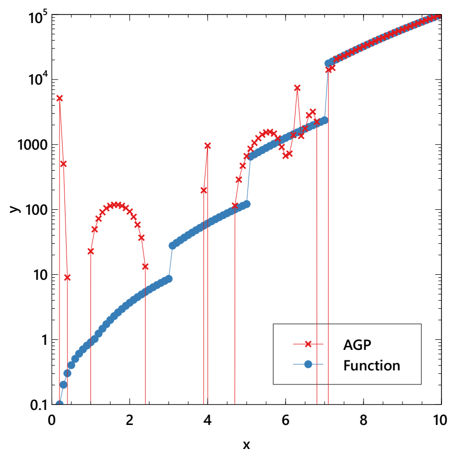
In this experiment, the task is to fit ~~the~~ function ~~in~~ (1). The ~~function has~~ one input variable x~~,~~ and one constant~~,~~ correspond~~ing~~ to two terminals in ~~our GP-based methods~~. The variable terminals were set to an actual value during the calculation process, whereas a constant integer between 0 and 10 was set as the terminal. During each test, 100 randomly pregenerated instances of different values of x were used as the test set. Fitness was defined as the standard variance between the fitted function and the original function. In other words, a fitness closer to zero indicates a good solution.

Table III presents the statistical results from all three GP-based methods in 30 tests. ~~Our~~ CD-GPHH performed significantly better than AGP and LGP~~ on this simple function fitting problem~~. As shown in Fig. 10, ~~when~~ comparing ~~the best results produced by the three GP individuals with~~ the original function, ~~it can be clearly seen that~~ AGP performs the worst, fitting only the case where x is greater than 7, while LGP fits the range from 3 to 10. ~~Since~~ when x is less than 3, the fitting errors lead to relatively small variance in y (equivalent to small variance in individuals' fitnesses)~~,~~ LGP does not fit well in 0 to 3 region~~, while our CD-GPHH does much better.~~ ~~As it can be seen~~ in Fig. 10, CD-GPHH has replicated almost 100% of the original function, indicating the potential benefits of a predefined hierarchical structure for multiscenario problems. The simplified best-performing individuals of each method ~~in the experiments~~ are as follows:

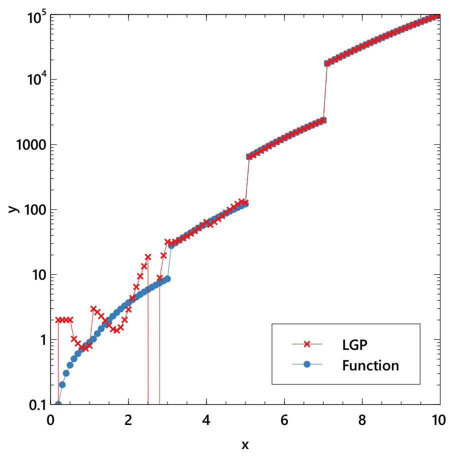
- AGP:
  $(3920805 * x^{10} - 90391140 * x^9 + 613811750 * x^8 + zoo * x^8 + 563298482 * x^7 - 22692826327 * x^6 + 78451667966 * x^5 - 16277641800 * x^4 + -218951495280 * x^3 + 62156445120 * x^2 + 14180443200 * x)/(3991680 * x^5 - 103783680 * x^4) + 1010892960 * x^3 - 4372885440 * x^2 + 7090221600 * x$
- LGP:
  $if\_else(-(5 >= 2*x + (x <= 7))/5 + (7 <= x)/5, x * (x^4 - 1) + x, x * ((x - if\_else(if\_else(if\_else((x - 3)/(2*x), (5 - x <= -(5 >= 2*x + (x <= 7))/5 + (7 >= x)/5), 1 - if\_else((x^2 * (x <= 8) + (x <= 0)), 0, 1)) + 1, if\_else((3 <= 2*x), x - 11/10, 1), 0), 1, 7) + 5) * (40 * x^2 - x^3 + 40 * (9 >= x) - 440) + 320)/40)$
- CD-GPHH:

Rule	Scenario	Calculation
1	$7 <= x$	x^5
2	$-16*x^2*(x-10)^2+51*x-6)/(4*x*(x-10)) >= (x+3/x >= 4)$	x^2
3	$x^2 + x <= 4$	x
4	$x - 10/x <= 3$	x^3
5	$x > 3/4$	x^4

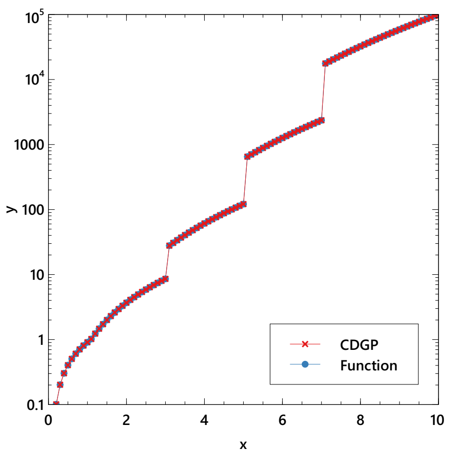
Obviously, the results by AGP and LGP are hard to ~~read~~ and differs greatly from the true function ~~in~~ (1). Although LGP~~'s results contain~~ the "IF-ELSE" ~~operator~~ and relational operator that were in the original expression, the results are ~~still~~ confusing and does not fit the original function well. By contrast, with the help of its bi-level structure, CD-GPHH obtained a concise and easy-to-understand function that is almost identical to the original one. ~~Comparing~~ the best evolution results of the three methods in Fig. 11, ~~it can be found that~~ CD-GPHH has escaped the local optimum trap and converged to the global optimum at the 150th generation, while both AGP and LGP ended up with a poor result. In general, under the evolutionary framework designed in this research, CD-GPHH takes advantage of two cooperative populations and



(a) AGP



(b) LGP



(c) CD-GPHH

Fig. 10. AGP, LGP, and CD-GPHH best result in fitting Eq. (1) compared with the original function.

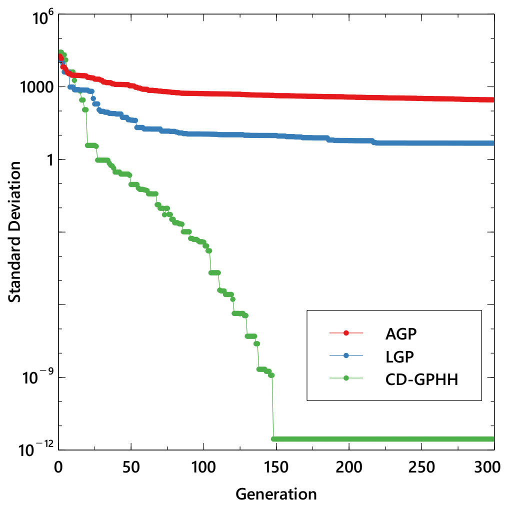


Fig. 11. The best performance of AGP, LGP and CD-GPHH on problem (1).

obtains well-performed, concise and understandable results, confirming ~~the~~ superiority ~~of CD-GPHH~~ on solving simple multi-scenario function fitting problem.

B. Results for Real-life Truck Dispatching

~~We used~~ three different GP methods ~~to address the~~ problem described in Section III. The proposed CD-GPHH ~~has a total of 14 GP features (see~~ Table IV ~~for details)~~. The other settings are given in Table II. To better tackle this complex real-world problem, 4 new operators were added ~~as shown in the Table I~~. The performance of the ~~proposed method is compared with other~~ methods ~~(~~Table V~~) in this subsection~~.

To update the environment state values of these features and evaluate the fitness of each individual in ~~our GP~~, we built an event-based simulator ~~according to~~ the mathematical model described in Section III. The simulator interacts with the dynamic truck dispatching system (Fig. 3) to provide functions for evaluating the fitness of each individual and generating new environment data after each truck dispatch. It can simulate all events occurring in real-world truck dispatching in marine container terminals, such as vehicle movement, container loading and unloading, and the real-time data exchange with the dynamic dispatching system.

The data sets used in this experiment were extracted from actual historical operational data ~~from~~ the Ningbo Meishan Port. The data sets simulate a typical situation where one

TABLE IV
FEATURES OF AGP, LGP, AND CD-GPHH IN REAL-WORLD TRUCK DISPATCHING PROBLEM

Name	Acronym	Description
travel_time	tt	Travel time form truck to task start crane
operate_type	ot	The ship load/unload task type (0 for load and 1 for unload)
dispatch_type	dt	The task dispatch type (0 for normal tasks and 1 for tasks need to be merged)
yard_crane_type	yct	The yard crane type (0 for normal crane and 1 for remote-controlled crane)
total_truck_num	ttn	Total trucks number can be dispatched
num_to_min_truck	ntmt	The difference between the quay crane trucks number and the minimum trucks number
start_node_truck_num	sntn	Total trucks number of the task start crane
end_node_truck_num	entn	Total trucks number of the task end crane
start_node_wait_truck_num	snwtn	Waiting trucks number of the task start crane
end_node_wait_truck_num	enwtn	Waiting trucks number of the task end crane
remain_task_num	rtn	Remaining tasks number of the task related quay crane
average_load_time	alt	Average load time of the crane
average_unload_time	aut	Average unload time of the crane
Constant Number	-	Random constant number

TABLE V
TRADITIONAL HEURISTIC METHODS FOR TRUCK DISPATCHING PROBLEMS

Name	Acronym	Description
fixed	-	Dispatchings are according to the binding QCs
manual	-	Select the task though manual crafted heuristic (Currently used in Meishan Port, Ningbo)
first-in-first-out	FIFO	Dispatchings are sequenced first-in-first-out
shortest traveling time	STT	Select the task with the shortest traveling time
longest traveling time	LTT	Select the task with the longest traveling time
random	-	random dispatch
most task remaining	MTR	Select the task with most task remaining
smallest task remaining	STR	Select the task with smallest task remaining

container ship berths at the port to load and unload containers, and the port needs to complete the work of the ship as soon as possible to ~~let~~ the ship ~~leave~~ the port earlier. Several instances were generated base on this situation, ~~and each instance has 1~~ ship berth with ~~6~~ QCs. The number of trucks is set to be the actual number of truck working during the data extraction time period, between 24 and 48, and the traffic map is shown in Fig. 1.

As aforementioned in Section III, due to strict traffic regulations (mainly single-direction road segments), there are very few route options between QCs and YCs. Therefore, the truck travel time is pre-computed through the shortest path algorithm on the port road network assuming an average truck speed of 8km/h. Meanwhile, the operating (load/unload) time of the crane for the container on the truck is uncertain.

We extracted 10 sets of historical task data of different time periods (ports have different operating scenarios at different times) with ~~5~~ sets for training (sets 1-5) and ~~5~~ sets for testing (sets 5-10). Each set (both training and testing) contains 10 instances in same time period with 200 tasks ~~with~~ a mixture of both loading and unloading operations. For each training instance, 30 independent runs with different random seeds were conducted, leading to a total of 10 * 30 = 300 runs on each training set, ~~and finally~~ the average results ~~over these 300 runs for each set is given~~ in Table VI. Recall that the objective is to maximise the number of tasks handled per unit time (hour). Therefore, larger values indicate better performance. ~~Then, we chose~~ the best-performing individuals from the three GP algorithms ~~and test them for~~ test sets 6-10. Each instance in the test sets run only once (because the evolved GP trees are deterministic). ~~However, since each set contains 10 instances, there are~~ total ~~of~~ 10*5 test instances ~~which are not seen during GP training. Therefore, they represent~~ significant robustness ~~test~~ for all the methods. Table VII provides the average results of all the methods across 10 test instances in each set as well as the averages across all 5 sets.

~~Below we include~~ the best-performing individuals of AGP, LGP and CD-GPHH. Note that these trees have been simplified for ease ~~of reading~~.

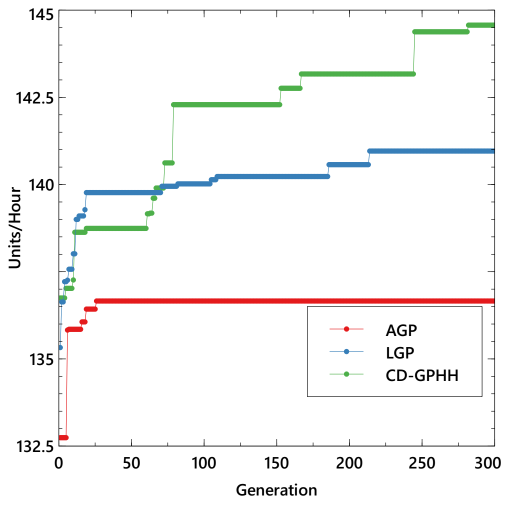


Fig. 12. The performance of AGP, LGP and CD-GPHH in Training Set 4.

- AGP:
  $(((entn + sntn) - rtn) + snwtn) + (ttn/((((((ttn + ot)/(((tt/yct) * rtn))) + ((rtn * alt)/(ntmt + alt))) - (((rtn + ntmt)/(entn + snwtn)) + ((yct + ttn) * (enwtn - sntn))))/((((entn + entn) * (snwtn * 2))/((aut/ttn)/(dt*sntn)))/(((ntmt+dt)-(tt*rtn))+ ((snwtn * aut) - (tt - sntn))))) + rtn))$
- LGP:
  $((((max(((ot|7|((entn/10) + ot))|min(max(2, rtn), -ntmt)), snwtn) <= ot)/10) * (entn * dt)) <= if\_else(rtn, tt, sntn)) * ((ttn/(((1 + if\_else(yct, 6, rtn)) * ((((sntn/max(dt, ntmt)) * (rtn\&snwtn)) /snwtn)\&(sntn * dt))) + (((ttn/rtn)/((entn/if\_else(tt, sntn, enwtn))\&enwtn)) >= ((10 - tt) <= (4 * entn)))))/(enwtn >= ntmt))$
- CD-GPHH:

Rule	Scenario	Calculation
1	$((((ntmt >= sntn)- ntmt + entn)) + (ot +yct + min(ttn, tt)))|((min(tt, enwtn) + max(dt, ot))/tt)$	$(dt >= enwtn) + max(entn, ttn) + rtn$
2	$ttn >= 6 * 5$	$snwtn/3+alt*aut$
3	$(min(tt, yct)/(ot\& snwtn) <= (ot + tt)) /((dt >= entn) <= (ot <= sntn))$	$min(((enwtn <= entn) + yct), ((ttn +entn) * ntmt))$
4	$max((entn >= 5), ntmt)$	$ttn * alt$

~~Among~~ the experiment results ~~proved by t-test~~ ($\alpha = 0.001, p = 0.00$)~~, the~~ performance of all the three GP algorithms ~~is better~~ than the traditional heuristic algorithms, ~~and CD-GPHH achieved the best. Using manual and fixed heuristic algorithms as benchmarks,~~ the improvement percentages (Imp. Pct.) of AGP, LGP and CD-GPHH are about 7%/15%, 11%/19% and 14%/25% , respectively. ~~By further comparing~~

TABLE VI
AGP, LGP, AND CD-GPHH TRAINING RESULTS (UNITS/H)

	Fixed	**Manual**	**Random**	**FIFO**	**STT**	**LTT**	**MTR**	**STR**	**AGP**	**LGP**	**CD-GPHH**
Set1	106.05	106.87	100.01	94.58	71.39	91.88	108.75	54.89	124.03	126.09	132.88
Set2	113.63	118.91	108.66	96.99	69.23	71.35	115.66	58.67	127.22	132.32	138.45
Set3	110.10	114.27	106.39	100.55	75.50	76.86	120.58	56.33	129.38	128.86	138.73
Set4	115.33	121.48	112.75	97.60	69.22	87.17	121.91	59.07	135.53	138.37	143.99
Set5	96.76	116.88	102.95	95.13	66.40	65.76	114.46	55.97	121.34	125.07	132.44
Mean	108.37	115.68	106.15	96.97	70.35	78.60	116.27	56.99	127.50	130.14	137.30
Imp.	0.00%	6.74%	-2.05%	-10.52%	-35.09%	-27.47%	7.29%	-47.42%	17.65%	20.09%	26.69%
	-6.74%	0.00%	-8.98%	-19.30%	-64.44%	-47.17%	0.51%	-102.99%	9.27%	11.11%	15.75%

TABLE VII
AGP, LGP, AND CD-GPHH TEST RESULTS (UNITS/H)

	Fixed	**Manual**	**Random**	**FIFO**	**STT**	**LTT**	**MTR**	**STR**	**AGP**	**LGP**	**CD-GPHH**
Set6	102.50	105.35	107.16	88.14	66.96	75.89	106.74	55.27	117.86	123.96	129.67
Set7	116.00	126.85	107.46	100.80	74.72	87.88	119.28	60.57	130.43	136.60	141.01
Set8	93.69	99.88	97.04	87.53	69.89	75.03	102.28	51.74	106.92	111.42	118.34
Set9	98.45	106.01	96.96	96.57	68.81	85.55	108.81	53.89	112.58	121.31	125.31
Set10	112.12	121.37	106.84	101.59	76.23	85.39	121.85	57.07	121.95	125.76	133.78
Mean	104.55	111.89	103.09	94.93	71.32	81.95	111.79	55.71	117.95	123.81	129.62
Imp.	0.00%	7.02%	-1.39%	-9.20%	-31.78%	-21.62%	6.92%	-46.72%	12.82%	18.42%	23.98%
	-7.02%	0.00%	-8.53%	-17.87%	-56.89%	-36.54%	-0.09%	-100.85%	5.14%	9.63%	13.68%

the listed individuals of ~~three GP algorithms, we can find that~~ CD-GPHH not only produced more efficient heuristics but also ~~had~~ much more readable ~~solutions~~ thanks to its double-layer structure. In contrast, AGP, and particularly~~,~~ LGP produced heuristics difficult to understand. ~~Because~~ these heuristics must often be further modified by the operator in practice, the heuristics produced by CD-GPHH have better usability.

~~In order to observe~~ the evolution process of the three GP algorithms~~, one result of~~ training set 4 ~~were~~ plotted in Fig. 12. It can be seen that for the real-world multi-scenario problem, the performance of AGP without scenario grouping is quite limited, while LGP and CD-GPHH can achieve better results. Especially, CD-GPHH did not suffer from the obvious limitations of AGP and performed best in the end.

C. Truck Dispatching Under Special Scenarios

Although problem instances based on real-life data are important to evaluate the practicality of the proposed method, they are less useful to ~~discover the~~ insights ~~of the problem because of~~ the real-life complexities and the combinatory effect of several uncontrolled factors. In this subsection, we evaluate the performance of different methods under three different scenarios created artificially. In a container terminal, the multiple scenarios are ~~caused~~ mainly ~~by~~ the following dynamically changing factors.

- ~~Distribution of~~ the load and unload QC tasks along the berth line~~. It is~~ practically ~~accepted that the operation times of the load and unload tasks~~ follow different distributions. The unloading tasks are often less likely disrupted by truck delays ~~because of~~ less strict precedence requirements. On the other hand, loading tasks must follow the predefined sequences exactly and hence delays can propagate exponentially, causing significant QC waiting. Therefore, different dispatch policies are required for scenarios with tasks dominated by either load or unload tasks.
- Distribution of operation nodes at yard cranes (YCs) for the tasks ~~are~~ also crucial. ~~When the~~ operation nodes ~~are clustered~~ at a few YCs~~, it is~~ more likely ~~to see~~ conflicts ~~arising at these YCs because they~~ support multiple QCs at the time. Specific policies are required to resolve these conflicts.
- The available number of trucks for dispatch is also important. When ~~plenty of~~ trucks are available, the priority should focus on the reduction of empty truck travel distances~~. However, when the number of trucks is relative small~~, the priority ~~would be about~~ avoiding costly QC waiting.

Following these considerations, we use $operation\_type$, $yard\_crane\_type$, and $total\_truck\_num$ to distinguish these scenarios respectively. This is also based on the observation ~~of~~ best-performing CD-GPHH individuals ~~that over 75% of them~~ use all three features in the scenario selection layer. Therefore, we created 3 new data sets (sets 11-13), each contain~~s~~ 20 instances with 2 special scenarios base on these 3 scenario features respectively. ~~First,~~ individuals were trained (30 runs per instance) with corresponding scenario features, then without. ~~cording to~~ statistics in Table VIII~~,~~ CD-GPHH~~'s superiority~~ ~~erformance become more obviously~~ in special scenarios data sets. ~~The~~ performance of AGP ~~is almost unaffected by the removal of~~ the scenario features~~ but~~ LGP ~~would have a~~ 2.9% ~~performance drop~~ when scenario features are excluded, compared with 8.1% from CD-GPHH~~, which proves~~ the effectiveness and importance of the scenarios identification used in ~~our~~ CD-GPHH ~~in~~ multi-scenario problems.

TABLE VIII
AGP, LGP, AND CD-GPHH ~~RESULT IN~~ SPECIAL SCENARIOS TRUCK DISPATCHING PROBLEM (UNITS/H)

Set	Feature	Manual	AGP	LGP	CD-GPHH
Set11	With	107.46	113.85	122.64	131.69
	Without	107.46	114.44	118.32	119.68
Set12	With	105.02	107.67	115.51	125.72
	Without	105.02	105.17	113.33	113.71
Set13	With	99.38	104.96	110.69	123.34
	Without	99.38	106.00	106.28	112.74
Mean	With	103.95	108.83	116.28	126.92
	Without	103.95	108.54	112.64	115.38
Imp.	With	0.00 %	4.48 %	10.60 %	18.09 %
	Without	0.00 %	4.23 %	7.72 %	9.90 %

Finally, ~~in this real-world container terminal truck dispatching simulating experiment,~~ It was confirmed that CD-GPHH can indeed produce better results than AGP and LGP in real-life multi-scenario problems~~, and~~ the generated results can be understood and then modified by operators. According to our statistics, the average test time and training time of each generation of AGP, LGP and CD-GPHH individual for 100 tasks is: 0.02s / 0.7s, 0.02s / 0.73s and 0.021s / 0.78s, respectively. ~~Which means that~~ CD-GPHH ~~does not greatly increase the~~ computational ~~consumption on the basis of AGP and LGP while improving performance~~. Although CD-GPHH is yet adopted in a real-life port, our manually crafted heuristic algorithm has been practiced ~~in~~ the Ningbo Port for years. ~~According to~~ statistical analyses conducted by the port~~,~~ work efficiency is increased by 8.1% and ship docking time decreased by 2.2%. This well-performed algorithm saved time, allowed ~~for the~~ operation of more ships, and in turn, increased the profit of the port company significantly. It is our next plan to work with the collaborators to fully evaluate and deploy the proposed algorithm in real world.

VI. CONCLUSION

~~We~~ proposed a novel cooperative double-layer genetic programming hyper-heuristic (CD-GPHH) algorithm that can evolve more efficient, user-friendly, and intuitive heuristics ~~and evaluated~~ its practicality and superiority ~~via~~ a real-world complex truck dispatching problem at marine container terminals. ~~We showed that~~ the proposed cooperative double-layer structure in ~~our~~ CD-GPHH can better handle the dynamics of scenario transitions~~ that~~ commonly~~ exist~~ in real-life~~ uncertainties~~. The separated scenario and calculation layers cooperated to utilise logic and arithmetic operators simultaneously while preventing its search space from growing exponentially. ~~Under~~ limited training time budget, the proposed CD-GPHH ~~performance~~ gains around 8-10% improvement compared with existing GP methods~~ (i.e. AGP and LGP)~~. ~~Our~~ CD-GPHH greatly enhances the ~~solution~~ usability with separate logic and arithmetic layers~~ and the resulting heuristics have a clear structure and better readability~~.

Our CD-GPHH can be further enhanced in future studies by addressing a few weaknesses, such as generalisation issues, relative inefficiency in rules evolution and existence of redundant subtrees. In particular, evolution may be better guided by involving human operators in the evolution process. Meanwhile, targeted redundancy removal algorithms can be developed specifically for CD-GPHH.

REFERENCES

[1] Y. Zhou, W. Daamen, T. Vellinga, and S. Hoogendoorn, "Review of maritime traffic models from vessel behavior modeling perspective," *Transportation Research Part C: Emerging Technologies*, vol. 105, pp. 323–345, 2019.

[2] J. Chen, R. Bai, H. Dong, R. Qu, and G. Kendall, "A dynamic truck dispatching problem in marine container terminal," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, IEEE, 2016.

[3] X. Chen, R. Bai, R. Qu, H. Dong, and J. Chen, "A data-driven genetic programming heuristic for real-world dynamic seaport container terminal truck dispatching," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2020.

[4] M. Ebner, "On the search space of genetic programming and its relation to nature's search space," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2, pp. 1357–1361, IEEE, 1999.

[5] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2013.

[6] M. A. Potter and K. A. D. Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary computation*, vol. 8, no. 1, pp. 1–29, 2000.

[7] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2012.

[8] N. X. Hoai, R. I. McKay, and D. Essam, "Representation and structural difficulty in genetic programming," *IEEE Transactions on evolutionary computation*, vol. 10, no. 2, pp. 157–166, 2006.

[9] Y. Bi, B. Xue, and M. Zhang, "Genetic programming with a new representation to automatically learn features and evolve ensembles for image classification," *IEEE transactions on cybernetics*, vol. 51, no. 4, pp. 1769–1783, 2020.

[10] P. Schonfeld and O. Sharafeldien, "Optimal berth and crane combinations in containerports," *Journal of waterway, port, coastal, and ocean engineering*, vol. 111, no. 6, pp. 1060–1072, 1985.

[11] K. H. Kim and Y.-M. Park, "A crane scheduling method for port container terminals," *European Journal of operational research*, vol. 156, no. 3, pp. 752–768, 2004.

[12] N. Kaveshgar and N. Huynh, "A genetic algorithm heuristic for solving the quay crane scheduling problem with time windows," *Maritime Economics & Logistics*, vol. 17, no. 4, pp. 515–537, 2015.

[13] O. A. Kasm, A. Diabat, and T. Cheng, "The integrated berth allocation, quay crane assignment and scheduling problem: mathematical formulations and a case study," *Annals of Operations Research*, pp. 1–27, 2019.

[14] F. Rodrigues and A. Agra, "Berth allocation and quay crane assignment/scheduling problem under uncertainty: a survey," *European Journal of Operational Research*, 2022.

[15] C. Tan and J. He, "Integrated proactive and reactive strategies for sustainable berth allocation and quay crane assignment under uncertainty," *Annals of Operations Research*, pp. 1–32, 2021.

[16] K. Lai and K. Lam, "A study of container yard equipment allocation strategy in hong kong," *International Journal of Modelling and Simulation*, vol. 14, no. 3, pp. 134–135, 1994.

[17] C. Zhang, J. Liu, Y.-w. Wan, K. G. Murty, and R. J. Linn, "Storage space allocation in container terminals," *Transportation Research Part B: Methodological*, vol. 37, no. 10, pp. 883–903, 2003.

[18] L. Chen, N. Bostel, P. Dejax, J. Cai, and L. Xi, "A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal," *European Journal of Operational Research*, vol. 181, no. 1, pp. 40–58, 2007.

[19] D. C. Hop, N. Van Hop, and T. T. M. Anh, "Adaptive particle swarm optimization for integrated quay crane and yard truck scheduling problem," *Computers & Industrial Engineering*, vol. 153, p. 107075, 2021.

[20] D. Kizilay and D. T. Eliiyi, "A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals," *Flexible Services and Manufacturing Journal*, vol. 33, no. 1, pp. 1–42, 2021.

[21] H. Javanshir and A. G. S. SEYED, "Yard crane scheduling in port container terminals using genetic algorithm," 2010.

[22] E. K. Bish, F. Y. Chen, Y. T. Leong, B. L. Nelson, J. W. C. Ng, and D. Simchi-Levi, "Dispatching vehicles in a mega container terminal," in *Container terminals and cargo systems*, pp. 179–194, Springer, 2007.

[23] H.-A. Lu and J.-Y. Jeng, "Modeling and solution for yard truck dispatch planning at container terminal," in *Operations Research Proceedings 2005*, pp. 117–122, Springer, 2006.

[24] Y.-L. Cheng, H.-C. Sen, K. Natarajan, C.-P. Teo, and K.-C. Tan, "Dispatching automated guided vehicles in a container terminal," in *Supply chain optimization*, pp. 355–389, Springer, 2005.

[25] E. Nishimura, A. Imai, and S. Papadimitriou, "Yard trailer routing at a maritime container terminal," *Transportation Research Part E: Logistics and Transportation Review*, vol. 41, no. 1, pp. 53–76, 2005.

[26] L. H. Lee, E. P. Chew, K. C. Tan, and Y. Wang, "Vehicle dispatching algorithms for container transshipment hubs," *OR spectrum*, vol. 32, no. 3, pp. 663–685, 2010.

[27] J. He, W. Zhang, Y. Huang, and W. Yan, "A simulation optimization method for internal trucks sharing assignment among multiple container terminals," *Advanced Engineering Informatics*, vol. 27, no. 4, pp. 598–614, 2013.

[28] J. He, C. Tan, and Y. Zhang, "Yard crane scheduling problem in a container terminal considering risk caused by uncertainty," *Advanced Engineering Informatics*, vol. 39, pp. 14–24, 2019.

[29] L. Zhen, L. H. Lee, and E. P. Chew, "A decision model for berth allocation under uncertainty," *European Journal of Operational Research*, vol. 212, no. 1, pp. 54–68, 2011.

[30] M. Sislioglu, M. Celik, and S. Ozkaynak, "A simulation model proposal to improve the productivity of container terminal operations through investment alternatives," *Maritime Policy & Management*, vol. 46, no. 2, pp. 156–177, 2019.

[31] Y. Zhang, R. Bai, R. Qu, C. Tu, and J. Jin, "A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties," *European Journal of Operational Research*, 2021.

[32] P. Pardalos and E. Romeijn, "Handbook of global optimization-volume 2: Heuristic approaches," 2002.

[33] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation," in *Workshops on Applications of Evolutionary Computation*, pp. 1–10, Springer, 2002.

[34] N. Pillay and R. Qu, *Hyper-heuristics: theory and applications*. Springer, 2018.

[35] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[36] D. Hermansyah and A. Muklason, "Evaluation of hyper-heuristic method using random-hill climbing algorithm in the examination timetabling problem," in *Journal of Physics: Conference Series*, vol. 1569, p. 022101, IOP Publishing, 2020.

[37] R. Bai, E. K. Burke, G. Kendall, and B. McCullum, "A simulated annealing hyper-heuristic for university course timetabling problem," *Abstract) PATAT*, vol. 6, 2006.

[38] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[39] D. Domović, T. Rolich, and M. Golub, "Evolutionary hyper-heuristic for solving the strip-packing problem," *The Journal of The Textile Institute*, vol. 110, no. 8, pp. 1141–1151, 2019.

[40] Y. Zhang, M. Harman, G. Ochoa, G. Ruhe, and S. Brinkkemper, "An empirical study of meta-and hyper-heuristic search for multi-objective release planning," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 1, pp. 1–32, 2018.

[41] Y. Yao, Z. Peng, and B. Xiao, "Parallel hyper-heuristic algorithm for multi-objective route planning in a smart city," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 10307–10318, 2018.

[42] W. Qin, Z. Zhuang, Z. Huang, and H. Huang, "A novel reinforcement learning-based hyper-heuristic for heterogeneous vehicle routing problem," *Computers & Industrial Engineering*, vol. 156, p. 107252, 2021.

[43] B. Chen, R. Qu, R. Bai, and W. Laesanklang, "A hyper-heuristic with two guidance indicators for bi-objective mixed-shift vehicle routing problem with time windows," *Applied Intelligence*, vol. 48, no. 12, pp. 4937–4959, 2018.

[44] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial intelligence through simulated evolution," 1966.

[45] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.

[46] J. Lin, L. Zhu, and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 140, p. 112915, 2020.

[47] L. Zhu, J. Lin, Y.-Y. Li, and Z.-J. Wang, "A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Knowledge-Based Systems*, vol. 225, p. 107099, 2021.

[48] E. Kieffer, G. Danoy, M. R. Brust, P. Bouvry, and A. Nagih, "Tackling large-scale and combinatorial bi-level problems with a genetic programming hyper-heuristic," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 44–56, 2019.

[49] H.-B. Song and J. Lin, "A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times," *Swarm and Evolutionary Computation*, vol. 60, p. 100807, 2021.

[50] B. Tan, H. Ma, Y. Mei, and M. Zhang, "A cooperative coevolution genetic programming hyper-heuristic approach for on-line resource allocation in container-based clouds," *IEEE Transactions on Cloud Computing*, 2020.

[51] B. Tan, H. Ma, and Y. Mei, "A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds," in *Australasian Joint Conference on Artificial Intelligence*, pp. 146–152, Springer, 2018.

[52] M. F. Brameier and W. Banzhaf, *Linear genetic programming*. Springer Science & Business Media, 2007.

[53] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *arXiv preprint cs/0102027*, 2001.

[54] T. Perkis, "Stack-based genetic programming," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 148–153, IEEE, 1994.

[55] S. Yan, H. Lin, and X. Jiang, "A planning model with a solution algorithm for ready mixed concrete production and truck dispatching under stochastic travel times," *Engineering Optimization*, vol. 44, no. 4, pp. 427–447, 2012.

[56] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*, pp. 449–468, Springer, 2010.

[57] D. Sobania, "On the generalizability of programs synthesized by grammar-guided genetic programming," in *European Conference on Genetic Programming (Part of EvoStar)*, pp. 130–145, Springer, 2021.

[58] J. M. Moyano and S. Ventura, "Auto-adaptive grammar-guided genetic programming algorithm to build ensembles of multi-label classifiers," *Information Fusion*, vol. 78, pp. 1–19, 2022.

[59] P. A. Whigham *et al.*, "Grammatically-based genetic programming," in *Proceedings of the workshop on genetic programming: from theory to real-world applications*, vol. 16, pp. 33–41, Citeseer, 1995.