To appear in the *International Journal of Production Research*
Vol. 00, No. 00, 00 Month 20XX, 1–35

Scheduling Twin Robots in a Palletising Problem

Oliver Thomasson[a], Maria Battarra[a*], Güneş Erdoğan[a], Gilbert Laporte[b]

[a]*School of Management, University of Bath, Bath BA2 7AY, United Kingdom*
[b]*HEC Montréal, 3000 chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7*

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

This paper introduces the Twin Robot Palletising Problem (TRPP) in which two robots must be scheduled and routed to pick up and deliver products at specified locations along a rail. The robots are initially located at the opposite ends of the rail and must preserve a minimum safe distance from one another. The objective is to minimise the makespan, defined as the time required to complete all operations and for both robots to return to their starting positions. The paper presents a proof of NP-Hardness of the TRPP, as well as two mixed integer linear programming models. Local search operators are introduced, before an iterated local search and a genetic algorithm are developed, in which a linear-time scheduling algorithm and dynamic programming are utilised to evaluate the quality of solutions. Extensive computational results demonstrate the limits of the mathematical models, the effectiveness of the metaheuristics, and the savings obtained by using twin robots instead of a single one.

Keywords: scheduling; genetic algorithms; palletising

1. Introduction

This paper introduces the Twin Robot Palletising Problem (TRPP) in which two robots must be scheduled and routed to pick up and deliver products at specified locations along a rail. The robots are initially located at the opposite ends of the rail and must preserve a minimum safe distance from one another. The objective is to minimise the makespan, defined as the time required to complete all operations and for both robots to return to their starting positions. The TRPP arises in automated industrial systems, which have the advantages of increasing productivity and quality of the product or process. Additionally,

*Corresponding author. Email: m.battarra@bath.ac.uk human labour costs are significantly reduced, as are the health and safety risks of dangerous processes. One industrial area that utilises automation extensively is palletisation. This is the process of transferring products from some warehouse location onto pallets. With an estimated two billion pallets in use in the USA in 2015 (LeBlanc 2017), it is clear that palletisation is essential to the effective transportation of goods in the twenty-first century.

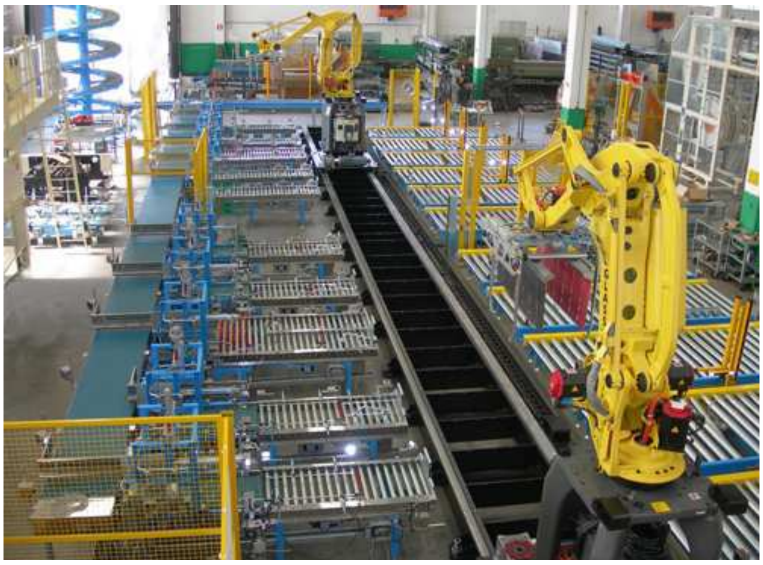
Figure 1. An industrial example of twin robots on a rail (AllGlass 2017).

To automate a palletising system like the one displayed in Figure 1 we need to design a scheduling algorithm to determine which robot will process each item, and in which order these items will be processed. The scheduling algorithms affect the system both operationally and strategically. Operationally, an optimal scheduling algorithm would enable the system to palletise the available items in the shortest possible time. Strategically, a good scheduling algorithm may inform manufacturers which palletising system would best suit their needs.

This paper introduces two mixed integer linear programming models of the TRPP, as well as two metaheuristics. The design of these methods is described, before their performances are evaluates with an extensive computational study. Additionally, we will compare the TRPP to a problem with a single robot.

2. Problem definition

In order to define the TRPP, we refer the reader to Figure 2 which demonstrates a potential industrial setup in which the TRPP arises. Here we have a set of workstations, each containing a unique type of product. Opposite the workstations are pallets, each requiring a set of products to complete a customer's order. These sets of products will therefore have to be relocated from the workstations to the pallets. Between the workstations and pallets we find a rail, on which a white robot and a black robot are situated. These robots are capable of transferring a single product from any workstation to any pallet. Additionally, at the extremities of the rail we see two depots, one for each robot. We require that the robots be located at their depots at the start and end of a palletising run. Our task is to schedule the robots such that the time taken to transfer all products from workstations to pallets (i.e., the *makespan*) is minimised. Additionally, we must ensure that the robots maintain a safety distance from each other at all times, to avoid collisions.

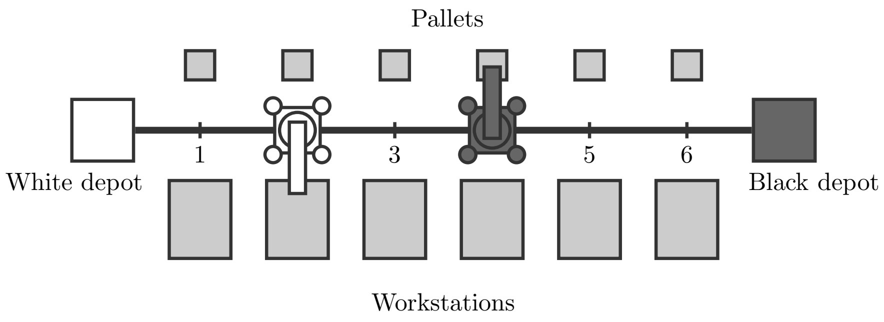

Figure 2. An industrial layout demonstrating the TRPP

We made some assumptions in order to tackle the TRPP. The pallets do not have to be filled in any order. We only wish to minimise the overall makespan, not the makespan of any individual pallet. The number of products available at any workstation is at least equal to the quantity of that product required by all orders. All orders are known a priori, so the set of jobs is static for the duration of the task. The positions of the pallets and workstations are fixed. The robots move with constant and equal speed, unless idle (acceleration and deceleration are not considered). If a robot starts a job, it will complete that job as soon as possible. There is no option to wait once a product is picked, and the robot is not permitted to deliver the product at any location other than its destination pallet. The time taken to pick up an item is constant, regardless of the item or robot, and is equal to the delivery time of the item.

2.1 *Literature review*

The Swapping Problem on a Line (Anily et al. 1999), and Scheduling Twin Robots on a Line (Erdoğan et al. 2013; Boysen et al. 2014) define the closest research problems to the TRPP in terms of definition and potential applications. The paper of Anily et al. (1999) introduces a problem, which the methods developed for can be used to optimally solve a single robot version of the TRPP in $\mathcal{O}(n^2)$ time. The papers of Erdoğan et al. (2013) and Boysen et al. (2014) develop methods to solve a problem like the TRPP, but with only the depots of the robots as entry points for jobs to the system.

Many other similar problems can be found in the port management literature. To identify the closest we refer to the classification scheme of Boysen et al. (2017), which categorises the TRPP as a [1D, 2, sm; mv^x, pos; C^{max}] type problem. The respective properties of the TRPP contained in this abbreviated classification are: its one-dimensional nature (1D), the two cranes or robots (2), the safety margins (sm), a constant travel speed (mv^x), specific initial and final positions of cranes/robots (pos), and a makespan objective (C^{max}). Many existing papers possess some of these properties. The single dimensional structure and the makespan objective alone are frequently found in the literature (Diabat and Theodoru 2014; Javanshir and Seyedalizadeh-Ganji 2010; Lee and Chen 2010; Lee and Wang 2010; Lee et al. 2008; Lim et al. 2007; Tang et al. 2014; Zhu and Lim 2006), along with papers containing problems with more similar features, such as those of Guan et al. (2013), Hakam et al. (2012), Liu et al. (2006), and Rodriguez-Molins et al. (2014). Each of these problems contains four of the same features as the TRPP, but again, differences are significant. The clearest difference is the fact that all referenced papers consider quay cranes, which differ from the robots in the TRPP as they are not required to collect their cargo from a certain workstation. Instead, trucks or yard cranes deliver cargo to the appropriate location for it to be loaded onto the ship. This removes the need to travel to a delivery point once a job is picked. If we were to use the TRPP to solve problems of this type, it would be equivalent to every job having its pickup location equal to its delivery location.

Scheduling problems of this type can often be found with intended applications outside of ports. Maschietto et al. (2017) study a crane scheduling problem in a steel coil distribution centre. While the applications of this problem are in the steel industry, the problem setup is similar to a yard crane arrangement from the port literature. As stated previously, the port setup does not produce a design we can use in the solving of the TRPP. Ge and Yih (1995) introduce a problem in the production of circuit boards, in which a single crane transfers circuit boards from an entry location to an available processing tank. Once the circuit board is processed the crane transfers the product to an exit location. Again, while this problem has similarities to the TRPP, the differences (invariant input/output locations, multiple transfers per job, intermediate drop-offs with waiting times) prohibit us from utilising the methods developed by Ge and Yih (1995). Yang et al. (2016) consider a multi-robot scheduling problem in which robots are tasked with transferring products to machines. The problem could be seen as a more open version of the circuit board production problem of Ge and Yih (1995), with multiple robots, the potential for multiple drop-offs on route to the exit location, and restrictions on which machines can process each job.

From a methodological viewpoint, while we could not find a problem with all of the same features as the TRPP in the literature, we did see the potential of using a genetic algorithm for our problem. Genetic, or evolutionary algorithms are very popular for problems in this subfield of operational research, and many examples can be found in the literature (Zhao et al. 2016; Wang et al. 2016; Pratap et al. 2016). Genetic algorithms are also popular in research on Vehicle Routing Problems (VRPs). One of particular interest to our work on the TRPP is the paper of Vidal et al. (2012). In developing a genetic algorithm for the TRPP, we have used some components of the Vidal et al. hybrid genetic algorithm for multi-depot and period VRP's; the details of which are provided in Section 4.5.

2.2 *Problem properties*

Before proving that the TRPP is $\mathcal{NP}$-hard, we demonstrate our method for visual representation of solutions. We use time-space Gantt charts to achieve this, an example of which can be seen in Figure 3. In this figure the y-axis represents the position of the robots on the rail, and the x-axis represents time. We see jobs represented as polygons, each showing the time in which a robot is picking up an item from a workstation, moving to the correct delivery location, and placing the item on the pallet, as well as the location of the robot on the rail during this time. We see the process time μ for pickup or delivery labelled on job 1, and repeated without labelling on all other jobs. When reading this Gantt chart, we can take any time instance (e.g., the time ζ) and use the chart to find the position of each robot at this time, as well as the job (if any) currently being processed by the robots. If the robot is not processing a job, we represent its position with a dashed line. At time ζ, we observe that the black robot is in the process of delivering job 5, while the white robot is moving from the delivery point of job 2 to the pickup point of job 3.

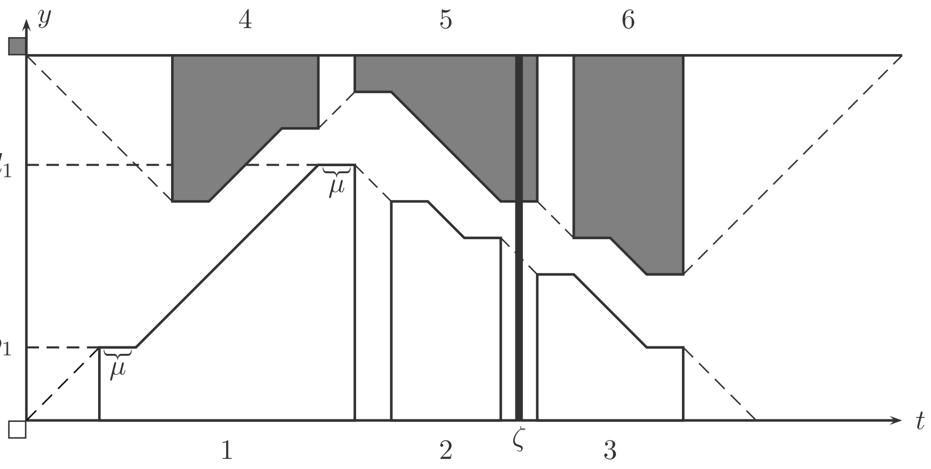

Figure 3. An example of a TRPP solution, represented as a Gantt Chart

Theorem 2.1. *The TRPP is $\mathcal{NP}$-Hard*

Proof. The proof proceeds by reduction to a general case of the TRSP introduced by Erdoğan et al. (2013) and proven to be $\mathcal{NP}$-hard. Table 1 compares the inputs of the TRSP to those of the TRPP. Chevrons show which inputs are required for the problems, and a tilde is used to show when only a partial set of inputs is required. While the TRSP does not have a set of workstations as in the TRPP we consider the depots as workstations Ψ_0 and Ψ_L.

Parameter	Symbol	TRSP	TRPP
Set of workstations	(Ψ_i)	∼	✓
Set of pallets	(Φ_i)	✓	✓
Length of the rail	(L)	✓	✓
Number of jobs	(n)	✓	✓
Set of jobs assigned to the white robot	(W)	✓	
Set of jobs assigned to the black robot	(B)	✓	
Pickup location of each job	(p_i)		✓
Delivery location of each job	(d_i)	✓	✓
Safety distance	(σ)	✓	✓
Travel time	(τ)	✓	✓
Handling time	(μ)		✓

Table 1. Inputs of the TRSP and TRPP.

To reduce the TRPP to the TRSP we set L, n, d_i, σ and τ identically to the TRSP. We then set $\mu = 0$, as the TRSP considers process time to be negligible. Finally, we transform the sets W and B such that $p_i = 0$ for $i \in \{1, ..., m\}$ and $p_i = L$ for $i \in \{m+1, ..., n\}$. This ensures all jobs to be processed by the white robot are at Ψ_0 and all jobs for the black robot are at Ψ_L.

The constraint of minimum safe distance, combined with out limited workstation set, ensures that any solution produced with this TRPP setup is a feasible solution for the TRSP. Since the solutions for both problems are presented as a set of start times (one for each job) no transformation is required to convert the outputs of our reduced TRPP to the corresponding TRSP solution.

□

2.2.1 Scheduling coefficients

To schedule jobs appropriately we require knowledge of the minimum intervals that must be respected between start times of each pair of jobs. For this reason we developed an algorithm to run prior to any other method. This algorithm takes every possible pair of jobs on opposite robots, and determines the minimum required time intervals. A visual representation of one of these pairs can be seen in Figure 4. Job j is assumed to have been previously scheduled, and is in a fixed position, with start time S_j. We then wish to schedule job i on the opposite robot, which could be scheduled before or after j. Job $i-$ shows the position of job i if scheduled as late as possible before j, and $i+$ shows the position of job i if scheduled as early as possible after j. The calculation of S_{i-} and S_{i+} then allows us to obtain the coefficients $\alpha^1_{i,j} = S_j - S_{i-}$ and $\gamma^1_{i,j} = S_{i+} - S_j$. If job j is assigned to the white robot we would calculate coefficients $\alpha^2_{i,j}$ and $\gamma^2_{i,j}$ instead. However, due to the design of the matrices, $\alpha^2_{i,j} \equiv \gamma^1_{j,i}$ and $\gamma^2_{i,j} \equiv \alpha^1_{j,i}$. Detailed pseudocodes for the calculation of these values can be found in Appendix A. The computation of the α and γ values is completed in constant time for each job pair, therefore the process of fully populating the coefficient matrices takes $\mathcal{O}(n^2)$ time.

3. Mathematical Models

In this section we present a mixed integer linear programming formulation for the TRPP. This formulation utilises a graph representation of the TRPP. In addition, we have developed a second formulation based on time-indexed variables. For the sake of brevity, the second formulation is presented in Appendix C. Both formulations are based on the assumptions that $\mu = 1$, $\tau = 1$, and $\sigma = 1$, since changes to these values do not affect the complexity of the problem.

3.1 *Graph-representation based formulation*

Let $G = (V, A)$ be a directed graph where $V = \{0, 1, ..., n + 1\}$; with vertices 0 and $n+1$ corresponding to the depots of the white robot and

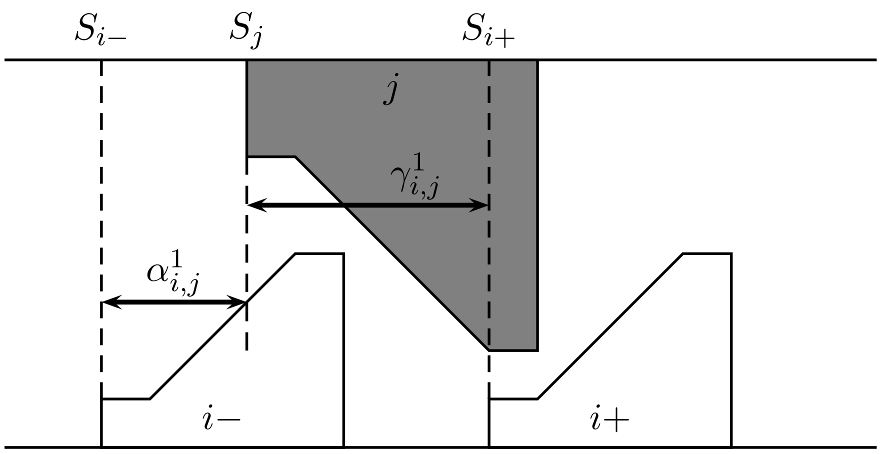

Figure 4. $\alpha^1_{i,j}$ and $\gamma^1_{i,j}$ coefficients

the black robot, respectively. Then we define $V_T = V \setminus \{0, n+1\}$, the set of jobs to be completed. The set A contains the arcs, connecting tasks. The length of an arc (i,j) is equal to θ_{ij}, the parameter defined as the necessary time from the end of job i to the end of job j (for $i, j \in \{0, n+1\}$; $\mu = 0$). We then have parameters η_{ij} the necessary time from the start of job i to the start of job j (for $i, j \in \{0, n+1\}$ there is no pickup or delivery operation); α^1_{ij}, α^2_{ij}, γ^1_{ij}, and γ^2_{ij}, defined as in Section 2.2.1; and T, as in the previous formulation.

Finally, we have the decision variables: x_{ij} which is 1 if the white robot executes task j immediately after task i and 0 otherwise; y_{ij} which is 1 if the black robot executes task j immediately after task i and 0 otherwise; s_i, the time at which the pickup operation of job i begins; and z_{ij} which is 1 if job i is scheduled immediately before job j and 0 otherwise. The formulation is as follows:

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

$$\text{subject to } \sum_{j \in V_T} x_{0j} \leq 1 \tag{2}$$

$$\sum_{j \in V_T} x_{j0} = \sum_{j \in V_T} x_{0j} \tag{3}$$

$$\sum_{j \in V_T} y_{n+1,j} \leq 1 \tag{4}$$

$$\sum_{j \in V_T} y_{j,n+1} = \sum_{j \in V_T} y_{n+1,j} \tag{5}$$

$$\sum_{j \in V} x_{ij} + y_{ij} = 1 \qquad i \in V_T \tag{6}$$

$$\sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} \qquad j \in V_T \tag{7}$$

$$\sum_{i \in V} y_{ij} = \sum_{i \in V} y_{ij} \qquad j \in V_T \tag{8}$$

$$w \geq \sum_{(i,j) \in A} \theta_{ij} x_{ij} \tag{9}$$

$$w \geq \sum_{(i,j) \in A} \theta_{ij} y_{ij} \tag{10}$$

$$s_j \geq s_i + \eta_{ij} - T(1 - x_{ij} - y_{ij}) \qquad i, j \in V_T \tag{11}$$

$$s_i \geq \eta_{0i} x_{0i} + \eta_{L+1,i} y_{L+1,i} \qquad i \in V_T \tag{12}$$

$$s_j \leq s_i - \alpha^1_{ij} + T(1 - z_{ij} + \sum_k y_{ki} + \sum_k x_{kj}) \qquad i, j \in V_T; k \in V \tag{13}$$

$$s_j \geq s_i + \gamma^1_{ij} - T(z_{ij} + \sum_k y_{ki} + \sum_k x_{kj}) \qquad i, j \in V_T; k \in V \tag{14}$$

$$w \geq s_i + \eta_{i0} x_{i0} + \eta_{i,L+1} y_{i,L+1} \qquad i \in V_T \tag{15}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{16}$$

$$y_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{17}$$

$$z_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{18}$$

$$s_i \geq 0 \qquad i \in V_T \tag{19}$$

$$w \geq 0. \tag{20}$$

Constraint (2) ensures the white robot leaves the depot at most once. Constraint (3) states that the number of jobs ending at the white depot is equal to the number of jobs originating there (at most one). Constraints (4) and (5) state the same conditions for the black robot. Constraints (6) state that jobs must be completed exactly once, by just one of the robots, and Constraints (7) and (8) ensure that all jobs are succeeded by another if performed by the white and black robot respectively. Constraints (9) and (10) limit the makespan to be at least the minimum time taken for either robot to complete its assigned jobs, and Constraints (11) restrict the start time of a job to be at least the end time of the previous job, plus the time taken to travel between the jobs. Constraints (12) state that the start time of a job is at least the time taken to travel from the robot's depot, to the origin of the job. Constraints (13) and (14) maintain the minimum offset which must be respected if we have a scheduled job and wish to schedule further jobs on the opposite robot. Constraints (15) state that the makespan is at least the end time of the final job of either robot, plus the time taken to return to the respective depot. Constraints (16) to (20) define the domains of the variables.

4. Metaheuristic algorithms

Since the formulations are unable to solve instances of the problem with 15 or more jobs within two CPU hours (details presented in Section 5), we have developed heuristics to find solutions for larger instances. We present three basic methods, and two metaheuristic algorithms, each building on the results of the previous one.

4.1 *Scheduling algorithm*

The following methods consider a solution representation of the TRPP as two arrays. The first array, Seq, lists the jobs in order of starting time. The second is a corresponding sequence of robot assignments, Asn. We have developed a linear-time algorithm which takes these two sequences, produces a feasible schedule of starting times, and calculates the makespan. This algorithm uses the precalculated scheduling parameter matrices α^1, α^2, γ^1, and γ^2, which are as defined earlier (Section 2.2.1). The algorithm selects the jobs in Seq one at a time, and schedules each one as early as possible to maintain the order of jobs and feasibility. A pseudocode for this algorithm is provided in Appendix B. Initial solutions are produced by randomly assigning a robot and an order to each job, then applying this scheduling algorithm.

4.2 *Basic Local Search*

To improve the results obtained from the scheduling algorithm we introduce seven local search operators to be performed on a solution. Here we state the seven operators, and their computational complexity:

(1) **Assignment change:** Select one job and change its assignment ($\mathcal{O}(n)$).
(2) **Assignment swap:** Select a pair of jobs with different assignments, and swap their assignments ($\mathcal{O}(n^2)$).
(3) **Order change:** Select one job and change its position in the sequence of orders, maintaining the assignment of all jobs ($\mathcal{O}(n^2)$).
(4) **Order swap:** Select a pair of jobs and swap their positions in the sequence of orders, maintaining the assignment of all jobs ($\mathcal{O}(n^2)$).
(5) **Order & Assignment Change:** Select one job and change its position in the sequence of orders, as well as its assignment ($\mathcal{O}(n^3)$).
(6) **Reverse job set:** Take a set of size 4 or more of consecutive jobs and reverse their positions in the order sequence, maintaining the assignment of all jobs ($\mathcal{O}(n^2)$).
(7) **Move job set:** Take a set of size 2 or more of consecutive jobs and change their position in the sequence of orders, maintaining the assignment of all jobs ($\mathcal{O}(n^4)$).

4.3 *Dynamic Program*

Due to the nature of our scheduling algorithm, there are some cases in which we could never find the optimal solution with the methods we have described so far. This is a consequence of the fact that none of our algorithms consider the possibility of delaying a job to improve the overall makespan. We know that this strategy will be necessary to find optimal solutions in some cases, and an example is depicted in Figure 5. In both charts, the order of jobs in Seq is the same when sorted by start time. Job 1 starts first, job 2 follows, and job 3 starts last. Chart A is the sequence that would be given by the scheduling algorithm. Since the jobs are scheduled based on an 'as early as possible' basis, job 2 forces the white robot to move back towards its depot, and wait, before processing job 3. A delay of one unit to the start time of job 2 allows job 3 to fit into the schedule earlier, reducing the time taken to process the three jobs from 16 units in Chart A to 11 units in Chart B. Potential delays can be identified using the α coefficients described in Section 3. To identify the potential for delay in Figure 5, we would check the value of $\alpha^1_{3,2}$ (i.e., the minimum time gap that must be respected between the start time of job 3, and the start of job 2, if job 3 is to be scheduled before job 2, and remain feasible.). If $\alpha^1_{3,2} \leq 0$ then there may be a benefit to delaying job 2.

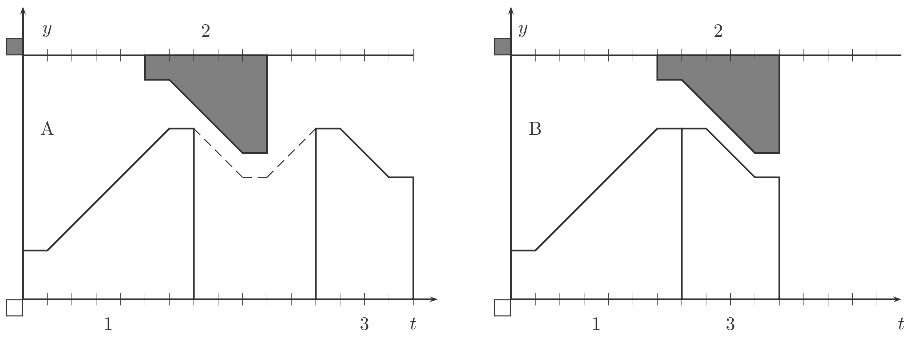

Figure 5. Example of a delay which benefits the makespan

To deal with the delay problem we developed a dynamic program which can work alongside any of our heuristic methods. This addition means that it is then always possible to find an optimal solution. A pseudocode for our dynamic program is given in Algorithm 1. Prior to the start of this process, all pairs of jobs have their respective α coefficients checked. As stated earlier, if $\alpha^1_{i,j} \leq 0$ or $\alpha^2_{i,j} \leq 0$, depending on the assignment of jobs, a delay to job j may be beneficial. This a priori coefficient check enables the **if** statement on lines 9 and 15 of Algorithm 1.

The dynamic program utilises a recursive function $RF(\kappa_w, \kappa_b, pos, \lambda)$, where κ_w and κ_b are the end times of the

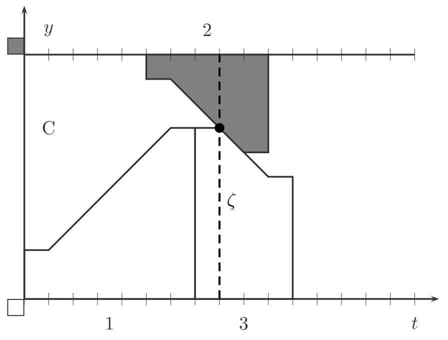

Figure 6. The delay calculation step of the dynamic program

most recent jobs completed by the white robot and black robot, respectively. Our current position in the sequence of jobs is denoted as *pos*. Our control array is λ, and our value array is Ω. In standard dynamic program implementation we would use Ω with indices of *pos*, κ_w, and κ_b to match states. However, the optimal solution value can easily be read as $\max\{\kappa_w, \kappa_b\}$. Hence we have used Ω with only two indices, *pos* and $\max\{\kappa_w, \kappa_b\}$, for algorithmic efficiency, where $\max\{\kappa_w, \kappa_b\}$ is the current makespan. Within this construct we initialise all values stored in Ω to M, and use Ω to store the value of the makespan of the robot which completes its assigned jobs earliest. Any state that has a worse solution that the one stored in Ω can then be discarded, since it is dominated.

The scheduling within the recursion is controlled by $\chi(robot, pos)$, which calculates the difference between the end time of the current job, and that of the previous job completed by the same robot. Delays are designated by the $D(Seq, pos)$ function, which calculates the necessary delay of the current job to allow a future job to be processed earlier. This calculation is carried out using the step displayed in Figure 6, we also consider the charts of Figure 5. Chart A shows the first step that occurs in function D, where job 3 is scheduled as normal. Upon calculating that the pairing of jobs 2 and 3 provides the potential for a delay, the D function schedules job 3 at the earliest possible time, ignoring all jobs completed by the black robot (Chart C). The location of both robots at time ζ, the end of the pickup operation of job 3, is then compared. The necessary delay to job 2 is then applied to obtain feasibility (Chart B).

4.4 *Iterated Local Search*

While the results from the local search operators show improvement from the scheduling algorithm alone, they still fail to reliably find the known optimal solutions for even instances with just five jobs. Our

Algorithm 1 Dynamic Program

```
1: **Set:** $\kappa_w = 0, \kappa_b = 0, pos = 0, \Omega(i, j) = M(\forall i \leq n, \forall j \leq T)$
2: **Function:** $RF(\kappa_w, \kappa_b, pos, \lambda)$
3: **if** $pos = n$ **then**
4:    **return** $\max\{\kappa_w, \kappa_b\}$
5: **if** $\Omega(pos, \max\{\kappa_w, \kappa_b\}) = M$
   or $(\kappa_w > \kappa_b$ and $\kappa_b < \Omega(pos, \kappa_w))$
   or $(\kappa_b > \kappa_w$ and $\kappa_w < \Omega(pos, \kappa_b))$ **then**
6:    $\Omega(pos, \max\{\kappa_w, \kappa_b\}) = \kappa_b$
7:    **if** $Asn(pos) = white$ **then**
8:       $K1 = RF(\kappa_w + \chi(white, pos), \kappa_b, pos + 1, \lambda)$
9:       **if** $\alpha^1_{Seq(pos),Seq(pos-1)} \leq 0$ **then**
10:         $K2 = RF(\kappa_w + \chi(white, pos) + D(Seq, pos), \kappa_b, pos + 1, \lambda)$
11:      **else**
12:         $K2 = M$
13:   **else**
14:      $K1 = RF(\kappa_w, \kappa_b + \chi(black, pos), pos + 1, \lambda)$
15:      **if** $\alpha^2_{Seq(pos),Seq(pos-1)} \leq 0$ **then**
16:         $K2 = RF(\kappa_w, \kappa_b + \chi(black, pos) + D(Seq, pos), pos + 1, \lambda)$
17:      **else**
18:         $K2 = M$
19:   **if** $K2 < K1$ **then**
20:      $\lambda(\kappa_w, \kappa_b, pos) = 1$
21:      **Return** $K2$
22:   **else**
23:      $\lambda(\kappa_w, \kappa_b, pos) = 0$
24:      **Return** $K1$
25: **else**
26:   **if** $\Omega(pos, \max\{\kappa_w, \kappa_b\}) < M$
      and $((\kappa_w > \kappa_b$ and $\kappa_b = \Omega(pos, \kappa_w))$
      or $(\kappa_b > \kappa_w$ and $\kappa_w < \Omega(pos, \kappa_b)))$ **then**
27:      **return** $\Omega(pos, \max\{\kappa_w, \kappa_b\})$
28:   **else**
29:      **return** $M$
```

next attempt at an improved heuristic was an iterated local search (ILS) as described by Lourenço et al. (2003), the framework of which is shown in Algorithm 2.

Here, s_0 represents the solution of the scheduling algorithm, on which a local search is then carried out, before performing a perturbation. A pseudocode for the *Perturb* function is shown in Algorithm 3, where $U[i, j]$ is the discrete uniform distribution from i to j. Here, l and m are parameters which allow us to change the size of perturbation available. This perturbation potentially allows an escape from the cur-

Algorithm 2 Iterated Local Search

```
1: $s_0 = GenerateInitialSolution$
2: $s^* = LocalSearch(s_0)$
3: while Termination condition not met do
4:     $s' = Perturb(s^*)$
5:     $s^{*\prime} = LocalSearch(s')$
6:     $s^* = AcceptanceCriterion(s^*, s^{*\prime})$
```

rent local minimum, in an attempt to find better solutions elsewhere in the search space. The ILS iteratively takes the perturbed solution, performs a local search, and tests if satisfies the acceptance criterion. In our implementation, if the current solution has a better makespan than the best solution found so far, it is accepted. This process is repeated until the termination condition is met.

Algorithm 3 Perturbation Function

```
1: Set $k_{max} = U[l, m]$
2: Set $k = 0$
3: while $k < k_{max}$ do
4:     Set $m = U[1, 7]$
5:     Perform Random move in Local Search operator $m$
6:     $k = k + 1$
```

4.5 *A Hybrid Genetic Algorithm*

We now introduce the final heuristic we implemented, a hybrid genetic algorithm. A pseudocode is shown in Algorithm 4. S is the set of solutions $\rho_{i,j}$ forming the population. Each population member has two indices as we have designed our population as a matrix. The structure of this matrix is shown in Figure 7.

$$\begin{bmatrix} \rho_{0,0} & \rho_{1,0} & \rho_{2,0} & \cdots & \rho_{R,0} \\ \rho_{0,1} & \rho_{1,1} & \rho_{2,1} & \cdots & \rho_{R,1} \\ \rho_{0,2} & \rho_{1,2} & \rho_{2,2} & \cdots & \rho_{R,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{0,I} & \rho_{1,I} & \rho_{2,I} & \cdots & \rho_{R,I} \end{bmatrix}$$

Figure 7. The population matrix

In this algorithm R is the number of perturbations to perform on the initial solution (line 3 of Algorithm 4), and I is the number of improvements to perform on each perturbed solution (line 5 of Algorithm 4). The function *Perturb* is as defined in Section 4.4. The function *Improve* is used to generate solutions of better quality than the random solutions, by performing local search moves until an improved solution is found. If no improving solution is found, the function performs a number of random moves to generate a non-improving solution, and continues the search.

Algorithm 4 Hybrid Genetic Algorithm

```
1: Generate Initial Solution $\rho_{0,0}$
2: for $i \in \{1, ..., R\}$ do
3:    $\rho_{i,0} = Perturb(\rho_{i-1,0})$
4:    for $j \in \{1, ..., I\}$ do
5:       $\rho_{i,j} = Improve(\rho_{i,j-1})$
6: while Stopping criteria not satisfied do
7:    $DiversityCalculation(S)$
8:    for $i \in \{1, ..., |Q|\}$ do
9:       Select: $P_1$, $P_2$
10:      $q_i = Crossover(P_1, P_2)$
11:      $Mutation(q_i)$
12:      $LocalSearch(q_i)$
13:   $DiversityCalculation(Q)$
14:   $Replacement(S, Q)$
15: Return: $Best(S)$
```

After the initial population generation, the algorithm enters a loop until termination criteria are met. Much like the ILS, there are multiple options for these criteria. We have tested a variety of computing time limits and iteration limits. Within the loop, multiple operations are performed. The *DiversityCalculation* function is called for the individuals in the population S. This function assigns a value of fitness to each population member quantifying how diverse the solution is when compared to all other population members. We perform this calculation to enable us to select the more diverse solutions as parents in the next step. To calculate diversity we use the evaluation methods described by Vidal et al. (2012). For a solution in the population $\rho_{i,j}$, or an offspring q_i, we calculate a *diversity contribution* $\Delta\rho$, and a *biased fitness* $BF(\rho)$ where

$$\Delta(\rho) = \frac{1}{2|S|^2} \sum_{\rho_2=1}^{|S|} \sum_{i=1}^{|S|} \left(1(\pi_i(\rho) \neq \pi_i(\rho_2)) + 1(\xi_i(\rho) \neq \xi_i(\rho_2))\right) \tag{21}$$

$$BF_1(\rho) = fit(\rho) + \left(1 - \frac{nbElit}{|S|}\right) dc(\rho) \tag{22}$$

$$BF_2(\rho) = fit(\rho) + (DW)dc(\rho). \tag{23}$$

In (21), $|S|$ is the number of solutions in the population. π_i and ξ_i contain the characteristics of the solution, commonly referred to as *chromosomes*. Note that, since Vidal et al. developed this hybrid genetic algorithm to tackle a multi-depot and periodic VRP, the characteristics described by the chromosomes will be different for our utilisation of this equation for the TRPP. For our problem, $\pi_i(\rho)$ represents the index of the job which is at position i in the job order sequence of solution ρ, and $\xi_i(\rho)$ represents the robot assignment of the same job in the same solution.

The calculation of $BF_1(\rho)$ (22) defined by Vidal et al. (2012) is preceded by a ranking of the solutions in the population by two criteria. First, solutions are ranked by makespan; this rank is the value $fit(\rho)$. The solutions are then ranked by their diversity contribution $\Delta(\rho)$ and this ranking is the value $dc(\rho)$. The value *nbElit* is the number of elite solutions in the population. Elite solutions are those with the best $BF_1(\rho)$ values. We therefore wish to keep them in the population until the next generation, regardless of whether we find enough offspring of better quality to fill the population.

Equation (23) is an alternative to the biased fitness calculation of Vidal et al. Here the variable DW is introduced and represents the *diversity weighting* of the calculation. By changing the value of DW we can alter the influence of the diversity ranking on the biased fitness of each solution. Higher values of DW imply a greater importance is placed on the diversity of a solution, over its makespan. Our hybrid genetic algorithm can be performed using BF_1 or BF_2.

Once diversity measures are assigned to all population members, parents are selected for each new solution to be created. Q is the set of solutions produced as offspring. This selection procedure, like the diversity measures, is based on the Vidal et al. (2012) hybrid genetic algorithm. The procedure is outlined in Algorithm 5. This selection is performed twice to obtain two parents. If the two parents are the same, the parent selection algorithm is repeated for P_2 until different parents are obtained.

Once parents are selected, a crossover is performed to produce an offspring. We used two procedures for this crossover: one-point crossover, and two-point crossover. These procedures are based on similar crossover operators for the Travelling Salesman Problem (Larraønaga et al. 1999), and are outlined in Algorithm 6. The procedures perform on both *Seq* and *Asn* of the selected parents.

Algorithm 5 Parent Selection

```
1: $h = \lfloor U[0, ..., R+1) \rfloor$          {column of candidate 1 in population}
2: $i = \lfloor U[0, ..., I+1) \rfloor$          {row of candidate 1 in population}
3: $j = \lfloor U[0, ..., R+1) \rfloor$          {column of candidate 2 in population}
4: $k = \lfloor U[0, ..., I+1) \rfloor$          {row of candidate 2 in population}
5: while $h = i$ and $j = k$ do
6:     $j = \lfloor U[0, ..., R+1) \rfloor$
7:     $k = \lfloor U[0, ..., I+1) \rfloor$
8: if $BF(\rho_{h,i}) > BF(\rho_{j,k})$ then
9:     $P = \rho_{h,j}$
10: else
11:     $P = \rho_{j,k}$
```

Algorithm 6 Crossover Procedures

```
1: One-Point Crossover:
2: $c = \lfloor U[1, ..., n) \rfloor$
3: for $i < c$ do
4:     $q(i) = P_1(i)$
5: for $i \geq c$ and $i < 2n$ do
6:     $inheritedJob = 0$
7:     for $k < n$ do
8:         if $q(k) = P_2(i \mod n)$ then
9:             $inheritedJob = 1$
10:    if $inheritedJob = 0$ then
11:        $q(i) = P_2(i \mod n)$
```

```
1: Two-Point Crossover:
2: $c_1 = \lfloor U[1, ..., n) \rfloor$
3: $c_2 = \lfloor U[1, ..., n) \rfloor$
4: while $c_1 = c_2$ do
5:     $c_2 = \lfloor U[1, ..., n) \rfloor$
6: if $c_1 > c_2$ then
7:     $Swap(c_1, c_2)$
8: for $i < c_1$ do
9:     $q(i) = P_1(i)$
10: for $i \geq c_1$ and $i < 2n$ do
11:     $inheritedJob = 0$
12:     for $k < n$ do
13:         if $q(k) = P_2(i \mod n)$ then
14:             $inheritedJob = 1$
15:     if $inheritedJob = 0$ then
16:         $q(i) = P_2(i \mod n)$
17:     if $i = c_2$ then
18:         Break
19: for $i \geq c_2$ and $i < 2n$ do
20:     $inheritedJob = 0$
21:     for $k < n$ do
22:         if $q(k) = P_1(i \mod n)$ then
23:             $inheritedJob = 1$
24:     if $inheritedJob = 0$ then
25:         $q(i) = P_1(i \mod n)$
```

Once the crossover procedure has been completed, successfully producing an offspring, a mutation operation may be performed on the offspring. *Mutation* is similar to the *Perturb* operation seen in Algorithm 4, but will not be applied to every offspring. Mutation occurs with a certain probability, and consists of a small number of random moves within a randomly selected local search operator. After the mutation, a local search is performed on the offspring to improve the solution. At the local search stage we may also wish to run the dynamic program described in Section 4.3. These steps are repeated to produce the required set of offspring, before calculating each offspring's diversity measures against the initial population.

Next, offspring are chosen to replace members of the population, using their biased fitness measures. First, the elite solutions discussed earlier are fixed in the population. The non-elite population members are then removed from the population, and pooled with the offspring. From this pool, the solution with the best biased fitness measure is selected and transferred from the pool of potential population members, into the population. This process is repeated until the population is full. The remaining solutions outside of the population are then discarded. The process of producing offspring and updating the population is then repeated until a termination criterion is met, at which point the best solution in the population is identified.

5. Computational study

Here we provide the results of our computational comparison of all methods. Each heuristic algorithm has been tested on 1,200 instances of the TRPP. These tests were grouped by number of jobs (n), and length of rail (L). For every tested combination of number of jobs and length of rail, 100 instances are tested. Tests are run with 5, 10, 20, 30, 50, and 100 jobs on rails of length 5 and 10. In all instances, we set $\mu = 1$, $\tau = 1$, and $\sigma = 1$ since changes to these values do not affect the complexity of the problem. Tests were run on Balena High Performance Computer (HPC) at the University of Bath, containing Intel Ivy Bridge 2.60GHz cores with 64GB RAM[1]. It should be assumed that tests were conducted for all 1,200 instances, unless stated otherwise. Full results for all tests are available upon request. To ensure our computational tests were thorough, we referred to the paper of Hall and Posner (2001) which describes appropriate strategies for instance generation in scheduling problems. Our input data is relatively simple, giving us only a set of jobs with pickup and delivery locations defined. We have sampled pickup and delivery locations from a discrete uniform distribution $U_d[1, L]$ to produce instances.

[1] Full technical specifications can be found at www.bath.ac.uk/bucs/services/hpc/facilities/

5.1 *Results of testing the formulations*

When initially tuning the performance of our formulations, we tested the models on a pilot set of instances, with a small number of jobs. These tests were run using CPLEX 12.6.1, and were limited to two hours of CPU time. Table 2 summarises the results of this initial testing, and full results can be seen in Appendix D. The given values are: the percentage of optimal solutions found (% Opt); the average computation time for those instances which both formulations could complete; and the average percentage gap between the upper and lower bound, for those instances that both formulations failed to complete.

	Graph representation	Time-indexed
% Opt	84	89
Average computation time (s)	312.58	15.68
Average % gap	33.77	187.13

Table 2. Summary of results of initial formulation testing

Both formulations can solve any instance with 10 or fewer jobs. The time-indexed formulation is also able to solve instances with 12 jobs. For the instances with 15 or more jobs, both formulations are unable to find the optimal solutions within the time limit. For the smallest instances the graph-based formulation finds the optimal solution the fastest, but this quality transfers to the time-indexed formulation as the instance size increases.

5.2 *Results of basic methods testing*

The results of running the scheduling algorithm (ScA), basic local search (LS), and dynamic program (DP) on all 1,200 instances are summarised in Table 3. Since the formulation could not solve any instances with 20 or more jobs in the time given, not all scheduling algorithm results can be compared to an optimal solution. For this reason, most results are compared to the best known solution from later methods, to give an average percentage deviation from the best known solution (% Dev.). We ran the dynamic program in two ways; DP1 is the case in which we ran it at the end of the full set of local search operators, and DP2 is the case in which we ran the dynamic program after every local search move. All computation times are negligible (less than 0.005 seconds) in these initial tests of the basic methods, but the difference in computation time between DP1 and DP2 will become more prevalent later within more complex algorithms.

The addition of the local search moves significantly reduces the deviation from the best known solutions, as would be expected. DP1 produces a small reduction of the the deviation on some instance sizes, while DP2 has a larger effect on every instance type. The improvements

Instance		ScA		ScA+DP		ScA+LS		ScA+LS+DP1		ScA+LS+DP2	
n	L	% Dev	# Opt	% Dev	# Opt	% Dev	# Opt	% Dev	# Opt	% Dev	# Opt
5	5	57.07	2	55.55	2	4.23	54	4.23	54	3.62	62
5	10	70.00	0	68.19	0	6.09	39	6.05	39	5.44	43
10	5	84.74	0	83.97	0	10.74	6	10.74	6	10.28	7
10	10	96.98	0	94.17	0	15.07	4	15.04	4	13.98	4
20	5	107.17		104.96		14.62		14.57		14.24	
20	10	111.63		106.92		18.69		18.65		18.24	
30	5	111.73		110.01		14.72		14.72		14.40	
30	10	124.78		119.61		19.11		19.11		18.78	
50	5	120.54		118.16		13.99		13.99		13.78	
50	10	129.30		126.22		16.97		16.97		16.64	
100	5	122.65		120.19		12.02		12.02		11.87	
100	10	129.11		124.73		11.85		11.85		11.68	
Average:		105.48		102.72		13.18		13.16		12.75	

Table 3. Summary of results of basic method testing

made can be further seen by observing the number of optimal solutions found by each method in Table 3. We see that test five clearly finds the largest number of optimal solutions. However, the basic methods alone clearly cannot be used to find a high number of optimal solutions for even the smallest instance sizes.

5.3 *Results of Iterated Local Search testing*

Table 4 summarises the best results of our ILS testing. As before, the dynamic program can be used in multiple ways. DP0 shows that the dynamic program was not utilised for the corresponding tests; DP1 and DP2 are as described in the previous section.

Instance type		DP0	DP1	DP2
n	L	% Dev.		
5	5	3.12	3.12	2.33
5	10	4.77	4.77	3.97
10	5	8.30	8.30	7.49
10	10	10.39	10.39	8.72
20	5	11.70	11.77	10.96
20	10	14.62	14.64	13.11
30	5	12.19	12.19	11.40
30	10	14.76	14.74	12.89
50	5	11.38	11.38	10.66
50	10	12.87	12.87	11.28
100	5	10.12	10.12	9.42
100	10	8.56	8.56	7.65
	Average	10.23	10.24	9.16

Table 4. Results of the best performing ILS tests

The results shown in Table 4 come from tests with a time limit termination condition. We also ran tests with iteration limits, which gave identical results for the smallest instances, worse quality results for medium size instances, and took too long to complete the largest instances to be viable. Further results of preliminary testing showed that scaling time limits with increasing instances size yielded the best results. We also tested with longer time limits, but no additional gains were made from the displayed results. The time limits, in seconds, for the tests carried out for Table 4, were six times the number of jobs on any instance.

5.4 *Results of Hybrid Genetic Algorithm testing*

When studying the hybrid genetic algorithm, there were seven parameters which had to be tuned; namely: the size of the population, the number of offspring, the number of elite solutions, the number of crossovers, the termination criteria, the use of the dynamic program, the weight of diversity in the biased fitness equation, and the probability of mutation. We decided on a baseline population of size 25, and used the calibration results of Vidal et al. (2012) on his hybrid genetic algorithm. This meant we started by considering 40 offspring, five elite solutions, and the diversity weighting given in Equation (22), which we refer to as BF_1. We set our termination criteria the same as that which gave the best results for the ILS testing; a time limit (in seconds) of six times the number of jobs. We also set the probability of mutation to zero, did not use the dynamic program, and utilised one-point crossovers.

Initially our testing had a focus of changing one parameter at a time in an attempt to identify which changes had the most influence. Those results are shown in Table 5 as tests one to 13. The table shows the parameters we set, then the average number of iterations completed (# Itn.), and the average percentage deviation from the best known solution (% Dev.).

The change in diversity weighting had the best impact on the tests. We therefore set this parameter accordingly for the second round of testing. We then continued the above process, adjusting one parameter at a time to see the impact and accepting adjustment of those which performed best. For subsequent rounds of testing we discarded the dynamic program as an option, because of its poor performance. We believe this occurs due to the high complexity discussed earlier. This issue can be best seen in the testing of DP2, where the average number of iterations which could be completed within the allotted time is significantly lower than in any other test. The results of our further testing can be seen in Table 5 as tests 14 to 20.

One test that deviates from the approach described earlier is test 14. This test simply took all of the best single parameter improvements from our first round of testing, and combined them. Clearly this approach yields poor results, worse than the baseline; so we continued

Test number	Pop size	Offspring	Elite solutions	Crossovers	Termination	Dynamic Program	Diversity weight	Mutation probability	# Itn.	% Dev.
1	25	40	5	1-point	6n	Off	BF_1	0%	13675	2.12
2	36	40	5	1-point	6n	Off	BF_1	0%	12402	1.44
3	49	40	5	1-point	6n	Off	BF_1	0%	10918	1.21
4	64	40	5	1-point	6n	Off	BF_1	0%	9337	1.09
5	81	40	5	1-point	6n	Off	BF_1	0%	7643	1.15
6	100	40	5	1-point	6n	Off	BF_1	0%	6152	1.78
7	25	40	5	2-point	6n	Off	BF_1	0%	13426	1.94
8	25	40	5	mixed	6n	Off	BF_1	0%	13461	1.95
9	25	40	5	1-point	12n	Off	BF_1	0%	27395	2.01
10	25	40	5	1-point	6n	DP1	BF_1	0%	10402	2.65
11	25	40	5	1-point	6n	DP2	BF_1	0%	754	4.27
12	25	40	5	1-point	6n	Off	50%	0%	13831	3.13
13	25	40	5	1-point	6n	Off	200%	0%	12921	0.79
14	64	40	5	1-point	6n	Off	200%	0%	8478	1.14
15	25	40	5	1-point	6n	Off	200%	1%	12811	0.77
16	36	40	5	1-point	6n	Off	200%	1%	11694	0.83
17	36	40	5	2-point	6n	Off	200%	1%	13504	0.77
18	25	40	5	2-point	12n	Off	200%	1%	26996	0.52
19	25	60	5	2-point	12n	Off	200%	1%	19676	1.13
20	36	60	5	2-point	12n	Off	200%	1%	17328	0.64

Table 5. Results of testing on the genetic algorithm

with our described approach, changing one parameter at a time. From these results we can observe that the best results come from test 18. However, it is worthy of note at this point that there is still a non-zero average deviation from the best known solution. This deviation exists due to the fact that not all jobs will have their best known solution found by any one method. As stated, our best results come from test 18; but these tests find the best known solution for only 60.3% of the instances. In fact, test 20 finds more of the best known solutions, with 62.6%. For this reason we take a closer look at these results by separating the instances by number of jobs. These results can be seen in Table 6. The tests are numbered as in the previous tables, and the best results highlighted.

Table 6 shows that test 18 gives the best performance for 20, 30, 50, and 100 job instances. Test 11 is best for five job instances, and there is a three way tie between tests 16, 18, and 20 as the best result for 10 job instances. It is worthy of note that, while test 11 is the best genetic algorithm test for five job instances, and multiple tests give the same results for 10 job instances, we know we can obtain optimal results by using either of the formulations.

	Number of jobs					
Test#	5	10	20	30	50	100
1	0.30	1.17	3.19	3.08	2.94	2.08
2	0.34	0.58	1.60	2.19	2.20	1.70
3	0.25	0.53	1.29	1.75	1.54	1.94
4	0.23	0.35	0.90	1.14	1.47	2.46
5	0.23	0.31	0.90	1.23	1.60	2.61
6	0.23	0.68	1.87	2.12	2.64	3.14
7	0.28	0.84	2.76	2.72	3.03	1.99
8	0.38	0.96	2.49	3.11	2.66	2.12
9	0.30	1.10	2.97	3.06	3.15	1.48
10	0.26	1.77	3.54	4.23	3.68	2.06
11	0.08	2.14	5.76	5.62	8.27	N/A
12	0.52	2.01	4.37	4.35	4.56	2.99
13	0.25	0.23	0.61	0.91	1.16	1.59
14	0.23	0.24	0.58	0.85	2.04	2.92
15	0.23	0.25	0.57	1.00	0.98	1.60
16	0.23	0.22	0.44	0.64	1.16	2.32
17	0.23	0.26	0.47	0.74	1.16	1.75
18	0.23	0.22	0.36	0.58	0.74	0.97
19	0.23	0.28	1.39	1.82	1.92	1.14
20	0.23	0.22	0.43	0.74	0.99	1.26

Table 6. Breakdown of result quality by number of jobs

n	L	Average % improvement
5	5	29.92
5	10	32.43
10	5	39.76
10	10	43.71
20	5	46.54
20	10	50.81
30	5	49.90
30	10	54.57
50	5	51.64
50	10	56.02
100	5	52.72
100	10	57.27
Average		47.11

Table 7. Improvements obtained by introducing a second robot

5.5 *Comparing to the single robot case*

To further assess the quality of our results we then compared our best known solutions for each of the 1,200 instances to the optimal solution for a version of the TRPP with just one robot. These optimal solutions can be found in $\mathcal{O}(n^2)$ time, using the algorithm described by Anily et al. (1999). We solved every instance twice, once for each robot being active, and took the best of these two makespans for comparison to our best known solution to the TRPP. The results of these tests can be seen in Table 7.

Again, the results are broken into groups based on instance type. The result given is the average improvement when using twin robots instead of a single robot. Note, the optimal single robot solution is always achieved with the described methods, whereas the twin robot results are upper bounds obtained from the metaheuristics for any instance with $n > 10$. The improvements continue to grow as the size of the instance increases, and for most larger instances the improvement is over 50%. An example of an instance with larger than 50% improvement is shown in Figure 8. Chart A shows the set of five jobs if completed by the white robot, Chart B shows the black robot processing the jobs, and Chart C shows the twin robots operating. The makespan of Charts A and B is 30, and the makespan in Chart C is 12. This instance is a bad case for the single robots as all jobs are located close to the ends of the rail. This means that, for a single robot, there will be a large amount of time spent travelling without handling a product. However, this case is easy for twin robots to process for the same reason. The close proximity of all jobs to the ends of the rail allows the robots to simply process their closest jobs, and no consideration of safety distance is required.

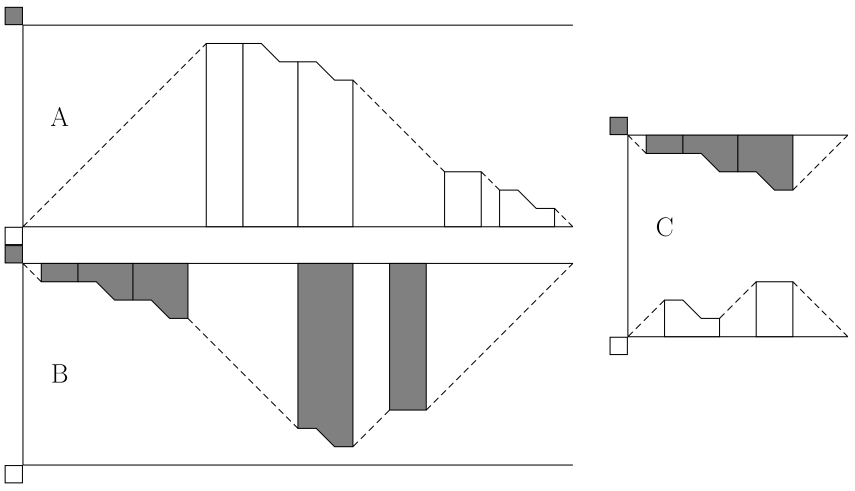

Figure 8. Example of the twin robots providing a 60% improvement from the single robot

5.6 *Analysis of Multiple Run Variance*

To further assess the strength of our methods we wish to perform replications of some tests. Since it produced our best results, we chose to run the hybrid genetic algorithm, with the best known parameter combination, on all instances, ten times. Figure 9 shows the results of these tests.

For each instance we obtained ten makespans. We then found the average of these results, and calculated the percentage deviation of

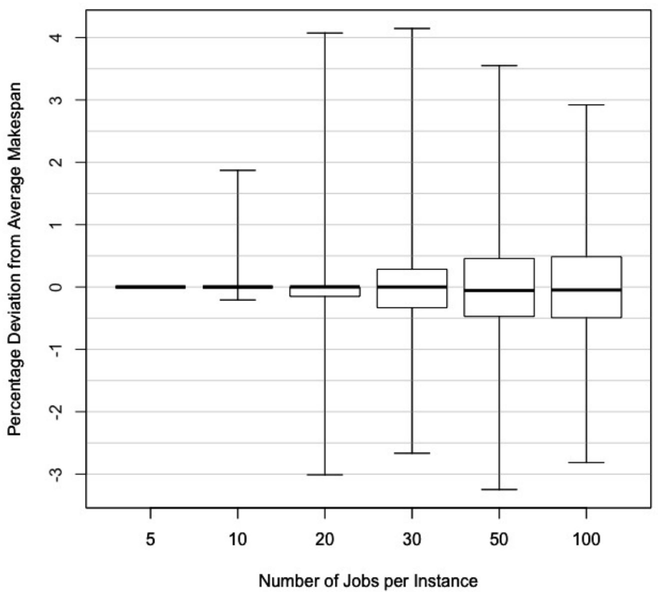

Figure 9. Box plots showing the variance in computational results.

each individual makespan from the average. We then grouped these results by the number of jobs in the instance to produce Figure 9. The figure shows that, regardless of instance size, at least 50% of results are within 0.5% of the average. Also, for five job instances, no results have any deviation from the average, and for ten job instances all but two of the results had zero deviation.

n	Average absolute deviation	Mean squared deviation	Minimum deviation	Maximum deviation
5	0.000	0.000	0.000	0.000
10	0.002	0.002	-0.208	1.871
20	0.328	0.414	-3.010	4.072
30	0.486	0.524	-2.665	4.146
50	0.576	0.579	-3.247	3.550
100	0.602	0.586	-2.815	2.921
Overall	0.333	0.351	-3.247	4.146

Table 8. Summary of deviations from average makespan

To further evaluate these results, we calculated the average absolute deviation, and mean squared deviation, for each size of instance. We also provide the maximum and minimum deviations. These results can be seen in Table 8.

6. Conclusions

This paper has introduced the TRPP and proved that the problem is $\mathcal{NP}$-Hard. We have presented two exact methods and two metaheuristic algorithms. We have shown the performance of all presented methods, comparing them to each other, testing parameters within these methods where applicable, and comparing our best results to a single robot version of the same problem. We have shown that the performance of methods is highly dependent on the size of input to the problem. Our best heuristic to date is the hybrid genetic algorithm.

Acknowledgements

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grant 2015-06189. This support is gratefully acknowledged. This research made use of the Balena High Performance Computing (HPC) at the University of Bath. Thanks are due to the referees and the Editor for their valuable comments.

References

AllGlass (2017). Robot onto rails. http://www.allglass.it/macchine.php?m=50.

Anily, S., M. Gendreau, and G. Laporte (1999). The swapping problem on a line. *SIAM Journal on Computing* 29(1): 327–335.

Boysen, N., D. Briskorn, and S. Emde (2014). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics* 62: 16–22.

Boysen, N., D. Briskorn, and F. Meisel (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research* 258: 343–357.

Diabat, A. and E. Theodoru (2014). An integrated quay crane assignment and scheduling problem. *Computers & Industrial Engineering* 73: 115–123.

Erdoğan, G., M. Battarra, and G. Laporte (2013). Scheduling twin robots on a line. *Naval Research Logistics* 61(2): 119–130.

Ge, Y. and Y. Yih (1995). Crane scheduling with time windows in circuit board production lines. *International Journal of Production Research* 33(5): 1187–1199.

Guan, Y., K.-H. Yang, and Z. Zhou (2013). The crane scheduling problem: models and solution approaches. *Annals of Operations Research* 203: 119–139.

Hakam, M., W. Solvang, and T. Hammervoll (2012). A genetic algorithm approach for quay crane scheduling with non-interference constraints at narvik container terminal. *International Journal of Logistics: Research and Applications* 15(4): 269–281.

Hall, N. and M. Posner (2001). Generating experimental data for computational testing with machine scheduling applications. *Operations Research* 49(6): 854–865.

Javanshir, H. and S. Seyedalizadeh-Ganji (2010). Yard crane scheduling in port container terminals using genetic algorithm. *Journal of Industrial Engineering International* 6(11): 39–50.

Larraønaga, P., C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review* 13: 129–170.

LeBlanc, R. (2017). The 2 billion pallet man. http://packagingrevolution.net/the-2-billion-pallet-man/.

Lee, D.-H. and J. H. Chen (2010). An improved approach for quay crane scheduling with non-crossing constraints. *Engineering Optimization* 42(1): 1–15.

Lee, D.-H. and H. Q. Wang (2010). Integrated discrete berth allocation and quay crane scheduling in port container terminals. *Engineering Optimization* 42(8): 747–761.

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

Lim, A., B. Rodrigues, and Z. Xu (2007). A m-parallel crane scheduling problem with a non-crossing constraint. *Naval Research Logistics* 54: 115–127.

Liu, J., Y. wah Wan, and L. Wang (2006). Quay crane scheduling at container terminals to minimize the maximum relative tardiness of vessel departures. *Naval Research Logistics* 53: 60–74.

Lourenço, H., O. Martin, and T. Stutzle (2003). Iterated local search. *Handbook of Metaheuristics, ISORMS* 57: 320–353.

Maschietto, G., Y. Ouazene, M. Ravetti, M. de Souza, and F. Yalaoui (2017). Crane scheduling problem with non-interference constraints in a steel coil distribution centre. *International Journal of Production Research* 55(6): 1607–1622.

Pratap, S., M. Kumar B, D. Saxena, and M. Tiwari (2016). Integrated scheduling of rake and stockyard management with ship berthing: a block based evolutionary algorithm. *International Journal of Production Research* 54(14): 4182–4204.

Rodriguez-Molins, M., M. A. Salido, and F. Barber (2014). A grasp-based metaheuristic for the berth allocation problem and the quay crane assignment problem by managing vessel cargo holds. *Applied Intelligence* 40: 273–290.

Tang, L., J. Zhao, and J. Liu (2014). Modeling and solution of the joint quay crane and truck scheduling problem. *European Journal of Operational Research* 236: 978–990.

Vidal, T., T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research* 60(3): 611–624.

Wang, K., W. Ma, H. Luo, and H. Qin (2016). Coordinated scheduling of production and transportation in a two-stage assembly flowshop. *International Journal of Production Research* 54(22): 6891–6911.

Yang, Y., Y. Chen, and C. Long (2016). Flexible robotic manufacturing cell scheduling problem with multiple robots. *International Journal of Production Research* 54(22): 6768–6781.

Zhao, F., Z. Shao, J. Wang, and C. Zhang (2016). A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems. *International Journal of Production Research* 54(4): 1039–1060.

Zhu, Y. and A. Lim (2006). Crane scheduling with non-crossing constraint. *Journal of the Operational Research Society* 57: 1464–1471.

Appendix A. Scheduling coefficient pseudocodes

Algorithm 7 Populate α matrix

```
for $i = 1, ..., n$ do
    for $j = 1, ..., n$ do
        if $(p_i \leq d_i)$ then
            if $(p_j \geq d_j)$ then
                if $(d_i < d_j)$ then
                    $\alpha^1_{i,j} = 0$
                else
                    if $(p_i < d_j$ and $d_j \leq d_i)$ then
                        $\alpha^1_{i,j} = \tau(p_j - d_j) - \tau(d_j - p_i) + 2\mu$
                    else
                        $\alpha^1_{i,j} = \tau(p_i - d_j + \sigma) + \tau(p_j - d_j) + 2\mu$
            else
                if $(d_j \leq p_i)$ then
                    $\alpha^1_{i,j} = \tau(p_i - d_j + \sigma) + \tau(d_j - p_j) + 2\mu$
                else
                    if $(p_j \leq p_i$ or $d_j \leq d_i)$ then
                        $\alpha^1_{i,j} = \tau(p_i - p_j + \sigma) + \mu$
                    else
                        if $(p_j \leq d_i)$ then
                            $\alpha^1_{i,j} = \tau(p_i - p_j + \sigma)$
                        else
                            $\alpha^1_{i,j} = 0$
        else
            if $(p_j < d_j)$ then
                if $(p_j > p_i)$ then
                    $\alpha^1_{i,j} = 0$
                else
                    if $(d_j \leq p_i)$ then
                        $\alpha^1_{i,j} = \tau(p_i - d_j + \sigma) + \tau(d_j - p_j) + 2\mu$
                    else
                        $\alpha^1_{i,j} = \tau(p_i - p_j + \sigma) + \mu$
            else
                if $(d_j \leq p_i)$ then
                    $\alpha^1_{i,j} = \tau(p_i - d_j + \sigma) + \tau(p_j - d_j) + 2\mu$
                else
                    $\alpha^1_{i,j} = 0$
```

Algorithm 8 Populate γ matrix

```
for $i = 1, ..., n$ do
    for $j = 1, ..., n$ do
        if $(p_i \leq d_i)$ then
            if $(p_j > d_j)$ then
                if $(d_j > d_i)$ then
                    $\gamma^1_{i,j} = 0$
                else
                    if $(p_j \leq d_i)$ then
                        $\gamma^1_{i,j} = 2\mu + \tau(2d_i - p_i - p_j + \sigma)$
                    else
                        $\gamma^1_{i,j} = \mu + \tau(2d_i - p_i - p_j) + \sigma$
            else
                if $(p_j > d_i)$ then
                    $\gamma^1_{i,j} = 0$
                else
                    $\gamma^1_{i,j} = 2\mu + \tau(2d_i - p_i - p_j + \sigma)$
        else
            if $(p_j > d_j)$ then
                if $(d_j > p_i)$ then
                    $\gamma^1_{i,j} = 0$
                else
                    if $(p_j \leq d_i)$ then
                        $\gamma^1_{i,j} = 2\mu + \tau(p_i - p_j + \sigma)$
                    else
                        if $(d_j \leq d_i$ or $p_j \leq p_i)$ then
                            $\gamma^1_{i,j} = \mu + \tau(p_i - p_j + \sigma)$
                        else
                            $\gamma^1_{i,j} = \tau(p_i - p_j + \sigma)$
            else
                if $(p_j > p_i)$ then
                    $\gamma^1_{i,j} = 0$
                else
                    if $(p_j \leq d_i)$ then
                        $\gamma^1_{i,j} = 2\mu + \tau(p_i - p_j + \sigma)$
                    else
                        $\gamma^1_{i,j} = \tau(p_i - p_j + \sigma) + \mu$
```

Appendix B. Scheduling algorithm

Here, pW and pB are used to record the previous job completed by the white and black robots respectively. S_i is the start time of job i and E_i is the end time of job i. The parameters Asn, α, and γ are as defined earlier.

Algorithm 9 Scheduling Algorithm

```
1: $pW = 0$, $pB = 0$
2: for $i = 1, ..., n$ do
3:     if $pB = 0$ and $pW = 0$ then
4:         if $Asn_i = white$ then
5:             $S_i = p_i$
6:             $pW = i$
7:         else
8:             $S_i = L + 1 - p_i$
9:             $pB = i$
10:    else
11:        if $Asn_i = white$ then
12:            if $pW = 0$ then
13:                $S_i = S_{pB}$
14:                if $S_i < p_i$ then
15:                    $S_i = p_i$
16:                if $S_i > S_{pB} - \gamma^1_{i,pB}$ and $S_i < S_{pB} + \alpha^1_{i,pB}$ then
17:                    $S_i = b$
18:            else
19:                if $pB = 0$ then
20:                    $S_i = E_{pW} + |p_i - d_{pW}|$
21:                else
22:                    $S_i = S_{pB}$
23:                    if $S_i < E_{pW} + |d_{pW} - p_i|$ then
24:                        $S_i = E_{pW} + |d_{pW} - p_i|$
25:                    if $S_i > S_{pB} - \gamma^1_{i,pB}$ and $S_i < S_{pB} + \alpha^1_{i,pB}$ then
26:                        $S_i = b$
27:                    $pW = i$
28:        else
29:            if $pB = 0$ then
30:                $S_i = S_{pW}$
31:                if $S_i < L + 1 - p_i$ then
32:                    $S_i = L + 1 - p_i$
33:                if $S_i > S_{pW} - \gamma^2_{i,pW}$ and $S_i < S_{pW} + \alpha^2_{i,pW}$ then
34:                    $S_i =$
35:            else
36:                if $pW = 0$ then
37:                    $S_i = E_{pB} + |p_i - d_{pB}|$
38:                else
39:                    $S_i = S_{pW}$
40:                    if $S_i < E_{pB} + |d_{pB} - p_i|$ then
41:                        $S_i = E_{pB} + |d_{pB} - p_i|$
42:                    if $S_i > S_{pW} - \gamma^2_{i,pW}$ and $S_i < S_{pW} + \alpha^2_{i,pW}$ then
43:                        $S_i = b$
44:                    $pB = i$
45:            $E_i = S_i + 2 + |p_i - d_i|$
46: $w = \max(E_{pW} + d_{pW}, E_{pB} + L + 1 - d_{pB})$
```

Appendix C. Time-indexed formulation

Let J be the set of all jobs, and Γ be the set of all time instants $\{0, ..., T\}$, where T is an upper bound for the makespan, calculated as the sum of all worst-case job processing times, given by (C1):

$$T = \sum_{i=1}^{n} \max\{p_i + d_i, 2(L+1) - (p_i + d_i)\} + 2\mu + |p_i - d_i|. \tag{C1}$$

We then have the following parameters: L, μ, p_i, and d_i as defined earlier, where $i \in J$. Our decision variables are then: x_t, the position of the white robot at time $t \in \Gamma$; y_t, the position of the black robot at time $t \in \Gamma$; s_{ti} which is 1 if job $i \in J$ starts at time instance $t \in \Gamma$ and 0 otherwise; e_{ti} which is 1 if job $i \in J$ ends at time instance $t \in \Gamma$ and 0 otherwise; z_i which is 1 if job $i \in J$ is performed by the white robot and 0 otherwise; l_{ij} which is 1 if job $i \in J$ is performed before job $j \in J$ and 0 otherwise; and the makespan w. The formulation is as follows:

$$
\begin{aligned}
\text{minimise } & w & & \text{(C2)}\\
\text{subject to } & x_0 = 0 & & \text{(C3)}\\
& y_0 = L + 1 & & \text{(C4)}\\
& x_t \geq x_{t-1} - 1 & t \in \Gamma \setminus \{0\} \quad & \text{(C5)}\\
& x_t \leq x_{t-1} + 1 & t \in \Gamma \setminus \{0\} \quad & \text{(C6)}\\
& y_t \geq y_{t-1} - 1 & t \in \Gamma \setminus \{0\} \quad & \text{(C7)}\\
& y_t \leq y_{t-1} + 1 & t \in \Gamma \setminus \{0\} \quad & \text{(C8)}\\
& x_t \leq y_t - 1 & t \in \Gamma \setminus \{0\} \quad & \text{(C9)}\\
& \sum_t s_{ti} = 1 & i \in J \quad & \text{(C10)}\\
& \sum_t e_{ti} = 1 & i \in J \quad & \text{(C11)}\\
& w \geq te_{ti} + z_i d_i + (1 - z_i)(L + 1 - d_i) & i \in J; t \in \Gamma \setminus \{0\} \quad & \text{(C12)}\\
& \sum_t te_{it} - \sum_t ts_{it} \geq 2\mu + |d_i - p_i| & i \in J \quad & \text{(C13)}\\
& x_t \leq z_i p_i + (L+1)(2 - z_i - s_{ti}) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C14)}\\
& x_t \geq p_i(z_i + s_{ti} - 1) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C15)}\\
& y_t \leq (1 - z_i)p_i + (L+1)(1 + z_i - s_{ti}) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C16)}\\
& y_t \geq p_i(s_{ti} - z_i) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C17)}\\
& x_t \leq z_i d_i + (L+1)(2 - z_i - e_{ti}) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C18)}\\
& x_t \geq d_i(z_i + e_{ti} - 1) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C19)}\\
& y_t \leq (1 - z_i)d_i + (L+1)(1 + z_i - e_{ti}) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C20)}\\
& y_t \geq d_i(e_{ti} - z_i) & t \in \Gamma \setminus \{0\}; i \in J \quad & \text{(C21)}\\
& \sum_t te_{tj} - \sum_t ts_{ti} \leq T(l_{ij} + 2 - z_i - z_j) & i, j \in J \quad & \text{(C22)}
\end{aligned}
$$

$$\sum_t ts_{tj} - \sum_t te_{ti} \geq T(l_{ij} + z_i + z_j - 3) \qquad i, j \in J \qquad \text{(C23)}$$

$$\sum_t te_{tj} - \sum_t ts_{ti} \leq T(l_{ij} + z_i + z_j) \qquad i, j \in J \qquad \text{(C24)}$$

$$\sum_t ts_{tj} - \sum_t te_{ti} \geq T(l_{ij} - 1 - z_i - z_j) \qquad i, j \in J \qquad \text{(C25)}$$

$$\sum_{t'=t+1}^{t+\mu} x_{t'} - \mu x_t \leq L\mu(2 - s_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C26)}$$

$$\sum_{t'=t+1}^{t+\mu} x_{t'} - \mu x_t \geq -L(2 - s_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C27)}$$

$$\sum_{t'=t+1}^{t+\mu} y_{t'} - \mu y_t \leq L\mu(1 - s_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C28)}$$

$$\sum_{t'=t+1}^{t+\mu} y_{t'} - \mu y_t \geq -L(1 - s_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C29)}$$

$$\sum_{t'=t}^{t+\mu-1} x_{t'} - \mu x_{t+\mu} \leq L\mu(2 - e_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C30)}$$

$$\sum_{t'=t}^{t+\mu-1} x_{t'} - \mu x_{t+\mu} \geq -L(2 - e_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C31)}$$

$$\sum_{t'=t}^{t+\mu-1} y_{t'} - \mu y_{t+\mu} \leq L\mu(1 - e_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C32)}$$

$$\sum_{t'=t}^{t+\mu-1} y_{t'} - \mu y_{t+\mu} \geq -L(1 - e_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \qquad \text{(C33)}$$

$$l_{ij} + l_{ji} = 1 \qquad i, j \in J; i \neq j \qquad \text{(C34)}$$

$$x_t \in \{0, L\} \qquad t \in \Gamma \qquad \text{(C35)}$$

$$y_t \in \{1, L + 1\} \qquad t \in \Gamma \qquad \text{(C36)}$$

$$s_{ti}, e_{ti} \in \{0, 1\} \qquad t \in \Gamma; i \in J \qquad \text{(C37)}$$

$$z_i \in \{0, 1\} \qquad i \in J \qquad \text{(C38)}$$

$$l_{ij} \in \{0, 1\} \qquad i, j \in J. \qquad \text{(C39)}$$

Constraints (C3) and (C4) ensure that the robots start a palletising run in their respective depots. Constraints (C5), (C6), (C7), and (C8) maintain a travel time of $\tau = 1$, and Constraints (C9) maintain a safety distance $\sigma = 1$. Constraints (C10) state that a job can only start at one time, and Constraints (C11) state that a job can only end at one time. Constraints (C12) ensures that w is at least the end time of any job, plus the time taken for the assigned robot to return to its depot. Constraints (C13) set the duration of jobs. Constraints (C14), (C15), (C16), and (C17) ensure that if a robot is assigned to a job, then it will be at the pick up location of the job at its starting time. Constraints (C18), (C19), (C20), and (C21) ensure that if a robot is assigned to a job, then it will be at the delivery location of the job at its end time. Constraints (C22), (C23), (C24), and (C25) ensure that a job cannot be started by a robot until that robot has completed all previous jobs assigned to it. Constraints (C26), (C27), (C28), (C29), (C30), (C31), (C32), and (C33) ensure that the process time μ is respected at the start and end of all jobs. Constraints (C34) state that either job i is before job j, or vice versa; both statements cannot be true. Constraints (C35), (C36), (C37), (C38), and (C39) define the domains of the variables.

Appendix D. Full results of initial formulation testing

		Graph-representation					Time-indexed				
n	L	Optimal solution found?	Upper bound	Lower bound	% gap	CPU Time	Optimal solution found?	Upper bound	Lower bound	% gap	CPU Time
5	5	Y	16	16	0.0	0.75	Y	16	16	0.0	1.12
		Y	22	22	0.0	0.45	Y	22	22	0.0	1.61
		Y	16	16	0.0	0.50	Y	16	16	0.0	1.00
		Y	18	18	0.0	0.44	Y	18	18	0.0	1.17
		Y	16	16	0.0	0.39	Y	16	16	0.0	1.14
		Y	17	17	0.0	0.34	Y	17	17	0.0	1.17
		Y	20	20	0.0	0.37	Y	20	20	0.0	1.26
		Y	14	14	0.0	0.31	Y	14	14	0.0	0.91
		Y	18	18	0.0	0.28	Y	18	18	0.0	1.06
		Y	17	17	0.0	0.33	Y	17	17	0.0	1.26
6	5	Y	18	18	0.0	1.54	Y	18	18	0.0	1.37
		Y	20	20	0.0	1.15	Y	20	20	0.0	2.78
		Y	14	14	0.0	0.33	Y	14	14	0.0	1.31
		Y	15	15	0.0	0.41	Y	15	15	0.0	1.70
		Y	22	22	0.0	2.15	Y	22	22	0.0	3.45
		Y	12	12	0.0	0.37	Y	12	12	0.0	1.45
		Y	18	18	0.0	1.31	Y	18	18	0.0	1.93
		Y	15	15	0.0	0.36	Y	15	15	0.0	1.84
		Y	19	19	0.0	0.59	Y	19	19	0.0	1.92
		Y	17	17	0.0	0.62	Y	17	17	0.0	1.72
7	5	Y	25	25	0.0	10.01	Y	25	25	0.0	4.79
		Y	28	28	0.0	9.76	Y	28	28	0.0	8.58
		Y	23	23	0.0	7.41	Y	23	23	0.0	5.49
		Y	22	22	0.0	3.13	Y	22	22	0.0	2.98
		Y	17	17	0.0	1.00	Y	17	17	0.0	3.45
		Y	28	28	0.0	19.84	Y	28	28	0.0	16.44
		Y	16	16	0.0	2.36	Y	16	16	0.0	3.37
		Y	24	24	0.0	5.91	Y	24	24	0.0	5.69
		Y	21	21	0.0	1.93	Y	20	20	0.0	5.21
		Y	22	22	0.0	2.51	Y	22	22	0.0	4.48
8	5	Y	22	22	0.0	10.46	Y	22	22	0.0	10.70
		Y	18	18	0.0	2.81	Y	18	18	0.0	7.79
		Y	28	28	0.0	50.17	Y	28	28	0.0	8.07
		Y	27	27	0.0	54.32	Y	27	27	0.0	27.00
		Y	23	23	0.0	54.88	Y	23	23	0.0	8.07
		Y	22	22	0.0	7.19	Y	22	22	0.0	6.40
		Y	25	25	0.0	55.69	Y	25	25	0.0	13.46
		Y	22	22	0.0	7.63	Y	22	22	0.0	6.92
		Y	20	20	0.0	8.84	Y	20	20	0.0	7.02
		Y	22	22	0.0	23.02	Y	22	22	0.0	11.69
9	5	Y	28	28	0.0	341.85	Y	28	28	0.0	37.82
		Y	24	24	0.0	195.34	Y	24	24	0.0	21.31
		Y	20	20	0.0	13.73	Y	20	20	0.0	13.17
		Y	23	23	0.0	116.34	Y	23	23	0.0	25.29
		Y	22	22	0.0	127.73	Y	22	22	0.0	18.66
		Y	29	29	0.0	2068.93	Y	29	29	0.0	40.78
		Y	36	36	0.0	2445.69	Y	36	36	0.0	137.90
		Y	25	25	0.0	1138.18	Y	25	25	0.0	35.27
		Y	22	22	0.0	110.77	Y	22	22	0.0	16.32
		Y	21	21	0.0	54.57	Y	21	21	0.0	11.14
10	5	Y	24	24	0.0	1162.13	Y	24	24	0.0	51.13
		Y	23	23	0.0	3715.34	Y	23	23	0.0	32.48
		Y	27	27	0.0	4724.22	Y	27	27	0.0	190.80
12	5	N	28	25	12.0	7200.00	Y	28	28	0.0	222.70
		N	29	25	16.0	7200.00	Y	27	27	0.0	350.25
		N	30	26	15.4	7200.00	Y	30	30	0.0	349.78
15	5	N	57	40	42.5	7200.00	N	54	16.7	223.3	7200.00
		N	45	36	25.0	7200.00	N	38	17.0	123.3	7200.00
		N	36	28	28.6	7200.00	N	35	26	34.6	7200.00
20	5	N	54	38	42.1	7200.00	N	46	14	228.6	7200.00
		N	54	40	35.0	7200.00	N	49	12.3	298.7	7200.00
		N	44	34	29.4	7200.00	N	42	13.4	214.2	7200.00