Hybrid evolutionary computation methods for quay crane scheduling problems

Su Nguyen[a], Mengjie Zhang[a], Mark Johnston[b], Kay Chen Tan[c]

[a] *School of Engineering and Computer Science, Victoria University of Wellington, New Zealand*
[b] *School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, New Zealand*
[c] *Department of Electrical and Computer Engineering, National University of Singapore, Singapore*

Abstract

Quay crane scheduling is one of the most important operations in seaport terminals. The effectiveness of this operation can directly influence the overall performance as well as the competitive advantages of the terminal. This paper develops a new priority-based schedule construction procedure to generate quay crane schedules. From this procedure, two new hybrid evolutionary computation methods based on genetic algorithm (GA) and genetic programming (GP) are developed. The key difference between the two methods is their representations which decide how priorities of tasks are determined. While GA employs a permutation representation to decide the priorities of tasks, GP represents its individuals as a priority function which is used to calculate the priorities of tasks. A local search heuristic is also proposed to improve the quality of solutions obtained by GA and GP. The proposed hybrid evolutionary computation methods are tested on a large set of benchmark instances and the computational results show that they are competitive and efficient as compared to the existing methods. Many new best known solutions for the benchmark instances are discovered by using these methods. In addition, the proposed methods also show their flexibility when applied to generate robust solutions for quay crane scheduling problems under uncertainty. The results show that the obtained robust solutions are better than those obtained from the deterministic inputs.

Keywords: genetic programming, local search, quay crane scheduling

1. Introduction

Container terminals play an important role in modern sea-freight transportation. With the rapid annual growth rates of the shipped container volume, container terminals have become the bottlenecks in the global supply chain [1] and the effectiveness of the container terminal is an important factor for liner shipping companies to decrease their cost [2]. In addition, a container terminal also needs

to improve its service to compete with other terminals. In order to improve their productivity and customer satisfaction, it is important that the terminals can effectively utilise their expensive resources such as ship berthing areas, quay cranes, and yard cranes [3]. To support terminal operational decisions, many operations research methods have been proposed [4, 5, 6, 7].

Quay crane (QC) scheduling is one of the most important operations within a container terminal because the effectiveness of this activity can strongly influence the productivity of the entire container terminal. The aim of the quay crane scheduling problem (QCSP) is to find a good schedule for the loading/unloading operations of a vessel with a given number of quay cranes in order to minimise the overall vessel handling time (or makespan) [8]. An illustration of the working QCs at a vessel is shown in Figure 1. In this case, a number of QCs are allocated to the vessel for loading/unloading operations of containers (20ft or 40ft). All QCs move on a railway line parallel to the vessel and QCs are not allowed to cross each other. Different models focusing on different levels of complexity of the problems have been investigated. Bierwirth and Meisel [6] have provided a detailed classification of the existing models to handle QCSPs. Three main problem classes that are most popular in the research community are: (1) QCSP with container groups, (2) QCSP with complete bays (each bay is considered as a single task), and (3) QCSP with bay areas (a set of bays of a vessel is treated as a task to be exclusively handled by one QC) [6, 8].

The focus of this paper is QCSP with container groups in which containers of the same bay of the vessel are grouped as different tasks to be assigned to different QCs. In these problems, each task is located at a bay position of the vessel and the precedence constraints need to be satisfied due to the stacking dependent accessibility of tasks located in the same bay. To avoid congestion at the yard blocks, some tasks are also not allowed to be processed simultaneously [9]. Each QC has a ready time and an initial bay position. When operating, all QCs must not cross each other and two QCs cannot work at the same bay location and their safety distance (measured in unit of bays) have to be maintained [8]. The readers can refer to Kim and Park [10] and Bierwirth and Meisel [6] for detailed examples of QCSP. Many methods have been proposed in the literature to deal with these problems [9, 11, 12, 10, 13, 14]. However, there are still two major limitations with these methods. First, the running times of these methods increase rapidly as the problem size increases. Second, although some methods can provide very good solutions, they are not flexible enough to cope with practical requirements (e.g. coping with the uncertainty, integrating with other operation decisions).

1.1. Goals

The overall goal of this work is to develop new methods for QCSPs with container groups which can effectively find near-optimal schedules within reasonable

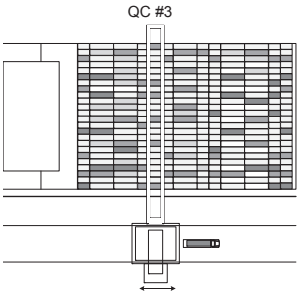


Figure 1: Illustration of QCs working on a vessel.

computational times and have the flexibility to taking into account different requirements of the real-world applications. Due to the effective search of near-optimal solutions for scheduling problems [11, 12], we propose new evolutionary computation methods to tackle QCSPs with container groups. The research objectives of this work can be summarised as follows:

1. Developing new hybrid methods that combine the advantages of evolutionary computation methods and a local search heuristic for QCSPs,

2. Comparing the proposed methods with the existing methods in the literature and analysing their advantages and disadvantages,

3. Extending the methods to handle the QCSPs with uncertain processing times, and

4. Analysing the behaviour of the proposed methods.

The novelty of the proposed methods is the use of a new priority-based schedule construction procedure, where the priorities are determined based on two representations that allow the proposed methods to simultaneously decide the assignments of tasks to quay cranes and the sequencing of tasks. In the first representation, the individual is represented as an order of tasks to be processed by the available quay cranes, which is usually found in applications of genetic algorithm (GA) [15] for scheduling problems. The second representation is a priority function in a tree form, which is widely used in genetic programming (GP) [16, 17]. Different from the individuals in the first representation that can be directly used as the priorities for task assignment and sequencing, the tree-based individuals will indirectly calculate the priorities of tasks based on the status/attributes of tasks and quay cranes at the moments that scheduling decisions need to be made. While GA as well as some other evolutionary computation methods such as particle swarm optimisation (PSO) or ant colony optimisation (ACO) has been applied regularly in the

scheduling literature (PSO and ACO have not been applied to solve QCSPs with container groups), GP is not a conventional method for these problems from the optimisation viewpoint. Therefore, it would be interesting to have a comparison of these two methods in this work. Two advantages of GA and GP are that they are easy to be implemented, and that they are flexible enough to be extended to deal with different objectives or to be integrated with other operation decisions. In order to further improve the quality of the scheduling solutions, we also introduce a new simple local search procedure to refine the schedules obtained by GA and GP. This paper also presents the first work that compares the performance of the simulation-optimisation methods with that of deterministic methods to handle QCSPs with uncertain handling times.

1.2. Organisation

The rest of this paper is organised as follows. An overview of existing methods used to deal with QCSPs are presented in Section 2 and we give a mathematical programming model of the considered QCSP in Section 3. In Section 4, details about the new hybrid methods are provided to show how they can be used to solve QCSPs. Section 5 and Section 6 show the experimental design and the results obtained by the proposed methods on a large number of benchmark instances. Section 7 extends the proposed methods to deal with QCSPs under uncertainty. Further discussion about the proposed methods is presented in Section 8. Finally, we provide conclusions and highlight future research directions.

2. Related work

Study of QCSPs were initiated with the early work of Daganzo [18] who investigated the problem with multiple vessels at a berth and cranes which can move freely. An exact method was provided to deal with small problems. Perterkofsky and Daganzo [19] proposed a branch-and-bound algorithm to solve real size problems. However, these studies did not consider interference between the quay cranes.

Kim and Park [10] further investigated this problem at a greater level of detail by dividing a task into smaller fractions (referred to as clusters or container groups) as compared to those from Daganzo [18] and Perterkofsky and Daganzo [19]. They also included realistic constraints such as quay crane ready times, non-crossing and precedence constraints in their model. A first formal mixed-integer linear program (MILP) was also presented in this work to avoid interference and later this was improved by Moccia et al. [13], Sammarra et al. [14] and Bierwirth and Meisel [9]. Unfortunately, QCSPs with the interference constraints are too complex to be solved to optimality by MILP solvers. Kim and Park [10] developed a branch-and-bound (B&B) algorithm to find optimal *non-delay* schedules with minimum

makespan (the completion time of the last task). However, B&B can only solve problems with fewer than 20 tasks. Moccia et al. [13] proposed a new Branch and Cut algorithm for QCSPs and showed that this algorithm can significantly improve solutions obtained by B&B [10] within a limited running time of two hours.

Since QCSP is an *NP-complete* problem [20], some heuristics and meta-heuristics have been proposed to find approximate solutions instead. Kim and Park [10] proposed a greedy randomised adaptive search procedure (GRASP) based on the non-delay schedule construction approach and a local search to improve the sequence of tasks handled by each QC. Sammarra et al. [14] applied Tabu Search (TS) and showed that it provided better results as compared to GRASP [10]. This method also significantly reduced the computational time compared to the Branch and Cut (B&C) algorithm [13] with only a slight deterioration in the solution quality. Chung and Choy [11] proposed a GA method in which an individual represents both the task-to-quay crane assignment and sequencing decisions. The performance of the GA method was better than that of GRASP but worse than that of TS [14] and B&C [13] as the problem size increases. Bierwirth and Meisel [9] developed a Unidirectional Scheduling (UDS) heuristic which employs a B&B algorithm to search for the optimal unidirectional schedule (cranes do not change the moving direction after any initial repositioning). The experimental results have shown that UDS outperformed the previous proposed methods both in term of quality and computational time. Kaveshgar et al. [12] proposed another GA method that can effectively find solutions that are very close to those obtained by UDS with shorter computation time when dealing with large benchmark instances [8].

Some variants of the QCSP have also been considered. Lee et al. [20] proposed a GA method for the QCSP with non-interference constraints (without precedence constraints) and showed that the proposed GA method can effectively find optimal or near-optimal solutions. Tavakkoli-Moghaddam et al. [21] extended the MILP model proposed by Kim and Park [10] to take into account the QC-to-vessel assignment decisions with total cost as the objective function. They also developed a GA method in which each individual represents the number of QCs assigned to a vessel and the sequence of tasks to be processed on a vessel. The gap between solutions from this GA method and the optimal solution is about 3% and their algorithm was able to handle larger instances. Legato et al. [22] developed a simulation-based optimisation method for QCSP based on simulated annealing. This method was shown to be effective when applied to practical scenarios.

3. Problem description

The mathematical formulation presented here for the QCSP with interference and precedence constraints is based on the MILP models developed by Bierwirth and Meisel [9]. This model is an extended version of the models developed by Kim

and Park [10], Moccia et al. [13], and Sammarra et al. [14] to properly avoid interferences of cranes.

The following notations are used in the mathematical formulation.

Indices

i, j	tasks which are ordered in an increasing order of their bay positions
k, υ, ω	quay cranes which are ordered in an increasing order of their bay positions

Problem data

p_j	handling time of task j
l_j	bay position of task j
l_0^k	initial bay position of quay crane k
r^k	ready time of quay crane k
$\hat{t}$	time to move a QC to move across a bay
δ	the smallest safety distance between two QCs
t_{ij}	travel time between position l_i and l_j calculated as $t_{ij} = \hat{t} \cdot \lvert l_i - l_j \rvert$
t_{0j}^k	time to move QC k from its initial position to l_j is calculated as $t_{0j}^k = \hat{t} \cdot \lvert l_0^k - l_j \rvert$
$\Delta_{ij}^{\upsilon\omega}$	minimum timespan to elapse between the processing of two tasks i and j which are respectively handled by QCs υ and ω

$$\Delta_{ij}^{\upsilon\omega} = \begin{cases} (l_i - l_j + \delta_{\upsilon\omega}) \cdot \hat{t} & \text{if } \upsilon < \omega \text{ and } i \neq j \text{ and } l_i > l_j - \delta_{\upsilon\omega} \\ (l_j - l_i + \delta_{\upsilon\omega}) \cdot \hat{t} & \text{if } \upsilon > \omega \text{ and } i \neq j \text{ and } l_i < l_j - \delta_{\upsilon\omega} \\ 0 & \text{otherwise.} \end{cases}$$

Sets of indices

Ω	set of tasks $\Omega = \{1, 2, \ldots, n\}$
Q	set of QCs $Q = \{1, 2, \ldots, q\}$ assigned to handle tasks in Ω
Φ	set of task pairs (i, j) (task j cannot start before the completion of task i)
Ψ	set of task pairs that cannot be processed simultaneously ($\Phi \subseteq \Psi$)
others	$\Omega^0 = \Omega \cup \{0\}$, $\Omega^T = \Omega \cup \{T\}$ and $\overline{\Omega} = \Omega \cup \{0, T\}$ where 0 and $T = n + 1$ are two dummy tasks with zero processing time are also included to indicate the beginning and the end of the service of the considered vessel
Θ	set of all combinations of tasks and QCs that potentially lead to crane interference $\Theta = \{(i, j, \nu, w) \in \Omega^2 \times Q^2 \mid (i < j) \wedge (\Delta_{ij}^{\upsilon\omega} > 0)\}$

Decision variables

x_{ij}^k	1, if tasks i and j are processed consecutively by QC k; otherwise $x_{ij}^k = 0$
z_{ij}	1 if and only if task j starts after task i is completed; otherwise $z_{ij} = 0$
c_i	completion time of task i. The objective to be minimised is the completion time of the final task, c_T

The QCSP is formulated as follows:

$$\text{minimise } c_T \tag{1}$$

$$\sum_{j \in \Omega^T} x_{0j}^k = 1 \quad (k \in Q) \tag{2}$$

$$\sum_{j \in \Omega^0} x_{jT}^k = 1 \quad (k \in Q) \tag{3}$$

$$\sum_{k\in Q}\sum_{j\in\Omega^T} x_{ij}^k = 1 \quad (i \in \Omega) \tag{4}$$

$$\sum_{j\in\Omega^0} x_{ji}^k - \sum_{j\in\Omega^T} x_{ij}^k = 0 \quad (i \in \Omega, k \in Q) \tag{5}$$

$$c_i + t_{ij} + p_j - c_j \leq M(1 - x_{ij}^k) \quad (i, j \in \overline{\Omega}, k \in Q) \tag{6}$$

$$c_i + p_j - c_j \leq 0 \quad ((i,j) \in \Phi) \tag{7}$$

$$c_i + p_j - c_j \leq M(1 - z_{ij}) \quad ((i,j) \in \Omega) \tag{8}$$

$$c_j + p_j - c_i \leq M z_{ij} \quad ((i,j) \in \Omega) \tag{9}$$

$$z_{ij} + z_{ji} = 1 \quad ((i,j) \in \Psi) \tag{10}$$

$$\sum_{u\in\Omega^0} x_{ui}^{\nu} + \sum_{u\in\Omega^0} x_{uj}^{\omega} \leq 1 + z_{ij} + z_{ji} \quad ((i,j,\upsilon,\omega) \in \Theta) \tag{11}$$

$$c_i + \Delta_{ij}^{\upsilon\omega} + p_j - c_j \leq (3 - z_{ij} - \sum_{u\in\Omega^0} x_{ui}^{\nu} - \sum_{u\in\Omega^0} x_{uj}^{\omega}) \quad ((i,j,\upsilon,\omega) \in \Theta) \tag{12}$$

$$c_j + \Delta_{ij}^{\upsilon\omega} + p_i - c_i \leq (3 - z_{ji} - \sum_{u\in\Omega^0} x_{ui}^{\nu} - \sum_{u\in\Omega^0} x_{uj}^{\omega}) \quad ((i,j,\upsilon,\omega) \in \Theta) \tag{13}$$

$$r^k + t_{0j}^k + p_j - c_j \leq M(1 - x_{0j}^k) \quad (i \in Q, j \in Q) \tag{14}$$

$$c_i \geq 0 \quad (i \in \overline{\Omega}) \tag{15}$$

$$x_{ij}^k \in \{0,1\} \quad (i,j \in \overline{\Omega}, k \in Q) \tag{16}$$

$$z_{ij} \in \{0,1\} \quad (i,j \in \overline{\Omega}) \tag{17}$$

Constraints (2)–(3) force each quay crane to handle only one first task and one last task, and each task can only be handled by one QC. Constraint (4) ensures that each task must be completed by exactly one QC. Constraints (5)–(10) are to guarantee a well-defined sequence of tasks. Constraints (11)–(14) ensure that the interference is avoided and the precedence relations between tasks are satisfied. Constraints (15)–(17) ensures the correctness of the decision variables. For detailed discussions, we refer to the studies of Kim and Park [10], Moccia et al. [13], Sammarra et al. [14], and Bierwirth and Meisel [9].

4. Methodology

In this section, we introduce the key elements of our proposed methods. We first present a priority-based schedule construction procedure. Then, we describe two representations of solutions/individuals evolved by our proposed methods. Finally, a summary of the proposed methods is provided.

4.1. Priority-based schedule construction procedure

Figure 2 shows our proposed procedure to construct a schedule for QCSPs. The procedure starts by partitioning set Ω into a set $\mathbb{T}$ of ready tasks and a set

1: $\mathbb{T} \leftarrow \{\tau \in \Omega | prec^{\tau} = 0\}$ and $\mathbb{U} \leftarrow \{\Omega \setminus \mathbb{T}\}$ *
2: $r_{\tau} \leftarrow +\infty$ for $\forall \tau \in \Omega$ with $prec^{\tau} > 0$
3: $c^k = r^k$ for $\forall k \in Q$
4: **while** $\mathbb{T}$ *is not empty* **do**
5: let $\mathcal{Q}$ the index of QC with $c^{\mathcal{Q}} = \min_{k \in Q} \{c^k\}$ **
6: let $\mathbb{T}' \leftarrow \{\tau \in \mathbb{T} | r_{\tau} \leq c^{\mathcal{Q}}\}$
7: assign a priority v_{τ} for each task $\tau \in \mathbb{T}'$
8: sort $\mathbb{T}'$ such that $\mathbb{T}' \leftarrow \{\tau_1, \ldots, \tau_{|\mathbb{T}'|}\}$ with $v_{\tau_i} > v_{\tau_{i+1}}$
9: **for** $i = 1$ to $|\mathbb{T}'|$ **do**
10: **if** (no interference when quay crane $\mathcal{Q}$ reach task τ_i) **then**
11: assign $\mathcal{Q}$ to handle τ_i
12: **else**
13: **if** (other QCs can move for quay crane $\mathcal{Q}$ to process τ_i) **then**
14: assign $\mathcal{Q}$ to handle τ_i
15: **end if**
16: **end if**
17: **if** (quay crane $\mathcal{Q}$ is able to process τ_i) **then**
18: update the completion time $c^{\mathcal{Q}} = c^{\mathcal{Q}} + \hat{t} \cdot |h_{\mathcal{Q}} - l_{\tau_i}| + p_{\tau_i}$
19: update the QC position $h_{\mathcal{Q}} = l_{\tau_i}$, update $prec^{\tau}$ for $\forall \tau \in \mathbb{U}$
20: $\mathbb{U}' \leftarrow \{\tau \in \mathbb{U} | prec^{\tau} = 0\}$ and set $r_{\tau} = c^{\mathcal{Q}}$ for $\forall \tau \in \mathbb{U}'$
21: move all task in $\mathbb{U}'$ from $\mathbb{U}$ into $\mathbb{T}$
22: remove task τ_i from $\mathbb{T}$, and set $c_{\tau_i} = c^{\mathcal{Q}}$
23: break the loop
24: **end if**
25: **end for**
26: **if** (quay crane $\mathcal{Q}$ cannot be assigned to handle any task) **then**
27: update the completion time $c^{\mathcal{Q}} = \min_{k \in \{Q \setminus \mathcal{Q}\}} \{c^k\}$
28: **end if**
29: **end while**
30: **return** $\max_{\tau \in \Omega} \{c_{\tau}\}$

* $prec^{\tau}$ indicates the number of tasks preceding τ that have not been completed
** if ties occur, select the quay crane $\mathcal{Q}$ that updates its completion time earlier.

Figure 2: Priority-based schedule construction procedure for QCSPs.

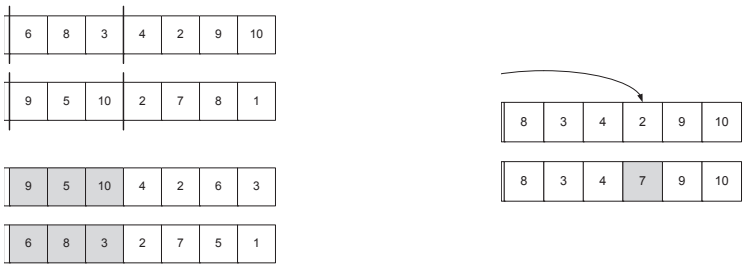
of unready tasks $\mathbb{U}$. A ready task is defined as a task which has the number of uncompleted precedence tasks $prec^{\tau}$ equal to zero. The ready time r_{τ} for each ready task is set to zero and the ready time c^k (or completion time of the previous task) for each quay crane is assigned based on the input data. This procedure will iteratively schedule a task in set $\mathbb{T}$. Within each iteration, quay crane $\mathcal{Q}$ with the earliest completion time is chosen to process the next task. If there are more than one QC with the same completion time, the one that updates its completion time earlier (in previous iterations) is chosen or the left most QC is selected if it is the first iteration and these QCs have the same ready time. A set $\mathbb{T}'$ of candidate tasks to be selected in this iteration is a subset of $\mathbb{T}$ which contains tasks with ready times earlier than the completion time of quay crane $\mathcal{Q}$ ($r_{\tau} \leq c^{\mathcal{Q}}$). Based on the

employed heuristics, a priority is assigned to each task in $\mathbb{T}'$ (more details on how the priorities are calculated will be shown in Section 4.2). Then, $\mathbb{T}'$ is sorted in a decreasing order of the priorities of tasks and task τ_i from the one with the highest priority is checked if it can be assigned to quay crane $\mathcal{Q}$ in this iteration. If there is no interference (crossing other QCs or violating the safety distance) occurring when quay crane $\mathcal{Q}$ attempts to reach the position of task τ_i, this task will be assigned to quay crane $\mathcal{Q}$. Otherwise, the procedure assigns quay crane $\mathcal{Q}$ to handle task τ_i if other QCs can shift away to avoid interference when quay crane $\mathcal{Q}$ processes task τ_i. If a task is assigned to quay crane $\mathcal{Q}$, the completion time $c^{\mathcal{Q}}$ and position $h_{\mathcal{Q}}$ of this quay crane are updated (based on the location and processing time of the selected task) and the values of $prec^{\tau}$ of unready tasks are adjusted to include the new ready tasks into $\mathbb{T}$. After quay crane $\mathcal{Q}$ is dispatched, task τ_i is removed from $\mathbb{T}$ and the procedure moves to the next iteration (break the loop). Otherwise, the next task τ_i in $\mathbb{T}'$ is examined. In case that no task can be assigned to quay crane $\mathcal{Q}$ in this iteration, the completion time of this quay crane will be updated to the earliest completion time of the other quay cranes. The procedure stops when all tasks are scheduled (set $\mathbb{T}$ is empty) and the completion time of the final task (makespan) is returned.

Basically, this procedure generates a schedule similar to the *non-delay* schedule generated by the *list scheduling* procedure proposed by Kim and Park [10]. The key difference is that our proposed procedure provides a more flexible way to assign tasks to quay cranes by selecting the task based on the priority of tasks instead of the first task in the list. Moreover, the quay crane with the earliest completion time may not always be assigned to a task immediately but can be delayed until the next earliest completion time of other quay cranes. As compared to the schedule construction procedure based on the disjunctive graph employed in UDS [9], our procedure is more flexible since it is able to generate bidirectional schedules where QCs can move in both directions during the schedule. This feature allows the search method to find potential solutions in the search space of bidirectional schedules, which cannot be found by UDS. However, since our procedure also restricts the idleness of QCs (mainly based on the earliest completion time), it cannot guarantee to provide an optimal solution.

4.2. Representations

Within the schedule construction procedure (see Figure 2), priorities need to be assigned to tasks in $\mathbb{T}'$. This is a crucial step of the whole procedure since it decides the sequence as well as the task-to-crane assignment and strongly influences the makespan of a schedule. In this work, we develop two representations for individuals (or solutions), which are respectively at the core of our proposed GA and GP methods and govern how the priorities are assigned.



(a) crossover (b) mutation

Figure 3: Crossover and mutation for the permutation-based representation.

4.2.1. Permutation representation for GA

In this representation, the solution is represented as an array of integers with the size equal to the number of tasks n. Each element in the array represents a task and the array represents the order in which tasks are supposed to be processed. The tasks located at the further left positions of the array will have higher priorities in the schedule construction procedure (step 7). New children are generated by partial-mapped crossover (PMX) [23] and two-point exchange mutation as shown in Figure 3, which are two popular genetic operations in GA, especially when applied to scheduling problems. For PMX, two random positions of the array are first selected and two substrings between two parents are exchanged. Then, the children are modified based on the mapping relationship between the two parents. Meanwhile the mutation is performed simply by randomly exchanging two positions of the array.

4.2.2. Tree representation for GP

In the tree representation, individuals are priority functions which are represented in tree-form. This representation is very popular in the GP research community to evolve computer programs for solving complex computational problems [16, 17]. Instead of a direct search for the priorities of tasks in the previous representation, we try to evolve priority functions that can synthesise the information of quay cranes and tasks at the decision moments to determine the priorities of tasks. The trees in this case are generated based on a terminal set and a function set. The terminal set contains terminals (leaf nodes) or variables/inputs of the problems which are connected by functions (internal nodes) in the function set. The descriptions of terminals used in our work are presented in Table 1. It is noted that the values for each terminal are different depending on the quay crane $\mathcal{Q}$, considered task τ and the state of quay cranes when the priorities are calculated in step 7 in Figure 2. The first five terminals in Table 1 are static values from the problem inputs. The next seven terminals are variables that characterise the status of quay crane $\mathcal{Q}$ and task τ. Random constants are also included in the terminal set. The

Table 1: Terminal set for the tree representation (for task τ and quay crane $\mathcal{Q}$)

Notation	Description	Value
P	task handling time	p_τ
T	moving speed to quay cranes	$\hat{t}$
S	safety distance between two QCs	δ
Q	number of QCs	$\|Q\|$
B	number of bays	b
D	distance between task τ and QC $\mathcal{Q}$	$\|l_\tau - h_\mathcal{Q}\|$
C	completion time of QC $\mathcal{Q}$	$c^\mathcal{Q}$
HWL	holding workload	$\sum_{\tau' \in S_\tau} p_{\tau'}$
LWL	local workload	$\sum_{\tau' \in \mathbb{L}} p_{\tau'}$
LQC	local quay crane	$\|Q^L\|$
DNQ	distance to the nearest quay crane $\mathcal{Q}'$	$\min_{k \in Q}\{\|c^k - l_\tau\|\}$
CNQ	completion time of the nearest quay crane $\mathcal{Q}'$	$c^{\mathcal{Q}'}$
#	ephemeral random constants (ERC) [17]	Uniform $[0, 1]$

* S_τ is the set of tasks that can only be processed when task τ is completed

* $\mathbb{L} \leftarrow \{\tau' \in \mathbb{T} \bigcup \mathbb{U} || l_\tau - l_{\tau'}| \leq \delta + 1\}$ contains remaining tasks in the radius of $(\delta + 1)$ of task τ

* Q^L is the number of QCs in the radius of $(\delta + 1)$ of task τ

function set will consist of standard mathematical operators $+, -, \times$, protected division % (similar to normal division but returning a value of 1 when division by 0 is attempted), min and max.

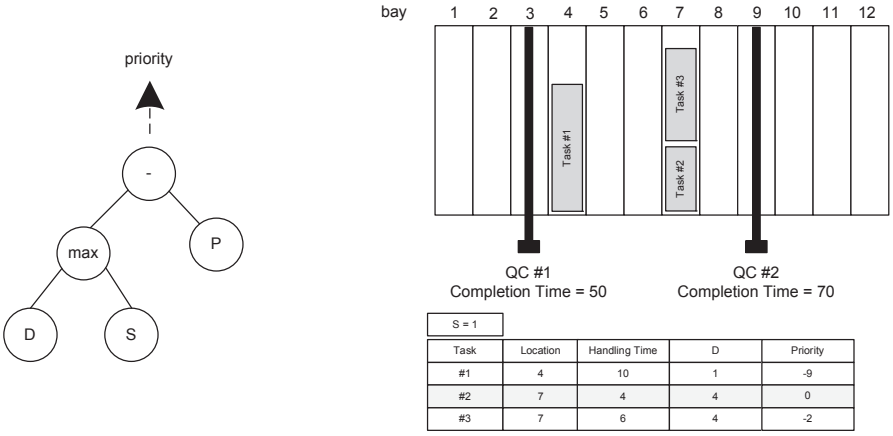
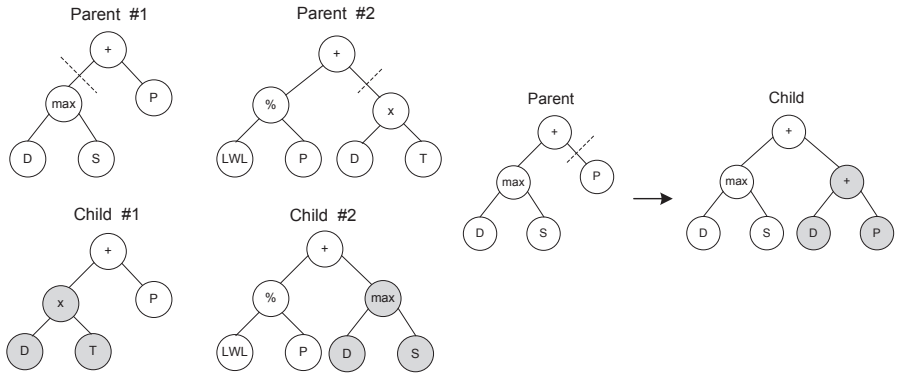


Figure 4: Illustration of tree representation for QCSPs.

An illustration of an individual with the tree representation is provided in Figure 4 along with a small example to show how it can be used to decide which task is to be processed next. In this example, the priority function obtained from the tree is $(\max(D, S) - P)$ and there are three tasks waiting to be processed. Since QC #1 is the one with the earliest completion time, it is selected for handling the next task.



(a) crossover (b) mutation

Figure 5: Crossover and mutation for the tree representation.

Based on the handling times (P) of tasks and the relative distance (D) between QC #1 and each task, the priorities are calculated. Because task #2 is the one that has the highest priority and there is no interference, it is handled by QC #1 in this iteration.

Crossover and mutation with the tree-based individual are very different from those used in the previous section. In this work, we employ GP subtree crossover and subtree mutation [17] which are popular in the GP community. For crossover, new individuals are created for the next generation by randomly recombining subtrees from two selected parents. Meanwhile, mutation is performed by randomly selecting a node of a chosen individual and replacing the subtree rooted at that node by a newly randomly-generated subtree. Illustrations of subtree crossover and subtree mutation are shown in Figure 5. The maximum depth for the newly generated individuals is restricted to eight in our work.

4.3. Local search heuristic

Even though GA and GP are powerful search methods, it is quite difficult for these methods to quickly converge to good solutions. For that reason, we also applied a simple local search heuristic (LSH) to refine the solutions obtained from the global search level by GA and GP. In our proposed LSH, we employ the permutation representation introduced in Section 4.2.1 as the solution representation. Therefore, if GA is used at the global search level, the solution can be directly transferred to LSH for further improvement. If GP is used, the order in which the tasks are processed with a GP individual is recorded as the input for the LSH. An overview of the LSH is shown in Figure 6. The neighbour solution Π' of the incumbent solution Π is generated by randomly swapping two tasks. Basically, the neighbour solution will be generated exactly the same way as the two point exchange mutation described in Section 4.2.1.

1: **Input:** sequence/order $\Pi \leftarrow \{\pi_1, \pi_2, \ldots, \pi_n\}$ of tasks to be processed
2: $\Pi^{lbest} \leftarrow \Pi$
3: $makespan^{lbest} \leftarrow$ makespan obtained from Π
4: **while** *stopping condition is not reached* **do**
5: $\quad \Pi' \leftarrow \Pi^{lbest}$
6: $\quad$ randomly select i and j from $[1, n]$ such that $i \neq j$
7: $\quad$ swap π_i and π_j in Π'
8: $\quad makespan^{new} \leftarrow$ construct a schedule from Π'
9: $\quad$ **if** $(makespan^{lbest} > makespan^{new})$ **then**
10: $\quad\quad \Pi^{lbest} \leftarrow \Pi'$ and $makespan^{lbest} = makespan^{new}$
11: $\quad$ **end if**
12: **end while**
13: **return** $makespan^{lbest}$

Figure 6: Local search heuristic for QCSPs.

4.4. Summary of the proposed methods

Figure 7 presents our general hybrid evolutionary computation approach, which can be employed by both GA and GP to deal with QCSPs. The method starts by reading the input data from an instance. Then, the initial population is randomly generated. In a generation, each individual is first applied to construct a schedule based on the procedure in Figure 2. After the schedule based on an individual is obtained, it is used as the input for the local search heuristic for further improvement. The (improved) makespan returned from local search is then used as the fitness of the individual. After all individuals are evaluated and the stopping condition is not reached, the new population is generated by using genetic operations and a new generation is started. The key difference between the GA-based method and the GP-based method is related to the three processes highlighted in Figure 7 by a bold frame. For GA, the initial individuals are generated by creating random permutations of tasks. Meanwhile, the initial population of GP is generated using the ramped-half-and-half method [17]. Since GA and GP use two different representations (permutation and tree), the ways that their individuals are used to construct schedules and the genetic operations applied to generate new individuals are also very different as discussed in Section 4.2.1 and Section 4.2.2. For ease of presentation, we simply call the versions of the hybrid methods based on GA and GP as HGA and HGP respectively in the rest of this paper.

5. Experimental design

5.1. Datasets

In this section, we test the two proposed methods HGA and HGP using the benchmark instances introduced by Kim and Park [10] and the large number of

benchmark instances proposed by Meisel and Bierwirth [8]. The instances in [8] were classified into seven sets which focus on different problem characteristics. The first three sets (A, B, and C) represent the problems with two QCs serving small, medium and large vessels. The scale of the vessel is decided by its number of bays. The number of tasks in these three sets ranges from 10 to 100 tasks. Sets D, E, F and G are for medium sized vessels with different handling rates (the percentage of containers handled in a service against the total vessel capacity), distributions of tasks on the vessel, numbers of QCs, and safety distances.

5.2. Parameter settings

Since the population size and stopping condition of the local search heuristic are two important factors that influences both computational times and solutions'

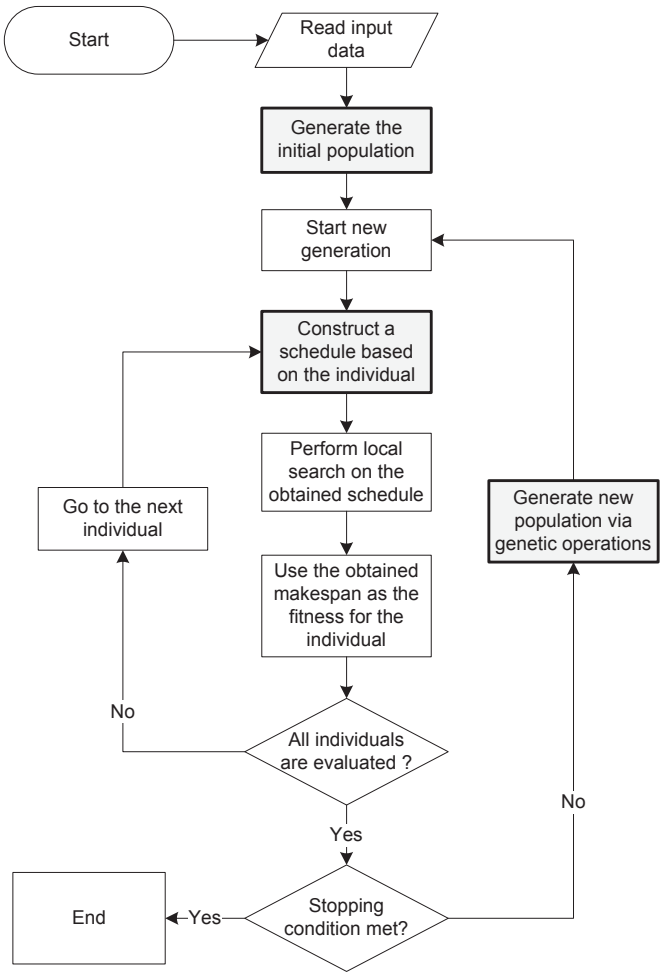
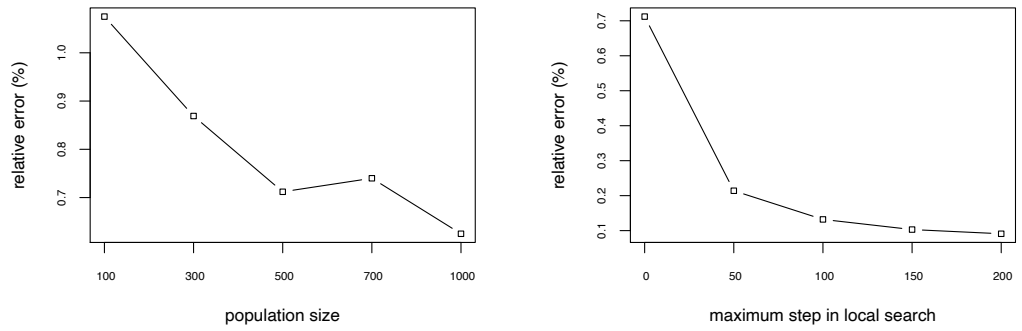


Figure 7: Proposed hybrid evolutionary computation approach for QCSPs

quality, we will perform some pilot experiments to find the suitable settings for these two parameters. Figure 8 shows the performance of HGA (HGP also shows similar behaviour) with different population sizes and stopping conditions (maximum number of steps) for the local search heuristic. In these experiments, five independent runs of HGA (in 10 seconds) were performed on the largest class of Set A [8] (with 40 tasks) for each configuration. These preliminary results indicate that a population size of 500 and maximum steps of 100 provide reasonable performance. Increasing the values of these parameters can improve the quality of the solutions slightly but will significantly increase the computational time, especially for larger instances.



(a) (b)

Figure 8: Influence of population size and maximum steps in the local search heuristic

The parameters used in both HGA and HGP are the same. The population size for the two methods are 500 and tournament selection with a tournament size of seven is used for individual selection. Although bidirectional schedules can provide better makespans as compared to unidirectional schedules, the search space of bidirectional schedules is much larger than that of unidirectional schedules. For that reason, it is faster and easier to search for good unidirectional schedules in large instances. To create a balance between the search time and solution quality, we use half of the population (250 individuals) to build bidirectional schedules and the other half to build unidirectional schedules. The unidirectional schedules can be easily created using the procedure in Figure 2 by adding a condition to forbid QCs from moving backward and bypassing a task that cannot be approached by any other QC. The crossover, mutation and reproduction rates are 80%, 10% and 10% respectively. These values are chosen after some preliminary experiments. Since the local search heuristic is used with GA and GP, the mutation rate is set higher than usual to avoid solutions trapped at the local optima. The details about the crossover and mutation operations for HGA and HGP were explained in Section 4. When reproduction (elitism) is applied, an individual is selected from the population by tournament selection and copied to the population of the next generation.

HGA and HGP are coded in Java and run on Intel Core i5-2400 3.10 GHz CPUs, single thread. The running time limits for HGA and HGP are 60 seconds for Kim and Park's instances [10] and ten independent runs for HGA and HGP are performed for each instance. The results from this set of instances are compared with those obtained by B&B and GRASP [10], B&C [13], tabu search [14], and UDS [9]. In Meisel's instances [8], running time limits are set as 10, 300, and 600 seconds for the small, medium and large vessels respectively and five independent runs of each method are performed for each instance. The results obtained from the two methods are compared to those obtained by UDS [9], GA [12], and the optimal solutions or lower bounds provided by using CPLEX to solve the MILP model [9, 8].

6. Results

Table 2 shows the results for instances $k13 - k49$ [10]. The column Opt.* gives the optimal makespans (or lower bounds) f_{opt} for each instance. The values reported for each method are the relative error in percentage $RE = (f_{best} - f_{opt})/f_{opt} \times 100$ where f_{best} is the makespan obtained by each method. For HGA and HGP, we show the average relative error $ARE(\%)$ and the best found makespan bf obtained from ten runs for each instance. It is easy to see that HGA and HGP can provide very good results as compared to other methods. The relative errors obtained by HGA and HGP are less than 4% for all instances. On average, the relative errors from our proposed methods are clearly better B&B, GRASP, and TS and very competitive as compared with B&C and UDS. The B&C method is slightly worse than HGP and slightly better than HGA. The results from HGP are only slightly worse than UDS. In addition, the proposed methods are able to discover new best known solution (instance with ▲ next to the relative errors) for three instances.

The average computational times for each group of instances are shown in Table 3. For our methods, we show the average and maximal times to find the best results. Even though each method is performed with different computer configurations, it is very obvious that our proposed methods are much faster than most methods and only slower than UDS in these instances. In general, these results have confirmed that our proposed methods are not only effective but also very efficient. Although our methods and those from Kim and Park [10] are both based on non-delay schedules, the priority-based schedule construction procedure and the hybrid search mechanism make our methods much more competitive.

A summary of the results for different classes of instances [8] obtained by HGP and HGA is shown in Table 4 and the computational times (in seconds) are presented in Table 5 (the detailed results for each instance are available at `https://ecs.victoria.ac.nz/foswiki/pub/Groups/ECRG/OnlineSupplimentaryMaterials/qcspAppendix.pdf`). It is clear that our proposed methods can obtain good results

Table 2: Comparison of HGA and HGP with other methods ($k13 - k49$ [10])

Instance	Opt.*	KP [10]		MCGL [13]	SCLM [14]	BM [9]	HGA		HGP	
		B&B	GRASP	B&C	TS	UDS	$ARE(\%)$	bf	$ARE(\%)$	bf
$k13$	453	0.00	0.00	0.00	0.00	0.00	0.00	453	0.00	453
$k14$	546	0.00	0.00	0.00	0.00	0.00	0.00	546	0.00	546
$k15$	513	0.00	0.00	0.00	0.00	0.00	0.00	513	0.00	513
$k16$	312	2.88	2.88	0.00	0.00	0.00	0.00	312	0.00	312
$k17$	453	0.66	0.66	0.00	0.00	0.00	0.00	453	0.00	453
$k18$	375	0.00	0.00	0.00	0.00	0.00	0.00	375	0.00	375
$k19$	543	1.66	1.66	0.00	0.00	0.00	0.00	543	0.00	543
$k20$	399	20.30	20.30	0.00	0.00	0.00	0.00	399	0.00	399
$k21$	465	0.00	0.00	0.00	0.00	0.00	0.00	465	0.00	465
$k22$	537	34.08	34.08	0.00	0.00	0.56	0.56	540	0.56	540
$k23$	576	0.00	2.60	0.00	1.04	0.00	0.05	576	0.00	576
$k24$	666	0.45	1.35	0.00	0.45	0.00	0.00	666	0.00	666
$k25$	738	0.00	0.41	0.00	0.41	0.00	0.00	738	0.00	738
$k26$	639	0.00	1.88	0.00	0.00	0.00	0.00	639	0.00	639
$k27$	657	0.00	4.57	0.00	0.46	0.00	0.00	657	0.00	657
$k28$	531	1.13	3.39	0.00	0.00	0.00	0.00	531	0.00	531
$k29$	807	0.00	1.49	0.00	0.37	0.00	0.00	807	0.00	807
$k30$	891	0.00	1.68	0.00	0.00	0.00	0.00	891	0.00	891
$k31$	570	0.00	0.00	0.00	0.00	0.00	0.00	570	0.00	570
$k32$	591	0.00	1.02	0.00	0.00	0.00	0.00	591	0.00	591
$k33$	603	0.00	10.45	0.00	0.00	0.00	0.05	603	0.25	603
$k34$	717	0.00	6.28	0.00	2.51	0.00	1.46	723	0.38	717
$k35$	684	0.88	2.19	0.00	0.88	0.00	1.23	684	0.13	684
$k36$	678	6.19	4.42	0.00	0.44	0.00	0.00	678	0.13	678
$k37$	510	1.18	5.88	0.00	1.76	0.00	0.06	510	0.00	510
$k38$	613.7	3.15	7.55	0.71	0.71	0.71	0.61▲	615	0.51▲	615
$k39$	508.4	8.58	13.89	0.91	2.09	0.91	0.91	513	0.91	513
$k40$	564	2.13	5.85	0.00	0.53	0.00	0.53	567	0.37	564
$k41$	585	11.78	9.73	0.50	1.53	0.50	0.40▲	585	0.00▲	585
$k42$	560.3	4.94	18.86	1.73	2.80	2.26	3.50	579	3.34	579
$k43$	859.3	10.67	9.62	4.38	2.29	1.94	1.80▲	873	2.22	876
$k44$	820.4	7.15	4.59	0.20	1.66	0.20	1.70	831	0.60	822
$k45$	824.9	4.38	5.83	1.83	3.29	1.11	2.09	840	1.87	840
$k46$	690	2.61	6.52	0.00	0.00	0.00	0.74	690	0.70	690
$k47$	792	15.15	1.89	0.00	0.00	0.00	1.10	792	0.00	792
$k48$	628.9	6.38	6.38	2.56	5.43	1.61	2.33	639	1.90	639
$k49$	879.2	4.07	10.55	5.43	3.73	1.68	2.19	897	2.53	897
Average		4.06	5.63	0.49	0.88	0.31	0.58		0.44	

Table 3: Average computational times (in minutes) ($k13 - k49$ [10])

Instances	KP [10]		MCGL [13]	SCLM [14]	BM [9]	HGA		HGP	
	B&B	GRASP	B&C	TS	UDS	*avg.*	*max.*	*avg.*	*max.*
$k13 - k22$	0.44	0.35	1.10	1.52	1.12×10^{-5}	0.01	0.02	0.01	0.01
$k23 - k32$	17.53	1.46	8.91	5.86	3.68×10^{-5}	0.04	0.13	0.03	0.08
$k33 - k42$	564.47	3.16	72.19	21.75	6.26×10^{-4}	0.18	0.38	0.20	0.51
$k42 - k49$	809.73	7.56	102.49	48.68	3.43×10^{-3}	0.57	0.89	0.39	0.75
CPU	P2, 466 MHz		P4, 2.5GHz	P4, 2.5GHz	P4, 2.8GHz	i5-2400, 3.10 GHz			

with only very small errors as compared to UDS (less than 0.6% for HGA and less than 0.5% for HGP). In addition, the computational times of the proposed methods are much shorter than those from UDS for many classes of instances. When tested with different handling rates f in set D, HGA and HGP show very good results and are able to beat UDS in several instances with the small handling rate ($f = 0.2$). For instances with the high handling rate ($f = 0.8$), the results from our proposed methods and UDS are competitive. Regarding the results in other sets, it is noted that the proposed methods perform very well on instances with more quay cranes and large safety distances. These experimental results are able to confirm the effectiveness and efficiency of the proposed methods when dealing with different scenarios. The proposed methods are also able to improve the best known solutions (BKS) of some instances as compared to CPLEX and UDS. From the experiments, we also see that HGA and HGP outperform GA [12] in both computational time and solution quality.

Table 4: Average % relative errors of HGA and HGP compared to UDS for Sets $A - G$ [8]

Sets	A	B	C	D	E	F	G
HGA	0.03	0.56	0.55	0.26	0.42	0.21	0.19
HGP	0.00	0.46	0.49	0.21	0.30	0.12	0.18
# of new BKS	3	3	20	12	0	7	5

Table 5: Average computational times (in seconds) for Sets $A - G$ [8]

Sets	A	B	C	D	E	F	G
UDS	0.00	1451.33	3541.33	1416.16	1331.2	1590	2828.21
HGA	1.37	115.89	320.25	74.23	88.51	63.38	51.90
HGP	0.63	120.06	282.36	82.23	84.66	55.28	45.76

7. Simulation-optimisation methods for QCSPs

The previous section has shown that HGA and HGP are effective methods to deal with QCSPs. Although there are small gaps in the solution quality compared to UDS, the proposed methods have some practical advantages. First, the computational times for the proposed methods are short, which allows us to obtain good solutions within reasonable computational times even for large instances. Second, the proposed methods are quite flexible and easy to extend to cope with practical requirements such as simultaneously handling berth allocation and quay crane scheduling. In this section, we extend the use of the proposed methods to create *robust* schedules (task sequences) for QCSPs. This is motivated by the fact that there are always some types of uncertainty occurring when QCs are operating at a vessel such as unreliable handling times caused by operators' skills and breakdowns, or the delay of yard trucks. Within the QCSP literature, there have not

been many studies on this topic. Most recently, Han et al. [24] developed a GA method to deal with simultaneous berth and quay crane allocation with stochastic arrival and handling times. However, the detailed task to QC scheduling problems (like the QCSPs in our work) were not considered in their work. Legato et al. [22] proposed a simulation-optimisation (SO) method to deal with a special scenario of quay crane scheduling and compared the obtained schedules to one generated by the terminal planner. In our work, we investigate robust schedules under different scenarios/instances and different levels of random variation in handling times.

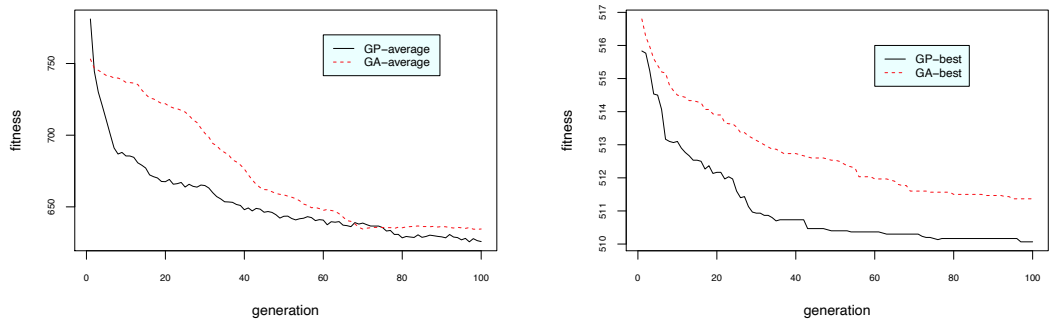
Different from the deterministic case where the goal is to find the schedule with the minimal makespan, we try to find a robust solution (task priorities or order of tasks to be handled) that minimises the expected makespan $E[c_T]$ when dealing with QCSPs under uncertainty. In these problems, the makespan is determined via stochastic simulation, in which the handling time of a task follows a uniform distribution $p'_j \sim U[(1-\gamma)p_j, (1+\gamma)p_j]$, where γ is the noise factor that decides how the handling time deviates from the average handling time p_j. A higher γ means that there can be larger differences between p'_j and p_j. In the dynamic operating condition of the simulation, quay cranes handle tasks based on the priority-based construction procedure in Figure 2. To estimate the expected makespan of a solution, we run multiple simulation replications based on the input data. The fitness of HGA and HGP is also the expected makespan obtained from a number of simulation scenarios (sample) in order to evolve robust solutions that are able to cope with possible changes of the problems. In order to evaluate the performance of the simulation-optimisation methods based on HGA (SO-HGA) and HGP (SO-HGP), we will test them using the input data in the instances in set A [8]. We will compare the solutions obtained by these two methods with the best known deterministic solutions (BKDS) found methods employed in the previous section. For each instance, the expected makespan for the comparison is obtained through 10,000 simulation replications. The sample size to determine fitness values for SO-HGA and SO-HGP is 100. Since the fitness evaluation in SO-HGA and SO-HGP is much more expensive than the original HGA and HGP, we will not apply the local search heuristic to each individual in the population but only to the best solution obtained by the methods at the end of each generation.

Five independent runs of 60 seconds for each method are performed for each instance in set A under different levels of noise and the results are summarised in Table 6. Values in this table are the relative deviation (in %) of the expected values and standard deviations of makespans ($RE_{E[c_T]}$ and $RE_{\sigma_{c_T}}$) as compared to those of BKDSs. The results show that solutions obtained by SO-HGA and SO-HGP are better than BKDS even with small noise of 5%. While the improvements in the expected makespan made by SO-HGA and SO-HGP are not large, the solutions obtained by the simulation-optimisation methods are much more robust since the

Table 6: Performance of simulation-optimisation methods (Set A [8])

	SO-HGA		SO-HGP	
	$RE_{E[c_T]}$	$RE_{\sigma_{c_T}}$	$RE_{E[c_T]}$	$RE_{\sigma_{c_T}}$
$\gamma = 5\%$				
$n = 10$	−1.06	−15.74	−1.06	−15.46
$n = 15$	−0.72	−9.88	−0.62	−6.07
$n = 20$	0.08	0.74	0.21	4.05
$n = 25$	−0.24	−5.05	−0.22	−4.41
$n = 30$	−0.05	−5.36	−0.06	−10.78
$n = 35$	0.11	−0.57	0.25	17.10
$n = 40$	0.09	−8.36	−0.20	−14.00
$\gamma = 10\%$				
$n = 10$	−1.36	−13.93	−1.38	−13.35
$n = 15$	−0.78	−9.10	−0.71	−10.07
$n = 20$	0.04	−0.23	0.11	0.07
$n = 25$	−0.32	−4.71	−0.30	−5.54
$n = 30$	−0.25	−11.32	−0.16	−11.56
$n = 35$	−0.15	−5.18	−0.06	−7.61
$n = 40$	0.03	−11.22	−0.28	−12.93
$\gamma = 15\%$				
$n = 10$	−1.47	−9.68	−1.41	−6.66
$n = 15$	−0.96	−9.55	−0.87	−9.32
$n = 20$	−0.07	−1.27	−0.03	−1.51
$n = 25$	−0.44	−5.05	−0.33	−5.97
$n = 30$	−0.46	−10.85	−0.42	−10.36
$n = 35$	−0.44	−8.23	−0.36	−8.83
$n = 40$	−0.17	−10.98	−0.39	−13.66
$\gamma = 20\%$				
$n = 10$	−1.65	−9.37	−1.63	−9.67
$n = 15$	−1.27	−10.85	−1.16	−10.32
$n = 20$	−0.19	0.29	−0.08	−1.55
$n = 25$	−0.66	−5.57	−0.42	−5.75
$n = 30$	−0.66	−9.05	−0.56	−10.05
$n = 35$	−0.62	−11.31	−0.47	−8.59
$n = 40$	−0.40	−13.61	−0.51	−13.50
$\gamma = 25\%$				
$n = 10$	−1.69	−7.81	−1.71	−7.80
$n = 15$	−1.43	−7.13	−1.51	−9.75
$n = 20$	−0.49	−4.12	−0.39	−4.85
$n = 25$	−0.96	−6.98	−0.70	−7.99
$n = 30$	−0.92	−9.80	−0.79	−9.61
$n = 35$	−0.88	−12.12	−0.80	−11.81
$n = 40$	−0.65	−13.91	−0.79	−15.22
$\gamma = 30\%$				
$n = 10$	−1.58	−5.06	−1.65	−5.74
$n = 15$	−1.91	−10.39	−1.91	−12.14
$n = 20$	−0.82	−4.57	−0.74	−5.45
$n = 25$	−1.20	−6.45	−0.87	−6.66
$n = 30$	−1.20	−10.46	−1.07	−10.69
$n = 35$	−1.15	−12.75	−0.96	−10.60
$n = 40$	−0.93	−15.19	−1.05	−14.88

standard deviation values of makespan are significantly smaller than those obtained in the deterministic case. This feature makes the solutions obtained by the SO methods more reliable as applied in practical situations. As the noise increases



(a) average fitness (b) best fitness

Figure 9: Behaviour of GA and GP for a single instance of QCSP

from 5% to 30%, the improvement in the expected makespan gained by the SO methods also increases. This confirms the importance of the proposed methods in order to create more effective and reliable solutions for QCSPs under uncertainty.

8. Further discussions

In Section 6, we have seen that the performance of HGP is better than HGA when tested on a large number of problem instances. Since both HGA and HGP use the same local search heuristic, these results must be achieved through the search mechanisms of GA and GP. Therefore, it would be interesting to understand how better performance is obtained by GP. We use an instance with 40 tasks (the first instance of this class in set A [8]) as the example to show the behaviour of GA and GP without the support of the local search heuristic. The parameters for this experiment are the same as those in the previous experiments but the two methods stop when they reach 100 generations. The results are shown in Figure 9 and the values in these figures are averaged from 30 independent runs of GA and GP.

It is obvious that GP performs better than GA both in terms of the average fitness and the best fitness. For the average fitness, it is interesting that the initial populations of GP are actually worse than that of GA but GP can converge very quickly to good solutions while GA can only converge roughly after 70 generations. Regarding the best fitness, it is also easy to see that GP finds better solutions as compared to GA even though the average fitness of GP initial populations are not as good as those from GA. An explanation for the good performance of GP is that GP tries to find the dispatching rules to solve the problem instead of specific solutions for the problem. Since dispatching rules (represented by GP individuals) define how all tasks should be processed, poor rules will provide very bad results, which are different from GA individuals whose quality depends on each gene. Because of this property, it is easier for GP to distinguish good rules from bad rules than GA to distinguish between good and bad solutions, and therefore GP can quickly improve the fitness of its population. The only drawback of GP in this experiment

is the computational time. While GA only needs 0.8 second on average to finish 100 generations, GP needs 4.5 seconds. This issue comes from the fact that GP needs more time to evaluate the tree/individual to obtain the priorities in each iteration of the schedule construction procedure. However, when the local search heuristics are applied, this gap in the computational time between GA and GP is very small as compared to the total computational time. This explains why HGP can provide better results with similar computational times as HGA in the experiments in Section 6.

Although the proposed methods are able to outperform most methods in the literature, they are still slightly worse than UDS in term of quality. There are two reasons for this issue. First, the proposed methods search in the search space of bi-directional schedules, which are much larger than the search space of UDS. Since most near-optimal solutions or optimal solutions in the benchmark instances can be found in the unidirectional search space, UDS is more competitive. A solution for this issue is to improve the priority-based construction heuristics to restrict the search space. Second, it is difficult for the local search heuristic within the proposed methods to find local optima with limited random steps especially in large and hard instances (e.g. complicated precedence constraints). Therefore, it is important for future study to develop exact and efficient local search heuristics to improve the effectiveness of HGA and HGP.

9. Conclusions

This paper proposes new hybrid evolutionary computation methods for QCSPs. In the proposed methods, schedules are generated by using a priority-based schedule construction procedure. Two representations based on GA and GP are proposed. A local search heuristic is also developed to improve the solutions obtained by GA and GP. The experimental results have shown that the proposed methods are very effective and efficient as compared to the existing methods in the literature. Moreover, new best known solutions to the benchmark instances are also found by the proposed methods and reported in this paper. One of the key advantages of the proposed methods is their flexibility, which allows them to easily handle practical requirements, compared to other methods.

In this work, we have also shown the importance of taking into account uncertainty in order to improve the effectiveness of the solutions when applied to practical situations. To tackle these problems, the proposed methods are extended to simulation-optimisation methods. The experimental results showed that the solutions obtained by the simulation-optimisation methods are better, both in terms of quality and reliability, compared to the optimal (or near-optimal) solutions obtained based on the deterministic assumption. These results indicate that more attention should be paid to handle uncertainty which is inevitable in practice.

In future studies, it would be interesting to adapt the hybrid methods to take into account other decisions such as berth and quay crane allocation or yard truck dispatching. Different strategies to handle uncertainty should also be considered and compared to understand their advantages and disadvantages. Also, the proposed methods can be easily extended to cope with multiple objectives. Regarding GP in this work, we have shown that it is a very promising method for solving QCSPs as compared to GA. Because the solution from GP can be applied to unseen instances, it would be interesting to further investigate how to take advantage of this method to boost the performance of other optimisation methods by, for example, reusing the evolved dispatching rules to create a set of potential initial solutions.

References

[1] M. E. Petering, K. G. Murty, Effect of block length and yard crane deployment systems on overall performance at a seaport container transshipment terminal, Computers & Operations Research 36 (5) (2009) 1711–1725.

[2] Q. Zeng, Z. Yang, Integrating simulation and optimization to schedule loading operations in container terminals, Computers & Operations Research 36 (6) (2009) 1935–1944.

[3] Z. Lu, X. Han, L. Xi, A. L. Erera, A heuristic for the quay crane scheduling problem based on contiguous bay crane operations, Computers & Operations Research 39 (12) (2012) 2915–2928.

[4] I. F. A. Vis, R. M. B. M. de Koster, Transshipment of containers at a container terminal: An overview, European Journal of Operational Research 147 (1) (2003) 1–16.

[5] D. Steenken, S. Voß, R. Stahlbock, Container terminal operation and operations research - a classification and literature review, OR Spectrum 26 (2004) 3–49.

[6] C. Bierwirth, F. Meisel, A survey of berth allocation and quay crane scheduling problems in container terminals, European Journal of Operational Research 202 (3) (2010) 615–627.

[7] R. Stahlbock, S. Voß, Operations research at container terminals: a literature update, OR Spectrum 30 (2008) 1–52.

[8] F. Meisel, C. Bierwirth, A unified approach for the evaluation of quay crane scheduling models and algorithms, Computers & Operations Research 38 (3) (2011) 683–693.

[9] C. Bierwirth, F. Meisel, A fast heuristic for quay crane scheduling with interference constraints, Journal of Scheduling 12 (4) (2009) 345–360, ISSN 1094-6136.

[10] K. H. Kim, Y.-M. Park, A crane scheduling method for port container terminals, European Journal of Operational Research 156 (3) (2004) 752–768.

[11] S. Chung, K. Choy, A modified genetic algorithm for quay crane scheduling operations, Expert Systems with Applications 39 (4) (2012) 4213–4221.

[12] N. Kaveshgar, N. Huynh, S. K. Rahimian, An efficient genetic algorithm for solving the quay crane scheduling problem, Expert Systems with Applications (2012) (online).

[13] L. Moccia, J.-F. Cordeau, M. Gaudioso, G. Laporte, A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal, Naval Research Logistics 53 (1) (2006) 45–59.

[14] M. Sammarra, J.-F. Cordeau, G. Laporte, M. F. Monaco, A tabu search heuristic for the quay crane scheduling problem, Journal of Scheduling 10 (4-5) (2007) 327–336, ISSN 1094-6136.

[15] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., 1st edn., 1989.

[16] W. Banzhaf, P. Nordin, R. Keller, F. Francone, Genetic Programming: An Introduction, Morgan Kaufmann, San Francisco, 1998.

[17] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992.

[18] C. F. Daganzo, The crane scheduling problem, Transportation Research Part B: Methodological 23 (3) (1989) 159–175.

[19] R. I. Peterkofsky, C. F. Daganzo, A branch and bound solution method for the crane scheduling problem, Transportation Research Part B: Methodological 24 (3) (1990) 159–172.

[20] D.-H. Lee, H. Q. Wang, L. Miao, Quay crane scheduling with non-interference constraints in port container terminals, Transportation Research Part E: Logistics and Transportation Review 44 (1) (2008) 124–135.

[21] R. Tavakkoli-Moghaddam, A. Makui, S. Salahi, M. Bazzazi, F. Taheri, An efficient algorithm for solving a new mathematical model for a quay crane scheduling problem in container ports, Computers & Industrial Engineering 56 (1) (2009) 241–248.

[22] P. Legato, R. Mazza, R. Trunfio, Simulation-based optimization for discharge/loading operations at a maritime container terminal, OR Spectrum 32 (2010) 543–567.

[23] M. Gen, R. Cheng, Genetic Algorithms and Manufacturing Systems Design, John Wiley & Sons, Inc., 1st edn., 1996.

[24] X. Han, Z. Lu, L. Xi, A proactive approach for simultaneous berth and quay crane scheduling problem with stochastic arrival and handling time, European Journal of Operational Research 207 (3) (2010) 1327–1340.