Optimisation of Scheduling Problem with Industrial Applications (Alternative Format Thesis)

Thomasson, Ollie

Award date:
2021

Awarding institution:
University of Bath

Optimisation of Scheduling Problem with Industrial Applications (Alternative Format Thesis)

Thomasson, Ollie

Award date:
2021

Awarding institution:
University of Bath

Optimisation of Scheduling Problems with Industrial Applications

submitted by

Oliver Thomasson

for the degree of Doctor of Philosophy

of the

University of Bath

School of Management
Information, Decisions and Operations Division

May 2021

Abstract

This thesis collects the work completed for my PhD. I begin with a discussion of my reasons for taking on a PhD, and describe how my developing ideas of the researcher I wanted to be led me to produce the research I did.

I then present the three papers written during the course of my PhD program. Each paper introduces an original problem which may be found in industry. Through a combination of mathematical programming and heuristic methods, each problem is solved.

Two of the papers in this thesis present problems found in automated palletising systems. The third presents a problem found in the production factories of Calzedonia, involving human workers.

Finally, I discuss the scientific contributions of the thesis, limitations within, and areas of research I am looking to explore in the future.

Acknowledgements

I first thank my outstanding supervisory team. **Dr Maria Battarra**, without her I would have not considered the idea of doing a PhD. Her unwavering support has been invaluable at every turn, even at the times where I put her through great difficulty in the supervision. She has been so much more than a supervisor, and I will never be able to thank her enough for everything she has done for me. **Prof Güneş Erdoğan** who has offered such excellent guidance throughout the process. His methodological coaching in the earlier phases pushed me to understand algorithms to a depth I did not realise I was capable of. His comments on every document have been swift and valuable. Though he has found himself at odds with my stubbornness on occasion, he has always shown me great kindness and respect. **Prof Gilbert Laporte**, who has shown great interest in me and my PhD despite being one of the greatest academics in operational research. His highly valuable insights have improved my work at every stage, and his passion for research will continue to inspire me in my future career.

Next, my dearest friends. **Dr Brittany Davidson** who I began the PhD alongside all those years ago. She has been an absolute rock of a friend throughout the process. Always there for whatever I need, and I only hope I have returned this in kind. To thank Britt fully would take more words than the entire remainder of this thesis. **Dr Joanne Hinds** who has offered such support through the best and worst times, from celebrating our successes at the Crystal Palace, to giving me a place to stay when I had nowhere else to go. I look forward to our long careers together. **Mehrnaz Tajmir** for sharing both the highs and the lows of the PhD experience, especially the walks around campus to work through the hardest times.

Then, my **Mum**, **Dad**, and sister **Katie**. When I pushed them to their limit, they still didn't give up on me, despite having every right to do so. Instead, they gave me every support I needed to get myself back in gear to finally finish this thing!

To all those at **The Threshold Centre** co-housing community, who welcomed me with open arms for the six months I took off to get myself back in working order, especially to **Chris Riley** for only showing care and love when I dropped into his life like a bomb. Also to **Jill Elms** and **Lynda Mills** for being excellent housemates. Not forgetting **Lionel**, **Stevie**, and **Mutti** for their "walking therapy", and allowing me to give them so many pets!

Suzanne (Sooz) Swallow, for helping me so much more than her role could possibly require. She has been a wonderful support, and I am glad that me finally finishing

brings her one step closer to a relaxing retirement, she deserves it!

To my IDO division desk buddies. **Gozdem Dural-Selcuk**, **Marianna Frangeskou**, **Neo Stylianou**, **Mehrnoush Sarafan**, **Jaz Kalra**, **Renate Taubeneder**, **Juliette Englehart**, and **Sian Smith**. Beyond sharing a desk we also shared laughs, success and difficulties. Each of you have contributed in some way to my development as an academic and as a person. Of course, as well I thank the whole of the IDO Division, for help and support with everything from learning how to use the printer on day one, teaching advice and help, to larger career advice. I am thrilled to have acquired my first academic job in this division, and look forward to many more years here.

Last but not least, my thanks go to all those members of the mental health team in Student Servies at the Univeristy of Bath, and the many members of the National Health Service, who helped me overcome the mental health challenges that almost sunk my PhD for good.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

This thesis focuses on scheduling problems that occur in the manufacturing industry. We consider both scheduling problems in automated and manual production systems. In some cases we take an existing industrial setup, and design new scheduling techniques to improve the efficiency of the system. In other cases we consider existing, well-scheduled systems, and suggest augmentations to the system which could be advantageous if paired with appropriate scheduling methods. For each problem studied in this thesis, the goal is to design an algorithm to solve the scheduling problem. Due to the variety of techniques available for algorithmic development, we in fact produce many algorithms for each scheduling problem. Each algorithm has its own strengths and weaknesses. Some find optimal schedules, some run very quickly, some find a balance of a non-optimal, but very good solutions with an acceptably fast running time.

The idea of doing a PhD first occurred during the final year of my undergraduate studies. I was in the third year of a BSc Mathematics degree at the University of Southampton. I had realised that coursework style assessment was a strength of mine, and to achieve the first class degree I knew I was capable of, I would have to carefully select my final year modules to ensure that the assessments suited my working style. Viewing the module sign-up sheet, one option stood out to me. It was a module simply titled "Mathematics Project", with an assessment of 100% coursework. I knew this played to my strengths, and upon further reading it seemed to me that this module was in essence a small dissertation, since the degree program I had chosen did not have a compulsory dissertation. We were told that to take this module we had to choose a supervisor in the School of Mathematics who would work with us on a project over the course of a semester. I was not quite sure what I wanted my mathematics project to be, so I turned to Dr Maria Battarra. At the time I was simply asking if Maria would be happy to supervise me for a semester-long project, not realising quite the scale of impact this decision would take on the majority of the next decade of my life. The initial project pitch was reasonably simple. Maria had a published paper, with her co-authors Professor Güneş Erdoğan, and Professor Gilbert Laporte, exploring a scheduling problem with robots on a rail tasked with moving products (Erdoğan et al., 2014). She had an idea for an extension to this, and wondered if I could design an algorithm to solve this new problem. The next four months consisted of learning how to use the C coding language, spending many long nights in the university library debugging code, and building a genuine adoration of the process of problem solving through algorithmic development, coding, and testing.

By the time the project deadline came I had written my first algorithm. It was a reasonably simple algorithm when compared to the work you will see in this thesis, but it was mine. I had looked at a problem to which no one had developed a solution, designed a process in my mind to solve it, transferred that to a white board, to an exercise book, to a pseudocode, and finally to a working computer code which could be given a set of products and decide how some robots should move them. At this point I had caught the research bug, and I knew I would have to do more. However, I had secured a place on the PGCE at Southampton, and if my final undergraduate year was when I caught the research bug, then it would have been around the age of seven that I caught the teaching bug, and had been taking as many opportunities to work in schools as I possibly could since then. I completed my PGCE, and started my Newly Qualified Teacher year (NQT) at a sixth form college in Southampton. While I was completing the PGCE, Maria had moved to the University of Bath, and during my NQT she reached out to see if I would still be interested in a PhD. Since my desire to research was still strong, I applied for a funded PhD at Bath, and as you can tell by the fact you're reading this thesis, I was successful.

First, we needed to finish the problem I started for my Mathematics Project. You will see the resulting paper presented in Chapter 2. To achieve this I had to learn much more than how to produce a constructive heuristic, which is the name for the type of algorithm I built for my Mathematics Project. The fact that it is a *heuristic* means that it will find a solution that satisfies the problem constraints, but does not necessarily find values that correspond to the best possible solution, called the *optimal solution*. We would therefore call this solution *feasible*, since it provides a possible solution to the problem, and *sub-optimal* since it is not guaranteed to find the optimal solution. The *constructive* part of constructive heuristic means that the heuristic begins without any known values for the variables, and builds a solution from scratch. The first major step of my PhD was to learn how to find optimal solutions. Here we should note that the class of problems which I solve in this thesis are called *Mixed-Integer Linear Programming Problems* (MILPs) (Floudas and Lin (2005), Vielma (2015)). The method at the core of many algorithms for solving IPs is the branch and bound algorithm. This method was initially proposed by Land and Doig (1960), and first named by Little et al. (1963). In practice, I solved integer programs using existing software. My software of choice was CPLEX (`https://www.ibm.com/uk-en/analytics/cplex-optimizer`). In most cases the algorithms used by CPLEX for solving IPs add more complex components to the branch and bound algorithm, but the differences are usually problem specific, so are discussed in the later chapters where necessary.

A challenge of solving IPs using software is the computational load that the IP often demands. In this thesis, some of the exact methods I introduce were not practically capable of finding optimal solutions, due to the complexity of the problems. Of course, the methods are always mathematically capable of finding these solutions, but could be constrained by the computational power available in my hardware, or the time required for an optimal solution to be found. For this reason, my next step was to build upon by existing knowledge of heuristic methods to develop *metaheuristics* (Glover and Kochenberger (2006), Talbi (2009)). Metaheuristics are a higher-level form of heuristic method. A simple example of a metaheuristic would be an algorithm to search through potential solutions and evaluate them. The search procedure would find solutions with promising properties, and pass this to an evaluating heuristic to find the solution value. Metaheuristics can be much more complex, but this simple two-stage type is where I began. There are many types of search procedure, but we initially focus on *local search*. Local search moves take a solution and identify *neighbours* i.e., solutions with only a small change from where we started. Most of the metaheuristic algorithms developed for this thesis are *local search based metaheuristics*, meaning that local search moves are an integral component of their solution selection process. Examples include Iterated Local Search, Variable Neighbourhood Search, and Hybrid Genetic algorithms. Since each algorithm is described at length in its respective chapter we do not explain them in depth here.

Learning about exact methods and metaheuristics enabled me to produce algorithms capable of solving my first problem. After writing up the findings, I was left to decide what the remainder of my PhD would consist of. The first problem had been in the field of scheduling, so it seemed appropriate to continue with this theme. Scheduling is the process by which tasks are assigned to resources, with a particular desired outcome. Some schedules are designed to maximise the number of tasks completed in a set time, some are designed to minimise the amount of idle time the resources have, some are designed to minimise the total time taken for all tasks to be completed, and some are designed with the aim of fairness so that no resource has significantly more work than any other. These objectives may interact or overlap, depending on the schedule being designed. In the design of a schedule it may be desired to find the best possible schedule. Alternatively it could be designed in such a way that the outcome is reasonably good, but calculable quickly. In practice, a schedule will be restricted by a set of conditions which put certain limits on the schedule design.

My largest contribution to the first paper was the development of the metaheuristic algorithms, and finding creative ways to improve their performance. For my second

paper I wanted to spend more time on the design of exact methods. Dr Battarra had been contacted by an ex-student working in industry, who had a problem we believed would be perfect for developing these skills. Through this paper I built a great deal of knowledge on exact methods, and these contributions can be seen in Chapter 4.

Between the first and second paper I had developed skills in constructing both exact algorithms and metaheuristics. For my final paper, I wished to return to the problem that made me initially take on the PhD, as I had thought of a way to generalise the problem. The main methodological learning I wished to achieve with this paper was to build algorithms which could take advantage of parallel computation.

1.2 Thesis overview

Now that the background and purpose of the thesis have been described, I outline the structure of the remainder of the thesis.

Chapter 2 presents the paper entitled Scheduling Twin-Robots in a Palletising Problem, published in the *International Journal of Production Research* (Thomasson et al., 2018). This paper introduces the Twin-Robot Palletising Problem (TRPP). Two mathematical models are introduced as well as two metaheuristic algorithms. A computational study is conducted, both to assess the performance of the methods, and to demonstrate the practical advantages of implementing this problem layout in an industrial environment.

Chapter 3 presents the paper entitled Pallet Assignment and Job Scheduling in a Twin-Robot System, which has been submitted to *Computers & Operations Research*. This paper introduces the Twin-Robot Pallet Assignment and Scheduling Problem (TRPASP), which is a generalisation of the TRPP from Chapter 2. A mathematical model is introduced to aid the problem definition, before outlining the four metaheuristic algorithms developed to solve the problem. A computational study is conducted to demonstrate the strengths and weaknesses of each metaheuristic, and to compare the best results to those found in Chapter 2.

Chapter 4 presents the paper entitled Algorithms for the Calzedonia Workload Allocation Problem, published in the *Journal of the Operational Research Society* (Battarra et al., 2020). This paper introduces the Calzedonia Workload Allocation Problem (WAP), stemming from a collaboration with the Italian garment manufacturer Calzedonia. We introduce two exact methods: a mathematical model; and a sequential algorithm formed of a model and heuristics to calculate bounds, and a mathematical

model to optimally solve given the bounds. Two heuristics are also presented before running computational tests to assess the performance of the algorithms.

Chapter 5 gives a final wrap-up, and suggests directions for future work.

Bibliography

Battarra, M., F. Fraboni, O. Thomasson, G. Erdoğan, G. Laporte, and M. Formentini (2020). Algorithms for the calzedonia workload allocation problem. *Journal of the Operational Research Society*.

Erdoğan, G., M. Battarra, and G. Laporte (2014). Scheduling twin robots on a line. *Naval Research Logistics 61*(2), 119–130.

Floudas, C. A. and X. Lin (2005). Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research 139*(1), 131–162.

Glover, F. W. and G. A. Kochenberger (2006). *Handbook of metaheuristics*, Volume 57. Springer Science & Business Media.

Land, A. H. and A. G. Doig (1960). An automatic method for solving discrete programming problems. *Econometrica 28*(3), 497–520.

Little, J. D., K. G. Murty, D. W. Sweeney, and C. Karel (1963). An algorithm for the traveling salesman problem. *Operations research 11*(6), 972–989.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*, Volume 74. John Wiley & Sons.

Thomasson, O., M. Battarra, G. Erdoğan, and G. Laporte (2018). Scheduling twin robots in a palletising problem. *International Journal of Production Research 56*(1-2), 518–542.

Vielma, J. P. (2015). Mixed integer linear programming formulation techniques. *Siam Review 57*(1), 3–57.

Chapter 2

Scheduling Twin Robots in a Palletising Problem

Statement of authorship

This declaration concerns the article entitled:
Scheduling twin robots in a palletising problem

Publication status: Published

Publication details:
Oliver Thomasson, Maria Battarra, Güneş Erdoğan, and Gilbert Laporte.
(2018)
Scheduling twin robots in a palletising problem.
International Journal of Production Research
56 (1-2), 518–542.
https://doi.org/10.1080/00207543.2017.1401249

Candidates contribution to the paper

- Discovery & exploration of idea (20%)
- Mathematical modelling (20%)
- Heuristic development (40%)
- Coding & testing (70%)
- Initial write-up (90%)
- Editing for publication (30%)

Statement from candidate
This paper reports on original research I conducted during the period of my Higher Degree by Research candidature

Signed:
[redacted]

Date:
Wednesday 7th July, 2021

2.1 Introduction

This paper introduces the Twin Robot Palletising Problem (TRPP) in which two robots must be scheduled 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, 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 2-1: An industrial example of twin robots on a rail (AllGlass, 2017).

To automate a palletising system like the one displayed in Figure 2-1 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 evaluated with an extensive computational study. Additionally, we will compare the TRPP to a problem with a single robot.

2.2 Problem definition

In order to define the TRPP, we refer the reader to Figure 2-2 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.

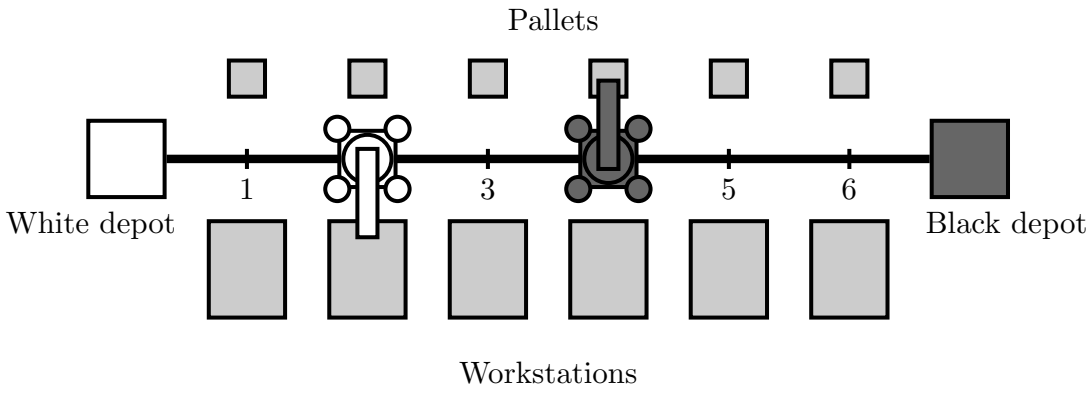

Figure 2-2: An industrial layout demonstrating the TRPP

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.

2.2.1 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. (2012) hybrid genetic algorithm for multi-depot and periodic VRP's; the details of which are provided in Section 2.4.5.

2.2.2 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 2-3. 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.

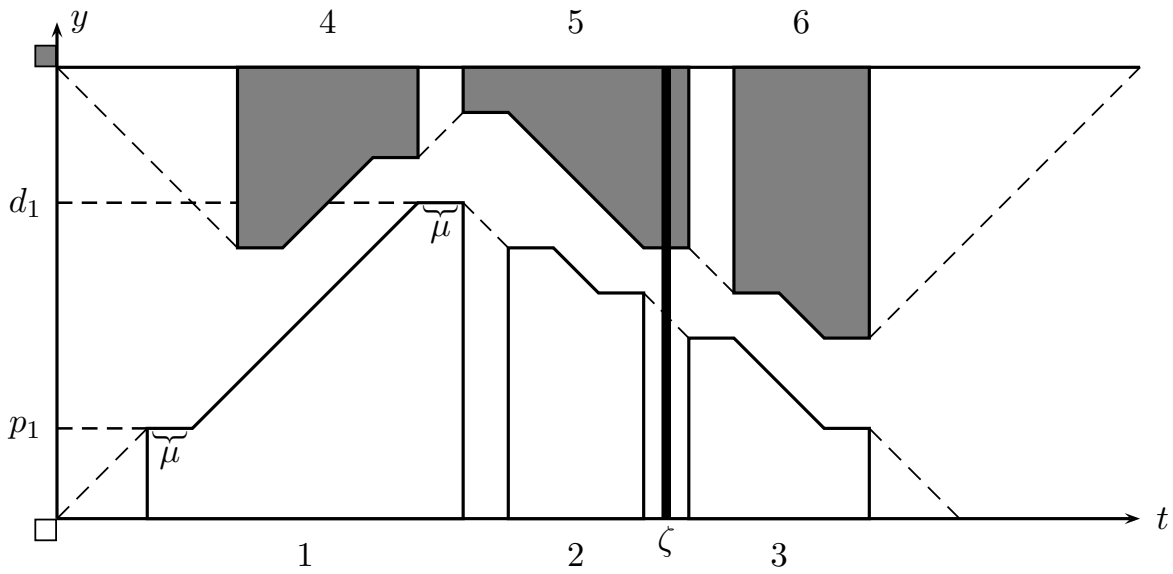

Figure 2-3: An example of a TRPP solution, represented as a Gantt Chart

Theorem 2.2.1. *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 2.1 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.

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.

Parameter	Symbol	TRSP	TRPP
Set of workstations	(Ψ_i)	$\sim$	✓
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 2.1: Inputs of the TRSP and TRPP.

The constraint of minimum safe distance, combined with our 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.

□

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 2-4. 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 2.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.

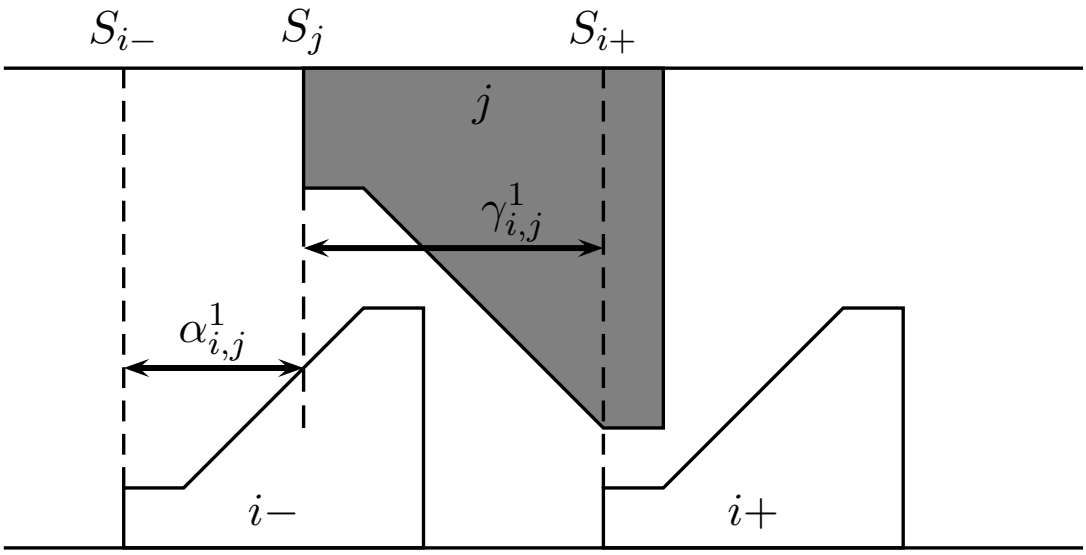

Figure 2-4: $\alpha^1_{i,j}$ and $\gamma^1_{i,j}$ coefficients

2.3 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 2.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.

2.3.1 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 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 2.2.2; and T is an upper bound for the makespan, calculated as the sum of all worst-case job processing times, given by (2.1):

$$T = \sum_{i=1}^{n} \max\{p_i + d_i, 2(L+1) - (p_i + d_i)\} + 2\mu + |p_i - d_i|. \tag{2.1}$$

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:

$$\text{minimise } w \tag{2.2}$$

$$\text{subject to } \sum_{j \in V_T} x_{0j} \leq 1 \tag{2.3}$$

$$\sum_{j \in V_T} x_{j0} = \sum_{j \in V_T} x_{0j} \tag{2.4}$$

$$\sum_{j \in V_T} y_{n+1,j} \leq 1 \tag{2.5}$$

$$\sum_{j \in V_T} y_{j,n+1} = \sum_{j \in V_T} y_{n+1,j} \tag{2.6}$$

$$\sum_{j \in V} (x_{ij} + y_{ij}) = 1 \qquad i \in V_T \tag{2.7}$$

$$\sum_{i \in V} x_{ij} = \sum_{i \in V} x_{ji} \qquad j \in V_T \tag{2.8}$$

$$\sum_{i \in V} y_{ij} = \sum_{i \in V} y_{ji} \qquad j \in V_T \tag{2.9}$$

$$w \geq \sum_{(i,j) \in A} \theta_{ij} x_{ij} \tag{2.10}$$

$$w \geq \sum_{(i,j) \in A} \theta_{ij} y_{ij} \tag{2.11}$$

$$s_j \geq s_i + \eta_{ij} - T(1 - x_{ij} - y_{ij}) \qquad i, j \in V_T \tag{2.12}$$

$$s_i \geq \eta_{0i} x_{0i} + \eta_{L+1,i} y_{L+1,i} \qquad i \in V_T \tag{2.13}$$

$$s_j \leq s_i - \alpha^1_{ij} + T(1 - z_{ij} + \sum_k y_{ki} + \sum_k x_{kj}) \qquad i, j \in V_T; k \in V \tag{2.14}$$

$$s_j \geq s_i + \gamma^1_{ij} - T(z_{ij} + \sum_k y_{ki} + \sum_k x_{kj}) \qquad i, j \in V_T; k \in V \tag{2.15}$$

$$w \geq s_i + \eta_{i0} x_{i0} + \eta_{i,L+1} y_{i,L+1} \qquad i \in V_T \tag{2.16}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A \tag{2.17}$$

$$y_{ij} \in \{0, 1\} \qquad (i, j) \in A \tag{2.18}$$

$$z_{ij} \in \{0, 1\} \qquad (i, j) \in A \tag{2.19}$$

$$s_i \geq 0 \qquad i \in V_T \tag{2.20}$$

$$w \geq 0. \tag{2.21}$$

Constraint (2.3) ensures the white robot leaves the depot at most once. Constraint (2.4) states that the number of jobs ending at the white depot is equal to the number of jobs originating there (at most one). Constraints (2.5) and (2.6) state the same conditions for the black robot. Constraints (2.7) state that jobs must be completed exactly once, by just one of the robots, and Constraints (2.8) and (2.9) ensure that all jobs are succeeded by another if performed by the white and black robot respectively. Constraints (2.10) and (2.11) limit the makespan to be at least the minimum time taken for either robot to complete its assigned jobs, and Constraints (2.12) 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 (2.13) 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 (2.14) and (2.15) 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 (2.16) 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 (2.17) to (2.21) define the domains of the variables.

2.4 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 2.5), 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.

2.4.1 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 2.2.2). 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 2.B. Initial solutions are produced by randomly assigning a robot and an order to each job, then applying this scheduling algorithm.

2.4.2 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 ($\mathcal{O}(n^4)$).

2.4.3 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 2-5. 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 2.3. To identify the potential for delay in Figure 2-5, 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.

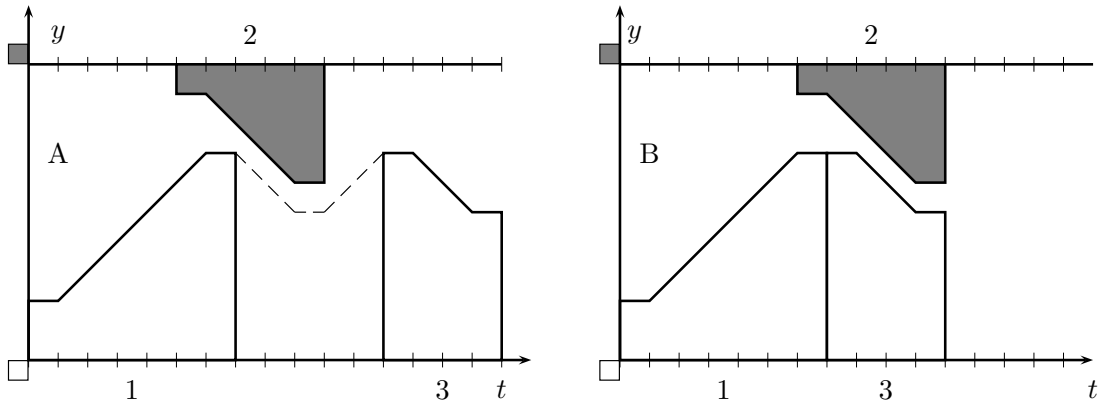

Figure 2-5: Example of a delay which benefits the makespan

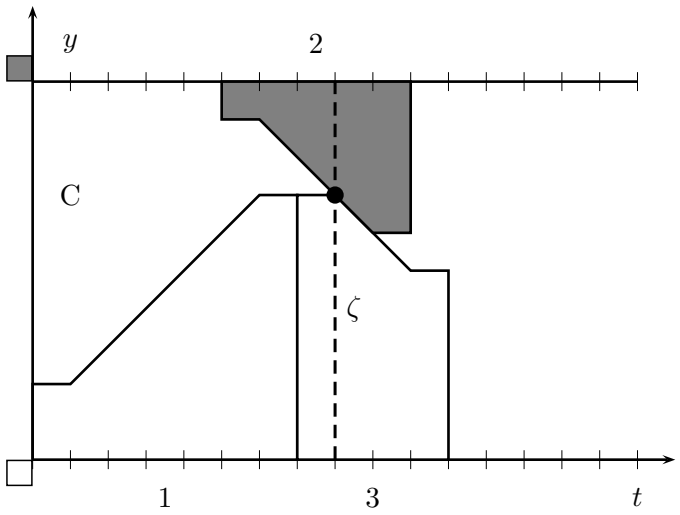

Figure 2-6: The delay calculation step of the dynamic program

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 2.1. 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 2.1.

Algorithm 2.1 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

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 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 latest. 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 2-6, we also consider the charts of Figure 2-5. 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).

2.4.4 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 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 2.2.

Algorithm 2.2 Iterated Local Search

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

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 2.3, 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 current 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 it 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 2.3 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: $k = k + 1$

2.4.5 Hybrid genetic algorithm

We now introduce the final heuristic we implemented, a hybrid genetic algorithm. A pseudocode is shown in Algorithm 2.4. 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 2-7.

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

Figure 2-7: The population matrix

In this algorithm R is the number of perturbations to perform on the initial solution (line 3 of Algorithm 2.4), and I is the number of improvements to perform on each perturbed solution (line 5 of Algorithm 2.4). The function *Perturb* is as defined in Section 2.4.4. 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.

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

Algorithm 2.4 Hybrid Genetic Algorithm

```
Generate Initial Solution ρ_{0,0}
for i ∈ {1, ..., R} do
    ρ_{i,0} = Perturb(ρ_{i−1,0})
    for j ∈ {1, ..., I} do
        ρ_{i,j} = Improve(ρ_{i,j−1})
while Stopping criteria not satisfied do
    DiversityCalculation(S)
    for i ∈ {1, ..., |Q|} do
        Select: P_1, P_2
        q_i = Crossover(P_1, P_2)
        Mutation(q_i)
        LocalSearch(q_i)
    DiversityCalculation(Q)
    Replacement(S, Q)
Return: Best(S)
```

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

$$\Delta(\rho) = \frac{1}{2|S|^2} \sum_{\rho_2=1}^{|S|} \sum_{i=1}^{|S|} (1(\pi_i(\rho) \neq \pi_i(\rho_2)) + 1(\xi_i(\rho) \neq \xi_i(\rho_2))) \tag{2.22}$$

$$BF_1(\rho) = fit(\rho) + \left(1 - \frac{nbElit}{|S|}\right) dc(\rho) \tag{2.23}$$

$$BF_2(\rho) = fit(\rho) + (DW) dc(\rho). \tag{2.24}$$

In (2.22), $|S|$ 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. (2012) 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 $BF_1(\rho)$ (2.23) 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 (2.24) is an alternative to the biased fitness calculation of Vidal et al. (2012). 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 2.5. 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.

Algorithm 2.5 Parent Selection

$h = \lfloor U[0, ..., R + 1) \rfloor$ ▷ column of candidate 1 in population
$i = \lfloor U[0, ..., I + 1) \rfloor$ ▷ row of candidate 1 in population
$j = \lfloor U[0, ..., R + 1) \rfloor$ ▷ column of candidate 2 in population
$k = \lfloor U[0, ..., I + 1) \rfloor$ ▷ row of candidate 2 in population
while $h = i$ **and** $j = k$ **do**
    $j = \lfloor U[0, ..., R + 1) \rfloor$
    $k = \lfloor U[0, ..., I + 1) \rfloor$
if $BF(\rho_{h,i}) > BF(\rho_{j,k})$ **then**
    $P = \rho_{h,j}$
else
    $P = \rho_{j,k}$

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 (Larrøanaga et al., 1999), and are outlined in Algorithm 2.6. The procedures perform on both Seq and Asn of the selected parents.

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

Algorithm 2.6 Crossover Procedures

```
1: One-Point Crossover:
2: c = ⌊U[1, ..., n)⌋
3: for i < c do
4:     q(i) = P_1(i)
5: for c ≤ 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 = ⌊U[1, ..., n)⌋
3: c_2 = ⌊U[1, ..., n)⌋
4: while c_1 = c_2 do
5:     c_2 = ⌊U[1, ..., n)⌋
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 c_1 ≤ 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 c_2 ≤ 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)
```

Perturb operation seen in Algorithm 2.4, 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 2.4.3. 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.

2.5 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 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.

2.5.1 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 2.2 summarises the results of this initial testing, and full results can be seen in Appendix 2.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 2.2: Summary of results of initial formulation testing

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

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.

2.5.2 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 2.3. 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 made can be further seen by observing the number of optimal solutions found by each method in Table 2.3. 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.

Instance		ScA		ScA+DP		ScA+LS	
n	L	%Dev	#Opt	%Dev	#Opt	%Dev	#Opt
5	5	57.07	2	55.55	2	4.23	54
5	10	70.00	0	68.19	0	6.09	39
10	5	84.74	0	83.97	0	10.74	6
10	10	96.98	0	94.17	0	15.07	4
20	5	107.17		104.96		14.62	
20	10	111.63		106.92		18.69	
30	5	111.73		110.01		14.72	
30	10	124.78		119.61		19.11	
50	5	120.54		118.16		13.99	
50	10	129.30		126.22		16.97	
100	5	122.65		120.19		12.02	
100	10	129.11		124.73		11.85	
Average:		105.48		102.72		13.18	

Instance		ScA+LS+DP1		ScA+LS+DP2	
n	L	%Dev	#Opt	%Dev	#Opt
5	5	4.23	54	3.62	62
5	10	6.05	39	5.44	43
10	5	10.74	6	10.28	7
10	10	15.04	4	13.98	4
20	5	14.57		14.24	
20	10	18.65		18.24	
30	5	14.72		14.40	
30	10	19.11		18.78	
50	5	13.99		13.78	
50	10	16.97		16.64	
100	5	12.02		11.87	
100	10	11.85		11.68	
Average:		13.16		12.75	

Table 2.3: Summary of results of basic method testing

2.5.3 Results of iterated local search testing

Table 2.4 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.

The results shown in Table 2.4 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

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 2.4: Results of the best performing ILS tests

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 2.4, were six times the number of jobs on any instance.

2.5.4 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 their hybrid genetic algorithm. This meant we started by considering 40 offspring, five elite solutions, and the diversity weighting given in Equation (2.23), 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 2.5 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 2.5 as tests 14 to 20.

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 2.5: Results of testing on the genetic algorithm

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 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 instances 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 2.6. The tests are numbered as in the previous tables, and the best results highlighted.

	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 2.6: Breakdown of result quality by number of jobs

Table 2.6 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.

2.5.5 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 2.7.

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 2.7: Improvements obtained by introducing a second robot

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 2-8. 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.

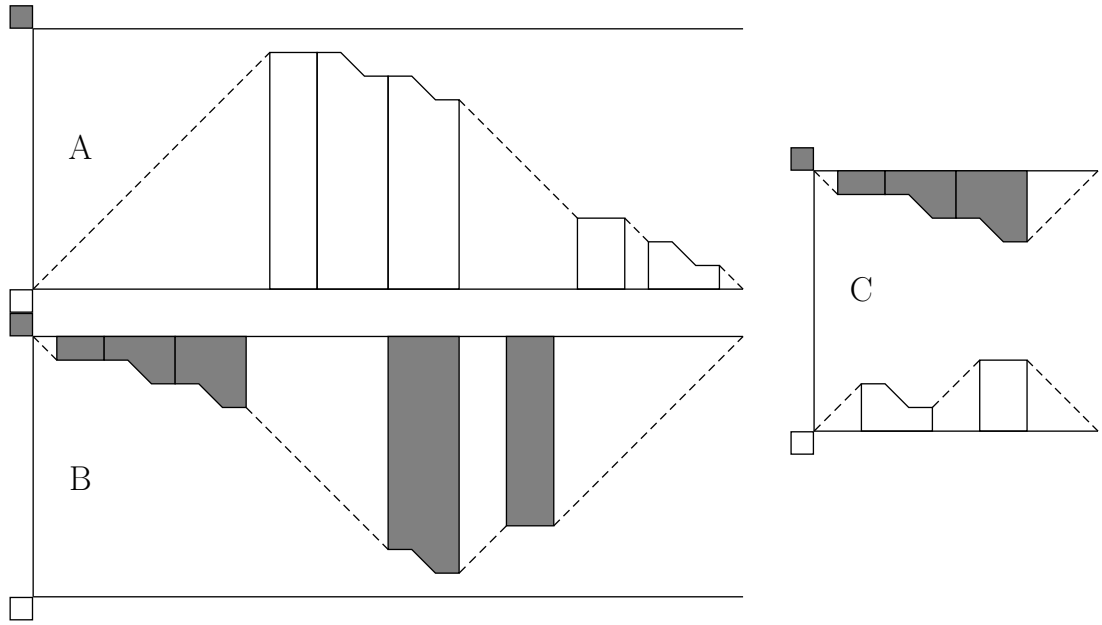

Figure 2-8: Example of the twin robots providing a 60% improvement from the single robot

2.5.6 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 2-9 shows the results of these tests.

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 2.8: Summary of deviations from average makespan

For each instance we obtained ten makespans. We then found the average of these results, and calculated the percentage deviation of each individual makespan from the average. We then grouped these results by the number of jobs in the instance to produce Figure 2-9. 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.

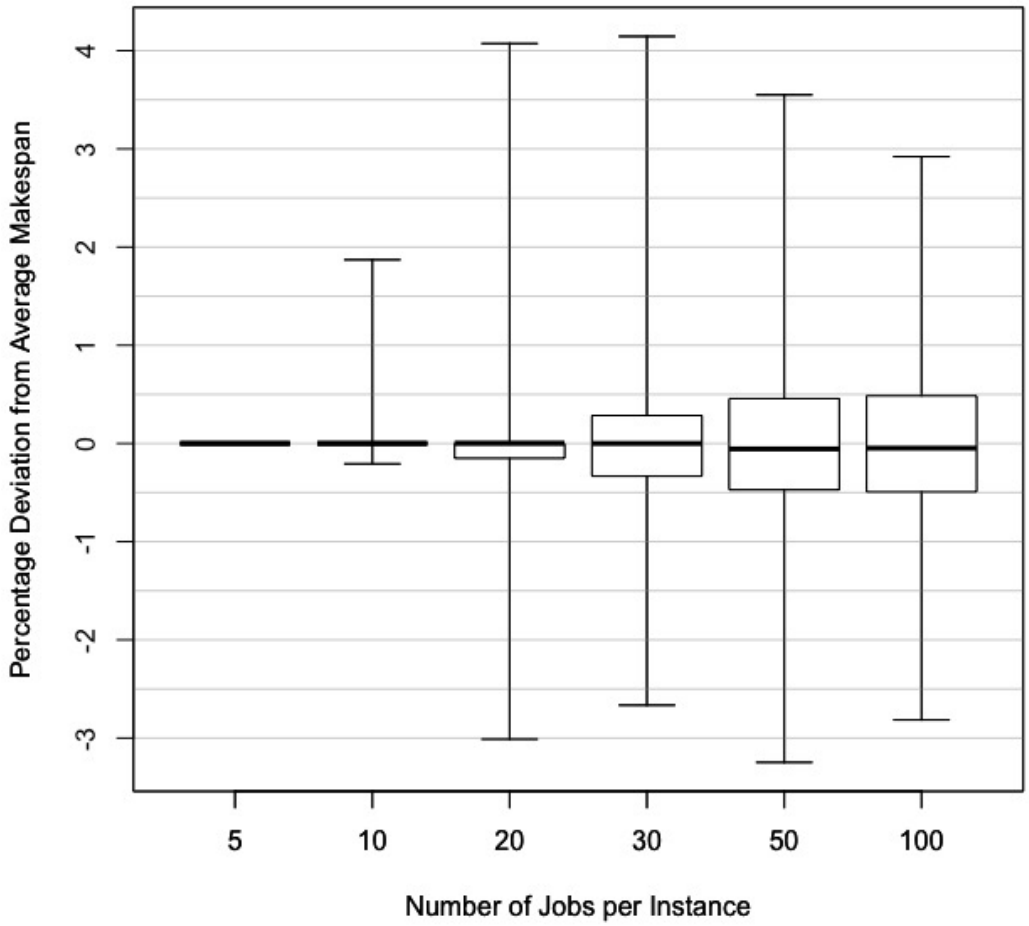

Figure 2-9: Box plots showing the variance in computational results.

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 2.8.

2.6 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.

Bibliography

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

2.A Scheduling coefficient pseudocodes

Algorithm 2.7 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 2.8 Populate γ matrix

1: **for** $i = 1, ..., n$ **do**
2: **for** $j = 1, ..., n$ **do**
3: **if** $(p_i \leq d_i)$ **then**
4: **if** $(p_j > d_j)$ **then**
5: **if** $(d_j > d_i)$ **then**
6: $\gamma^1_{i,j} = 0$
7: **else if** $(p_j \leq d_i)$ **then**
8: $\gamma^1_{i,j} = 2\mu + \tau(2d_i - p_i - p_j + \sigma)$
9: **else**
10: $\gamma^1_{i,j} = \mu + \tau(2d_i - p_i - p_j) + \sigma$
11: **else if** $(p_j > d_i)$ **then**
12: $\gamma^1_{i,j} = 0$
13: **else**
14: $\gamma^1_{i,j} = 2\mu + \tau(2d_i - p_i - p_j + \sigma)$
15: **else if** $(p_j > d_j)$ **then**
16: **if** $(d_j > p_i)$ **then**
17: $\gamma^1_{i,j} = 0$
18: **else if** $(p_j \leq d_i)$ **then**
19: $\gamma^1_{i,j} = 2\mu + \tau(p_i - p_j + \sigma)$
20: **else if** $(d_j \leq d_i$ or $p_j \leq p_i)$ **then**
21: $\gamma^1_{i,j} = \mu + \tau(p_i - p_j + \sigma)$
22: **else**
23: $\gamma^1_{i,j} = \tau(p_i - p_j + \sigma)$
24: **else if** $(p_j > p_i)$ **then**
25: $\gamma^1_{i,j} = 0$
26: **else if** $(p_j \leq d_i)$ **then**
27: $\gamma^1_{i,j} = 2\mu + \tau(p_i - p_j + \sigma)$
28: **else**
29: $\gamma^1_{i,j} = \tau(p_i - p_j + \sigma) + \mu$

2.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 in Section 2.2.2.

Algorithm 2.9 Scheduling Algorithm Part 1

$pW = 0$, $pB = 0$
for $i = 1, ..., n$ **do**
  if $pB = 0$ and $pW = 0$ **then**
    if $Asn_i = white$ **then**
      $S_i = p_i$
      $pW = i$
    else
      $S_i = L + 1 - p_i$
      $pB = i$
  else if $Asn_i = white$ **then**
    if $pW = 0$ **then**
      $S_i = S_{pB}$
      if $S_i < p_i$ **then**
        $S_i = p_i$
      if $S_i > S_{pB} - \gamma^1_{i,pB}$ and $S_i < S_{pB} + \alpha^1_{i,pB}$ **then**
        $S_i = b$
    else if $pB = 0$ **then**
      $S_i = E_{pW} + |p_i - d_{pW}|$
    else
      $S_i = S_{pB}$
      if $S_i < E_{pW} + |d_{pW} - p_i|$ **then**
        $S_i = E_{pW} + |d_{pW} - p_i|$
      if $S_i > S_{pB} - \gamma^1_{i,pB}$ and $S_i < S_{pB} + \alpha^1_{i,pB}$ **then**
        $S_i = b$
      $pW = i$

Algorithm 2.10 Scheduling Algorithm Part 2

```
26:     else if pB = 0 then
27:         S_i = S_pW
28:         if S_i < L + 1 − p_i then
29:             S_i = L + 1 − p_i
30:         if S_i > S_pW − γ²_{i,pW} and S_i < S_pW + α²_{i,pW} then
31:             S_i =
32:     else if pW = 0 then
33:         S_i = E_pB + |p_i − d_pB|
34:     else
35:         S_i = S_pW
36:         if S_i < E_pB + |d_pB − p_i| then
37:             S_i = E_pB + |d_pB − p_i|
38:         if S_i > S_pW − γ²_{i,pW} and S_i < S_pW + α²_{i,pW} then
39:             S_i = b
40:         pB = i
41:         E_i = S_i + 2 + |p_i − d_i|
42: w = max(E_pW + d_pW, E_pB + L + 1 − d_pB)
```

2.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 defined by Equation (2.1).

We then have the following parameters: L, μ, p_i, and d_i as defined in Section 2.2.2, 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:

$$\text{minimise } w \tag{2.25}$$

$$\text{subject to } x_0 = 0 \tag{2.26}$$

$$y_0 = L + 1 \tag{2.27}$$

$$x_t \geq x_{t-1} - 1 \qquad t \in \Gamma \setminus \{0\} \tag{2.28}$$

$$x_t \leq x_{t-1} + 1 \qquad t \in \Gamma \setminus \{0\} \tag{2.29}$$

$$y_t \geq y_{t-1} - 1 \qquad t \in \Gamma \setminus \{0\} \tag{2.30}$$

$$y_t \leq y_{t-1} + 1 \qquad t \in \Gamma \setminus \{0\} \quad (2.31)$$

$$x_t \leq y_t - 1 \qquad t \in \Gamma \setminus \{0\} \quad (2.32)$$

$$\sum_t s_{ti} = 1 \qquad i \in J \quad (2.33)$$

$$\sum_t e_{ti} = 1 \qquad i \in J \quad (2.34)$$

$$w \geq te_{ti} + z_i d_i + (1 - z_i)(L + 1 - d_i) \qquad i \in J; t \in \Gamma \setminus \{0\} \quad (2.35)$$

$$\sum_t te_{it} - \sum_t ts_{it} \geq 2\mu + |d_i - p_i| \qquad i \in J \quad (2.36)$$

$$x_t \leq z_i p_i + (L + 1)(2 - z_i - s_{ti}) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.37)$$

$$x_t \geq p_i(z_i + s_{ti} - 1) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.38)$$

$$y_t \leq (1 - z_i)p_i + (L + 1)(1 + z_i - s_{ti}) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.39)$$

$$y_t \geq p_i(s_{ti} - z_i) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.40)$$

$$x_t \leq z_i d_i + (L + 1)(2 - z_i - e_{ti}) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.41)$$

$$x_t \geq d_i(z_i + e_{ti} - 1) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.42)$$

$$y_t \leq (1 - z_i)d_i + (L + 1)(1 + z_i - e_{ti}) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.43)$$

$$y_t \geq d_i(e_{ti} - z_i) \qquad t \in \Gamma \setminus \{0\}; i \in J \quad (2.44)$$

$$\sum_t te_{tj} - \sum_t ts_{ti} \leq T(l_{ij} + 2 - z_i - z_j) \qquad i, j \in J \quad (2.45)$$

$$\sum_t ts_{tj} - \sum_t te_{ti} \geq T(l_{ij} + z_i + z_j - 3) \qquad i, j \in J \quad (2.46)$$

$$\sum_t te_{tj} - \sum_t ts_{ti} \leq T(l_{ij} + z_i + z_j) \qquad i, j \in J \quad (2.47)$$

$$\sum_t ts_{tj} - \sum_t te_{ti} \geq T(l_{ij} - 1 - z_i - z_j) \qquad i, j \in J \quad (2.48)$$

$$\sum_{t'=t+1}^{t+\mu} x_{t'} - \mu x_t \leq L\mu(2 - s_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \quad (2.49)$$

$$\sum_{t'=t+1}^{t+\mu} x_{t'} - \mu x_t \geq -L(2 - s_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \quad (2.50)$$

$$\sum_{t'=t+1}^{t+\mu} y_{t'} - \mu y_t \leq L\mu(1 - s_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \quad (2.51)$$

$$\sum_{t'=t+1}^{t+\mu} y_{t'} - \mu y_t \geq -L(1 - s_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \quad (2.52)$$

$$\sum_{t'=t}^{t+\mu-1} x_{t'} - \mu x_{t+\mu} \leq L\mu(2 - e_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T - \mu\} \quad (2.53)$$

$$\sum_{t'=t}^{t+\mu-1} x_{t'} - \mu x_{t+\mu} \geq -L(2 - e_{ti} - z_i) \qquad i \in J; t \in \{1, ..., T-\mu\} \quad (2.54)$$

$$\sum_{t'=t}^{t+\mu-1} y_{t'} - \mu y_{t+\mu} \leq L\mu(1 - e_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T-\mu\} \quad (2.55)$$

$$\sum_{t'=t}^{t+\mu-1} y_{t'} - \mu y_{t+\mu} \geq -L(1 - e_{ti} + z_i) \qquad i \in J; t \in \{1, ..., T-\mu\} \quad (2.56)$$

$$l_{ij} + l_{ji} = 1 \qquad i, j \in J; i \neq j \quad (2.57)$$

$$x_t \in \{0, L\} \qquad t \in \Gamma \quad (2.58)$$

$$y_t \in \{1, L+1\} \qquad t \in \Gamma \quad (2.59)$$

$$s_{ti}, e_{ti} \in \{0, 1\} \qquad t \in \Gamma; i \in J \quad (2.60)$$

$$z_i \in \{0, 1\} \qquad i \in J \quad (2.61)$$

$$l_{ij} \in \{0, 1\} \qquad i, j \in J. \quad (2.62)$$

Constraints (2.26) and (2.27) ensure that the robots start a palletising run in their respective depots. Constraints (2.28), (2.29), (2.30), and (2.31) maintain a travel time of $\tau = 1$, and Constraints (2.32) maintain a safety distance $\sigma = 1$. Constraints (2.33) state that a job can only start at one time, and Constraints (2.34) state that a job can only end at one time. Constraints (2.35) 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 (2.36) set the duration of jobs. Constraints (2.37), (2.38), (2.39), and (2.40) 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 (2.41), (2.42), (2.43), and (2.44) 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 (2.45), (2.46), (2.47), and (2.48) ensure that a job cannot be started by a robot until that robot has completed all previous jobs assigned to it. Constraints (2.49), (2.50), (2.51), (2.52), (2.53), (2.54), (2.55), and (2.56) ensure that the process time μ is respected at the start and end of all jobs. Constraints (2.57) state that either job i is before job j, or vice versa; both statements cannot be true. Constraints (2.58), (2.59), (2.60), (2.61), and (2.62) define the domains of the variables.

2.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

Table 2.9: Results of comparative tests of the formulations Part 1

		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
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

Table 2.10: Results of comparative tests of the formulations Part 2

Chapter 3

Pallet Assignment and Job Scheduling in a Twin-Robot System

Statement of authorship

This declaration concerns the article entitled:
Pallet assignment and job scheduling in a twin-robot system.

Publication status: Submitted
Submission details:
Oliver Thomasson, Maria Battarra, Güneş Erdoğan, and Gilbert Laporte
Pallet assignment and job scheduling in a twin-robot system.

Submitted to:
Computers and Operations Research
Submission date:
Thursday 18th February, 2021

Candidates contribution to the paper

- Discovery & exploration of idea (40%)
- Mathematical modelling (90%)
- Heuristic development (40%)
- Coding & testing (95%)
- Initial write-up (90%)
- Editing for publication (40%)

Statement from candidate
This paper reports on original research I conducted during the period of my Higher Degree by Research candidature

Signed:
[redacted]

Date:
Wednesday 7th July, 2021

3.1 Introduction

In this paper we introduce the Twin-Robot Pallet Assignment and Scheduling Problem (TRPASP). This is an original problem which features elements of scheduling, palletisation and assignment of both jobs and locations.

3.1.1 Problem definition

The TRPASP is best introduced using the diagram in Figure 3-1. The first key element we see in this diagram is a set of pickup locations. These are end points of production lines or simply an area in which products are contained ready to be shipped. Opposite the pickup locations are a set of pallets. Each pallet will need to be filled with an *order*. Here we define an order to be a combination of products requested by a customer. A customer can request any quantity and combination of products and, since pallets are considered identical, any order can be placed on any pallet. Therefore one part of the problem is that we must assign orders to pallets. Between the pickup locations and the pallets is a rail. Situated on this rail we find two robots, referred to as twin-robots since they share the rail. We call these robots the white robot and the black robot. The robots are tasked with transferring products from pickup locations to pallets in such a way that each pallet is filled with one order.

The objective of this problem is to transfer all of the products from pickup locations to pallets in the shortest possible time. We call this time the *makespan* as is standard in the literature. We observe that this problem has three major and interrelated parts. The first is assigning orders to pallets. The second element is assigning *jobs* to robots. We define a job as the process of picking up one product from its pickup location and moving it to the pallet to which its associated order has been assigned. The third part is the schedule. Once we have an assignment of orders to pallets and an assignment of jobs to robots we are still left to schedule the timings of the jobs, such that the makespan is minimised.

Additionally, we must ensure that the robots never collide, therefore we set a minimum safety distance which must be respected by the robots at all times. The unit travel time is the time taken for either of the robots to travel from one rail location to an adjacent rail location, and the handling time is the time taken for a robot to either pick a product from a pickup location, or place a product on a pallet once in the correct rail location. We also require that the robots start and end any palletising run at their respective extremity of the rail, named the depots.

This problem applies to automated systems that manage robots. The field of automa-

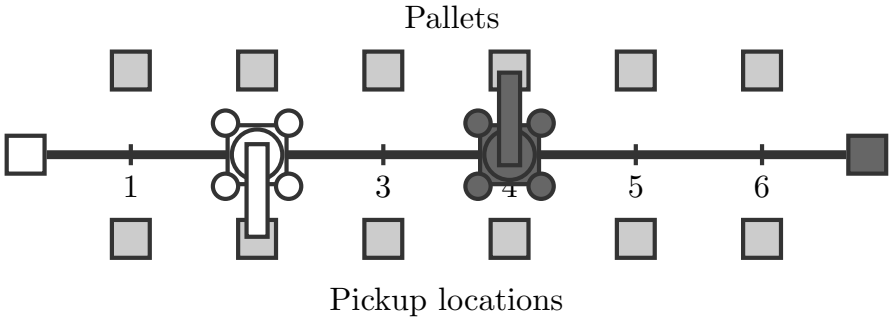

Figure 3-1: An industrial layout in which the TRPASP would occur

tion is of high importance to the modern world, particularly in the area of shipping. Palletisation is an incredibly important component of shipping, with billions of pallets in circulation being used to transfer products (Vanderbilt, 2012). Automation has many advantages in terms of safety of workers. It allows a warehouse or factory environment to have fewer workers in close proximity in any given space.

3.1.2 Literature review

The most relevant literature to the TRPASP is the work done on the scheduling of twin-robots on a line. The closest example is Thomasson et al. (2018) who study the Twin-Robot Palletising Problem (TRPP), of which the TRPASP is a generalisation. The TRPP has mostly the same properties as the TRPASP, but includes the assumption that orders are assigned a priori to a pallet location. The $\mathcal{NP}$-hardness of the TRPASP follows from the fact that the TRPP is $\mathcal{NP}$-hard. Thomasson et al. (2018) present two formulations and multiple heuristic methods for the TRPP, and conclude that a hybrid genetic algorithm is the most effective for finding good solutions within a reasonable time scale. The TRPP is itself a generalisation of the Twin-Robot Scheduling Problem (TRSP) introduced by Erdoğan et al. (2014). In the TRSP pickup locations need not be considered as all products are introduced to the system at the depots. This means that the input of a problem instance contains an assignment of jobs to robots, removing this decision from the problem to be solved. Erdoğan et al. (2014) give two mixed-integer linear programs, and two constructive heuristics for solving the TRSP. A heuristic decomposition procedure for the TRSP was introduced by Boysen et al. (2015), and approximation algorithms were developed by Jaehn and Wiehl (2020) for a version of the TRSP with variable pickup and delivery times.

Similar features of these twin-robot on a line problems can be found in the crane

scheduling literature. Since traditional cranes and the robots from the TRPASP exhibit very similar features we will use the word *operators* to refer to both for this literature review. Boysen et al. (2017) cite the TRSP as an example of a crane scheduling problem with interference, and we can use the classification scheme provided to classify the TRPASP as a [1D, 2, sm; mv^x, pos; C^{max}] problem. This describes the TRPASP as a problem with one dimension, two operators, a safety margin requirement, a constant travel time, specific initial and final positions of the operators, and a makespan objective.

The features of being unidimensional and having a makespan objective are shared by many problems in crane scheduling. No papers classified by Boysen et al. (2017) share all six classifiers with the TRPASP. The most classifiers observed is four, in problems introduced by Liu et al. (2006), Hakam et al. (2012), Guan et al. (2013) and Rodriguez-Molins et al. (2014). The clearest differences between the TRPASP and the most similarly classified problems lie in the processes which must be undertaken by the operators. The problems identified as highly similar by Boysen et al. (2017) all study quay cranes, which are dedicated to transferring containers to and from ships. Usually the containers to be loaded are delivered by trucks or yard cranes to the correct location. The complexity in these problems stems from the stacking of containers in ships, which is a consideration deemed unnecessary in the TRPASP.

Other examples of twin-operator problems in crane scheduling literature include Ng (2005), Gharehgozli et al. (2015) and Briskorn et al. (2016). While the problems introduced by these papers all share the one-dimensional operator movement of the TRPASP, the containers being moved are considered to be movable in either two or three dimensions, making the complexities of placement and stacking a key component of the problems. The TRSP, TRPP, and TRPASP are problems which exist entirely in one dimension.

Some pickup and delivery problems share this feature of being defined in one dimension, and are surveyed by Berbeglia et al. (2007). While most pickup and delivery problems are two-dimensional problems concerned with routing an operator, some are defined on a line. Examples include Atallah and Kosaraju (1988) and Wang et al. (2006). Of particular note is the paper of Anily et al. (1999), which presents the swapping problem on a line, a problem equivalent to the single-robot version of the TRPP. In our literature search we were unable to find a problem which is equivalent to the single robot version of the TRPASP.

From this review we see that certain elements of the TRPASP can be seen in many

areas of literature, but to the best of our knowledge no problem containing all features of the TRPASP has ever been studied.

3.1.3 Paper overview

We have now introduced the problem, and discussed the relevant literature. The remainder of the paper is constructed as follows. Section 3.2 introduces a mathematical model for the TRPASP. Section 3.3 first introduces the algorithmic components which appear in many of our methods, before describing the four heuristic algorithms we developed to solve the TRPASP. In Section 3.4 we present the results of extensive computational tests of these heuristic algorithms, and compare our results with those of the TRPP in order to assess the benefits of considering pallet assignment in the problem. In Section 3.5 we discuss our conclusions.

3.2 Mathematical model

To formalise the definition of the TRPASP we now introduce a mathematical model. The mathematical notation used is as follows:

- Sets and indices
  - $J = \{1, ..., n\}$ is the set of jobs
  - $L = \{1, ..., l\}$ is the set of rail locations
  - o_i is the order containing job $i \in J$
  - p_i is the pickup location of job $i \in J$
- Parameters
  - η is the handling time
  - σ is the safety distance
  - τ is the travel time
  - $\alpha_{i,i',k,k'}$ is the minimum required time difference between the start of jobs i and i' in order to maintain a distance of at least σ between the robots, given job i is performed by the black robot and has delivery location k, job i' is performed by the white robot and has delivery location k', and job i is performed **before** i'

- $\beta_{i,i',k,k'}$ is the minimum required time difference between the start of jobs i and i' in order to maintain a distance of at least σ between the robots, given job i is performed by the black robot and has delivery location k, job i' is performed by the white robot and has delivery location k', and job i is performed **after** i'

- Variables
  - $x_{i,j} = 1$ if and only if job i is the j^{th} job to be performed, and is performed by the white robot. Otherwise $x_{i,j} = 0$
  - $y_{i,j} = 1$ if and only if job i is the j^{th} job to be performed, and is performed by the black robot. Otherwise $y_{i,j} = 0$
  - $z_{i,j} = 1$ if and only if order i is assigned to pallet j. Otherwise $z_{i,j} = 0$
  - t_j is the start time of the job scheduled to be performed j^{th}
  - e_j is the end time of the job scheduled to be performed j^{th}
  - w is the makespan

For this paper we set $\eta = \sigma = \tau = 1$.

An important feature of this model is a pair of matrices $[\alpha]$ and $[\beta]$. These matrices share the names and function of a similar pair of matrices defined by Thomasson et al. (2018) for the TRPP. In that paper, the matrices contain offset parameters such that for the job i assigned to the white robot and the job i' assigned to the black robot, the minimum allowable difference between starting times is either $\alpha_{i,i'}$ if i comes before i' or $\beta_{i,i'}$ if i comes after i'. In the TRPP these parameters could be precomputed, as the pickup and delivery locations of every job are fixed in the instance inputs. Unfortunately we cannot do the same for the TRPASP. Instead we have two choices: we can use the same parameter calculations as were used for $\alpha_{i,i'}$ and $\beta_{i,i'}$, and calculate values once a delivery location is known; or we can expand these matrices with additional indices to $\alpha_{i,i',k,k'}$ and $\beta_{i,i',k,k'}$ where k is the delivery location of i and k' is the delivery location of i'. While the precomputation of these four-dimensional matrices is not preferred in heuristic methods, for our model they can be precomputed without affecting performance. The model is as follows:

$$\text{minimise } w \tag{3.1}$$

$$\text{subject to } \sum_{i \in L} z_{i,j} = 1 \qquad j \in L \tag{3.2}$$

$$\sum_{j \in L} z_{i,j} = 1 \qquad i \in L \tag{3.3}$$

$$\sum_{i \in J} (x_{i,j} + y_{i,j}) = 1 \qquad j \in J \tag{3.4}$$

$$\sum_{j \in J} (x_{i,j} + y_{i,j}) = 1 \qquad i \in J \tag{3.5}$$

$$t_1 \geq \textstyle\sum_{i \in J} \tau \cdot p_i x_{i,1} + \sum_{i \in J} \tau(l + 1 - p_i) y_{i,1} \tag{3.6}$$

$$e_j \geq t_j + (2\eta + \tau|p_i - k|)(x_{i,j} + y_{i,j} + z_{o_i,k} - 1) \qquad i, j \in J; k \in L \tag{3.7}$$

$$t_j \geq e_{j'} + \tau|p_i - k| z_{o_{i'},k} - M(2 - x_{i,j} - x_{i',j'}) \qquad i, i', j \in J; 1 \leq j' < j; k \in L \tag{3.8}$$

$$t_j \geq e_{j'} + \tau|p_i - k| z_{o_{i'},k} - M(2 - y_{i,j} - y_{i',j'}) \qquad i, i', j \in J; 1 \leq j' < j; k \in L \tag{3.9}$$

$$t_{j'} \geq t_j + \beta_{i,i',k,k'} + M(x_{i,j} + y_{i',j'} + z_{o_i,k} + z_{o_{i'},k'} - 4) \qquad i, i', j' \in J; 1 \leq j < j'; k, k' \in L \tag{3.10}$$

$$t_j \geq t_{j'} + \alpha_{i,i',k,k'} + M(x_{i,j} + y_{i',j'} + z_{o_i,k} + z_{o_{i'},k'} - 4) \qquad i, i', j \in J; 1 \leq j' < j; k, k' \in L \tag{3.11}$$

$$w \geq e_j + \tau \cdot k \cdot z_{o_i,k} - M(1 - x_{i,j}) \qquad i, j \in J; k \in L \tag{3.12}$$

$$w \geq e_j + \tau(l + 1 - k) z_{o_i,k} - M(1 - y_{i,j}) \qquad i, j \in J; k \in L \tag{3.13}$$

$$x_{i,j} \in \{0, 1\} \qquad i, j \in J \tag{3.14}$$

$$y_{i,j} \in \{0, 1\} \qquad i, j \in J \tag{3.15}$$

$$z_{i,j} \in \{0, 1\} \qquad i, j \in L \tag{3.16}$$

$$t_j \geq 0 \qquad j \in J \tag{3.17}$$

$$e_j \geq 0 \qquad j \in J \tag{3.18}$$

$$w \geq 0. \tag{3.19}$$

Constraints (3.2) state that every pallet is assigned exactly one order. Constraints

(3.3) ensure that every order is assigned to exactly one pallet. Constraints (3.4) ensure that every position in the job schedule contains a job performed by exactly one of the robots. Similarly, Constraints (3.5) state that every job is scheduled to exactly one position in the job schedule, and performed by exactly one robot. Constraint (3.6) sets the start time of the first job in the schedule. Constraints (3.7) set the end time of all jobs. Constraints (3.8) ensure that if a job i is assigned to the white robot, then it cannot have a start time less than the end time of the jobs previously completed by the white robot, plus the travel time to p_i. Constraints (3.9) state the same as Constraints (3.8), but for the black robot. Constraints (3.10) enforce the use of the β matrix to ensure that the start time of a job for the white robot is suitably late to ensure that minimum safe distance is maintained from the black robot. Constraints (3.11) enforce the use of the α matrix to ensure that the start time of a job for the black robot is suitably late to ensure that minimum safe distance is maintained from the white robot. Constraints (3.12) and (3.13) ensure that the makespan cannot be less than the end time of any job, plus the time taken to return to the depot. Constraints (3.14) to (3.19) set the domains of the variables.

3.3 Algorithms for the TRPASP

In this section we present the algorithms we developed to solve the TRPASP. The four heuristics we present are:

1. A Hungarian algorithm to construct promising candidate solutions before utilising existing algorithms to solve the problem in full.
2. A Variable Neighbourhood Search (VNS) algorithm.
3. A hybrid algorithm consisting of Local Search (LS) for pallet locations and parallel Genetic Algorithm (GA) for consequent scheduling decisions.
4. A Genetic Algorithm that encodes both pallet location and scheduling decisions.

A solution for the TRPASP is characterised by the vectors $\mathbf{a} = (a_i)$, $\mathbf{d} = (d_i)$, and $\mathbf{s} = (s_i)$, where a_i is the robot assignment of job i; d_i is the destination pallet location of job i, also called the delivery location of job i; and s_i is the position of job i in the job schedule. These vectors can be evaluated to calculate the makespan.

A key value used in all methods is the *in-job distance* ($\hat{w}$) defined as

$$\hat{w} = \sum_{i=0}^{n} (2 + |p_i - d_i|).$$

Our algorithms rely on an initial assumption that in-job distance and optimal makespan are positively correlated, meaning that we can use the in-job distance as a proxy for the optimal makespan.

To check that this assumption is well founded we first ran computational tests to assess the correlation. In these tests we generated every possible pallet assignment for a given instance, before using the algorithm of Thomasson et al. (2018) for the TRPP to find a makespan. We then picked the minimum makespan among these as the best known makespan value w^*. Since the number of possible pallet assignments for any given instance is $l!$, we produced a set of instances with $l = 5$. We chose four values of the number of jobs n: 20, 30, 50 and 100, and for each value we produced 10 instances, giving us a set of 40 instances. We then calculated the correlation coefficient between w^* and $\hat{w}$ for these instances. The average correlation coefficient across all instances was 0.8490. Figure 3-2 shows boxplots for the correlation values, arranged in rows by instance size.

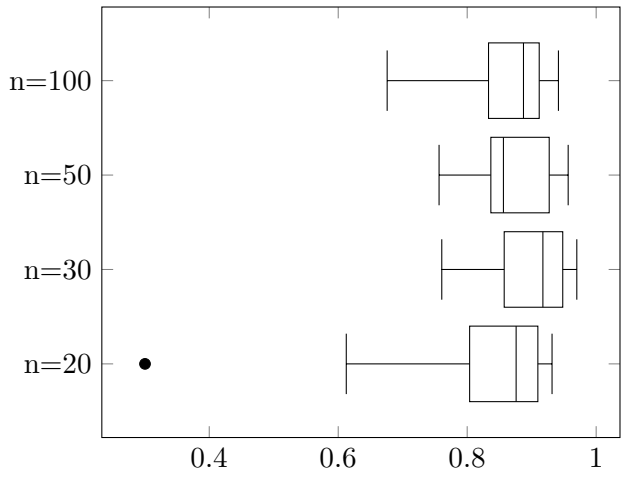

Figure 3-2: Boxplot of correlation between w^* and $\hat{w}$.

We see that the initial assumption was well founded and we can use in-job distance as a proxy for the optimal makespan. This is valuable for our algorithm development, as computing $\hat{w}$ has a complexity of $\mathcal{O}(n^2)$, while finding optimal makespan from a pallet assignment is $\mathcal{NP}$-hard, as proven by Thomasson et al. (2018). In the instance subset with $n = 20$ we can observe one outlier, with a correlation value of 0.3004. Upon closer

inspection of the instance, we found that this is due to the jobs being generated in such a way that pallet location has little effect on total in-job distance. We calculated the standard deviation of $\hat{w}$ for each instance, since each instance produces 5! values of $\hat{w}$. The outlier had a standard devation of 1.67 in $\hat{w}$, compared to an average of 4.03 for the subset of instances with $n = 20$, and 5.91 for the full set of instances.

3.3.1 Shared features

Some subroutines are implemented in more than one of the algorithms presented in this section, so for brevity we present them here first. Some of these subroutines were developed by Thomasson et al. (2018) for the TRPP, and some were developed for the TRPASP.

Constant-time heuristic for makespan calculation

This heuristic was developed for the TRPP, and is used to calculate the makespan, given solution elements **a**, **d**, and **s**. The heuristic calculates the earliest possible start time for each job, based on a job sequence **s**. Subsequently, the end time of each job is also calculated. The calculation of start times relies on the matrices $\alpha_{i,i'}$ and $\beta_{i,i'}$ discussed in Section 3.2. We refer to this subroutine as TRP0.

Hybrid genetic algorithm for TRPP

As discussed in Section 3.1.2, Thomasson et al. (2018) found that a hybrid genetic algorithm was the best heuristic for the TRPP on realistically sized instances. We use this algorithm in some of our methods to evaluate solutions in which only a pallet assignment **d** is defined. We refer to this subroutine as TRP1.

Local search moves

Many of the algorithms use local search moves. These are used in both traditional local search procedures and in perturbation moves for finding new solutions. Here are the moves we use in our heuristics, together with their computational complexity. We recall that n is the number of jobs and l is the number of rail locations.

1. **Change robot assignment ($\mathcal{O}(n)$):** for a job i, the robot assignment a_i is changed from white to black, or from black to white as appropriate.
2. **Swap robot assignment ($\mathcal{O}(n^2)$):** for two selected jobs i, j with $a_i \neq a_j$, the values of a_i and a_j are swapped.

3. **Swap job sequence ($\mathcal{O}(n^2)$):** for two selected jobs, their positions in the sequence are swapped.

4. **Relocate a job ($\mathcal{O}(n^3)$):** the selected job is placed in a new position in the job sequence.

5. **Relocate a job and change robot assignment ($\mathcal{O}(n^3)$):** as above, but the robot assignment of the job is also changed.

6. **Reverse a job sequence ($\mathcal{O}(n^3)$):** a selected subsequence of jobs in the job sequence is reversed. This is only applicable to subsequences of length four or more, otherwise the move is equivalent to move three.

7. **Relocate a job sequence ($\mathcal{O}(n^4)$):** a selected subsequence of jobs in the job sequence is moved to a new position in the job sequence.

8. **Swap pallet assignment ($\mathcal{O}(l^2)$):** two orders are swapped in the pallet assignment.

9. **Relocate pallet ($\mathcal{O}(l^3)$):** one order is picked and moved to a new pallet. This will move some other orders, but all orders other than the selected order will maintain their relative assignments.

Hungarian algorithm for pallet assignment selection

Based on the strong correlation between $\hat{w}$ and w^* shown at the start of this section, we developed a fast constructive algorithm to find the permutations of $\mathbf{d}$ with the best $\hat{w}$ values. The algorithm first builds a matrix of travel distances $\delta_{a,b}$ calculated as

$$\delta_{a,b} = \sum_{i \in J : o_i = a} |p_i - b| \qquad\qquad a, b \in L \tag{3.20}$$

With this matrix of $\delta_{a,b}$ values we can then solve a matching problem to assign orders to pallets, and use a polynomial time algorithm (e.g., the Hungarian algorithm of Kuhn (1955)) in order to find a pallet assignment with minimum $\hat{w}$. To produce more than one initial solution we work through each $\delta_{a,b}$ in the solution, penalising one at a time by ignoring the value calculated by Equation (3.20), instead giving the value 10,000, before running the Hungarian algorithm to obtain new good pallet assignments. This procedure gives us up to $l + 1$ initial pallet assignments, but can be extended to find more as required. We label this algorithm as H_0.

3.3.2 Hungarian algorithm with scheduling

The first improvement algorithm we developed (H_1) takes the $l+1$ assignments of $\mathbf{d}$ produced by H_0, and passes them to TRP1. To improve computational testing time we run TRP1 on the initial solutions in parallel, but any reported CPU times in Section 3.4 are equivalent to running in series. While this is a very simple algorithm, it gives us a base to which we can compare the rest of our algorithms.

3.3.3 Variable neighbourhood search

Our next algorithm (H_2) is a variant of Variable Neighbourhood Search without perturbation moves. It consists of two neighbourhoods, one for $\mathbf{a}$ and $\mathbf{s}$, and one for $\mathbf{d}$. Each neighbourhood has its own local search algorithm. The first is named LS_1 and consists of moves one to seven from Section 3.3.1. The second is named LS_2 and consists of moves eight and nine from Section 3.3.1. An outline of H_2 can be seen in Algorithm 3.1.

Algorithm 3.1 Pseudocode for H_2

Run H_0 to produce initial $\mathbf{d}$
Randomly generate values for $\mathbf{a}$ and $\mathbf{s}$
set $\mathbf{a}' = \mathbf{a}$, $\mathbf{d}' = \mathbf{d}$, and $\mathbf{s}' = \mathbf{s}$
do
    if $TRP0(\mathbf{a}, \mathbf{d}, \mathbf{s}) > TRP0(\mathbf{a}', \mathbf{d}', \mathbf{s}')$ **then**
        set $\mathbf{a} = \mathbf{a}'$, $\mathbf{d} = \mathbf{d}'$, and $\mathbf{s} = \mathbf{s}'$
    $(\mathbf{a}', \mathbf{d}', \mathbf{s}') \leftarrow LS_1(\mathbf{a}, \mathbf{d}, \mathbf{s})$
    $(\mathbf{a}', \mathbf{d}', \mathbf{s}') \leftarrow LS_2(\mathbf{a}', \mathbf{d}', \mathbf{s}')$
while $TRP0(\mathbf{a}', \mathbf{d}', \mathbf{s}') < TRP0(\mathbf{a}, \mathbf{d}, \mathbf{s})$
return solution makespan

The initial solution makes use of H_0 for the initial $\mathbf{d}$, while initial $\mathbf{a}$ and $\mathbf{s}$ are randomly generated. At each iteration both local searches look for improvement to the solution in its neighbourhood. At the end of an iteration, if no improvement has been found the algorithm terminates, otherwise the next iteration begins. The simplicity of H_2 means that the computation times are very small.

3.3.4 Local search based parallel metaheuristic

Figure 3-3 depicts the overall structure of our next algorithm (H_3) in a flowchart.

The key component in H_3 is RLS, which is a local search with a reduced set of possible moves used for finding promising candidate $\mathbf{d}$ vectors to pass to TRP1 in the next iteration. For each solution $(\mathbf{a}, \mathbf{d}, \mathbf{s})$ the steps of RLS are:

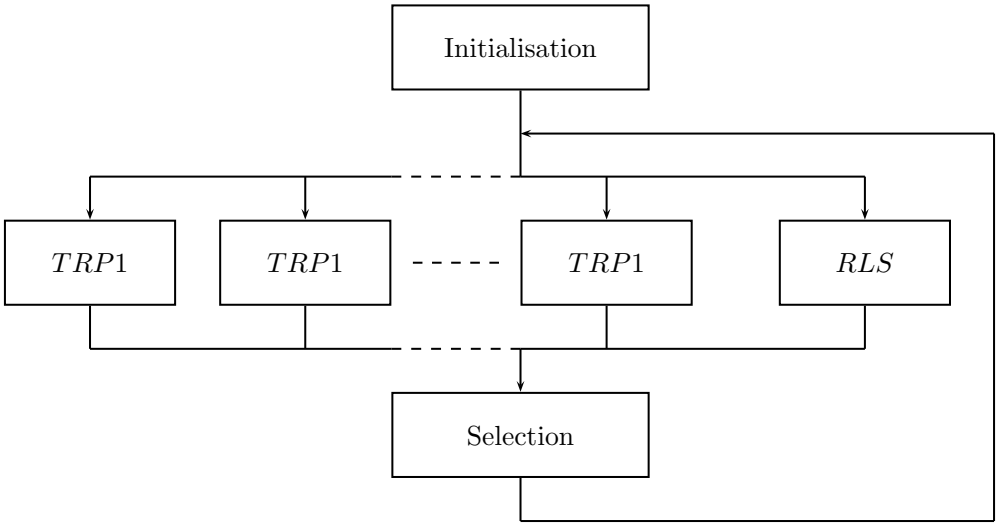

Figure 3-3: H_3 flowchart illustrating the parallel process

1. Produce a copy of $(\mathbf{a}, \mathbf{d}, \mathbf{s})$, to ensure the following steps are non-destructive.
2. Perturb $\mathbf{d}$ until an unexplored $\mathbf{d}$ is found.
3. Identify *inefficient* jobs in $(\mathbf{a}, \mathbf{d}, \mathbf{s})$.
4. Perform moves from LS_1 affecting only the *inefficient* jobs, using TRP0 for solution evaluation.
5. Find $(\mathbf{a_i}, \mathbf{d_i}, \mathbf{s_i})$ with worst w in the set of solutions.
6. If $TRP0(\mathbf{a}, \mathbf{d}, \mathbf{s}) < TRP0(\mathbf{a_i}, \mathbf{d_i}, \mathbf{s_i})$, $(\mathbf{a_i}, \mathbf{d_i}, \mathbf{s_i}) = (\mathbf{a}, \mathbf{d}, \mathbf{s})$.

In step (2) a simple perturbation is performed on $\mathbf{d}$ by a single randomly selected local search move from LS_2. If the new assignment has already been investigated at some point in the algorithm we perturb again. If five perturbations do not yield a non-investigated $\mathbf{d}$, the pallet assignment is randomly shuffled until one is found. We can identify a pallet assignment as investigated or non-investigated due to a matrix containing every pallet assignment that the algorithm has explored.

In step (3) we quantify which jobs are *inefficient* by considering the *unloaded time*. For a job i, the unloaded time μ_i is defined as

$$\mu_i = (t_i - e_{i-1}) + (t_{i+1} - e_i),$$

where jobs $i-1$ and $i+1$ are the jobs performed directly before and after job i by the same robot. The jobs with the greatest μ_i values are labelled as *inefficient* jobs.

Initialisation defines the solution objects to be passed to the blocks of TRP1 or *RLS*. We produce three sets of solutions, B_1, B_2 and B_3. The number of solutions in each set is equal to the number of computation threads minus one. B_1 contains the solutions to be evaluated by TRP1, B_2 contains the solutions to be explored by *RLS*, and B_3 is used to store the best known solutions. The initialisation populates B_1 with pallet assignments produced by H_0, and randomly generates **a** and **s** vectors. At the first point of passing to the parallel section B_1 is identical to B_2, and B_3 is empty.

Selection is used for solution management to ensure the correct solutions are being passed, stored, or discarded as appropriate. At the termination of the parallel section, selection receives B_1 and B_2. The procedure compares the solutions in B_1 to the solutions in B_3, keeping the solutions with the best makespans on the condition that every solution in B_3 must have a unique **d** vector. Once this update of B_3 is complete, the solutions in B_2 are copied into B_1 for evaluation by TRP1, and the next iteration of the parallel section begins, with *RLS* continuing to search for better **d** vectors, while TRP1 evaluates the best **d** vectors found so far.

3.3.5 Hybrid genetic algorithm

Our final algorithm (H_4) is a hybrid genetic algorithm. Unlike H_1 or H_3 it incorporates pallet assignments into the hybrid genetic structure instead of using TRP1. An outline of the structure of H_4 is provided below.

We first produce a number of solutions with pallet assignments **d** given by H_0, and the other two chromosomes (**a** and **s**) randomly generated. From each of these initial solutions we produce improved solutions by performing randomly selected local search moves from Section 3.3.1 until no more improvement is possible. If no improvement can be found, the new solution is randomised to fill the initial population.

We calculated diversity for each population member $(\mathbf{a}, \mathbf{d}, \mathbf{s})$ by comparing each element of $(\mathbf{a}, \mathbf{d}, \mathbf{s})$ to the corresponding element in every other population member. If the elements are different we add one to $(\mathbf{a}, \mathbf{d}, \mathbf{s})$'s diversity value. We then calculate a value defined by Vidal et al. (2012) as the *biased fitness*, but use a slightly different formula to calculate this value. Biased fitness is given by $bf_i = mr_i + W \cdot dr_i$ where bf_i

Algorithm 3.2 Hybrid genetic algorithm (H_4)

```
1: generate initial population
2: while time limit not met do
3:     calculate population diversity
4:     for each offspring do
5:         select two parents from population
6:         perform crossover to produce an offspring
7:         if mutation occurs then
8:             mutate offspring
9:         perform local search on offspring
10:    calculate offspring diversity
11:    keep or replace population members as appropriate
12: return minimum makespan within population
```

is the biased fitness value of i, mr_i is the makespan rank of i, dr_i is the diversity rank of i, and W is a parameter to control the weighting of diversity in the calculation.

Parents are selected at random from the population. To select one parent, we first randomly identify two population members. The member with the larger bf_i value is chosen to be a parent. This is repeated for the second parent.

Once parents are selected we perform two-point crossover. This crossover procedure is performed twice. The first crossover is performed on pallet assignment, and the second crosses both schedule and assignment, since these chromosomes are better able to be moved together. A detailed pseudocode of the procedure can be found in 3.A.

Mutation occurs with some probability, and consists of a single, randomly chosen local search move on the offspring with forced acceptance.

Calculating offspring diversity is the same as the diversity calculation for the population. Each offspring is given a diversity score by comparing it to each member of the population.

A few best solutions in the population are labelled as *elite* solutions, and cannot be removed in the current iteration. All other population members are then pooled with the offspring, and the best member of this pool is added to the new population until it is full. At this point all other solutions are discarded and a new iteration begins with the new population.

Once the time limit is met, the population member with the best makespan is identified, and this makespan is returned as the best known solution to the TRPASP.

3.4 Computational results

To test our metaheuristics we used a set of 100 generated instances. We produced instances for $l \in \{10, 20\}$ and $n \in \{20, 30, 50, 75, 100\}$. For each pairing of an l and n value we produced ten instances. The full set of instances is available upon request. We ran each algorithm 10 times on each instance to account for random elements in the algorithms.

The algorithms which contain TRP1 use the tuning parameters and termination conditions used by Thomasson et al. (2018). All algorithms were coded in C++ and those with parallel elements used openMP. All tests were run on Balena HPC at the University of Bath. Further details on Balena HPC can be found at `https://www.bath.ac.uk/bucs/services/hpc/facilities/`.

The algorithms were tested with the following time limits:

- **H_1:** no time limit for the Hungarian algorithm, since its CPU time is negligible. TRP1 given instance dependent time limit of $0.2n$ minutes.
- **H_2:** no time limit set.
- **H_3:** one hour.
- **H_4:** instance dependent limit of $0.2n$ minutes.

Table 3.1 provides the following results for each algorithm:

- **nBest:** The number of times the algorithm was able to find the best known solution.
- **avgDev:** The average deviation of the makespan found by the algorithm, from the best known makespan.
- **avgCPU:** The average CPU time in seconds. For H_1 and H_3 these are CPU times ignoring the parallel implementation.

From Table 3.1 we can clearly see H_4 outperforms every other method on all instance sizes. H_1 is the weakest of the algorithms, as we would expect given its simplicity of pallet assignment selection. Comparing H_2 and H_3 is interesting as we see H_3 performing better on all instances with $l = 10$, and H_2 performing better on all instances with $l = 20$. The comparison of H_2 and H_3 leans more positively towards H_2, given its added value in computation time when compared to H_3. The computation time of H_2 is at most 2% of the one hour we allow H_3.

Instance size		H_1			H_2			H_3			H_4		
n	l	nBest	avgDev	avgCPU	nBest	avgDev	avgCPU	nBest	avgDev	avgCPU	nBest	avgDev	avgCPU
20	10	0	38.4	2642.4	0	25.4	0.04	9	18.2	39872.2	29	3.1	240.0
30	10	0	28.0	3842.1	0	20.1	0.05	6	10.7	54378.1	11	3.8	240.1
50	10	0	18.2	3970.4	0	14.4	0.22	7	6.2	40360.0	15	2.5	360.1
75	10	1	17.6	5773.1	0	14.7	0.24	3	7.1	55062.7	11	3.5	360.2
100	10	0	15.8	6663.5	1	10.1	2.37	2	7.7	40823.8	8	2.3	600.6
all	10	1	23.6	4578.3	1	16.9	0.58	27	10.0	46099.4	74	3.0	360.2
20	20	0	74.0	9689.3	3	27.7	2.43	0	49.3	55497.7	14	5.5	600.7
30	20	0	41.5	10176.1	0	17.4	16.30	0	31.1	40321.2	14	4.0	901.9
50	20	0	26.5	14712.6	0	13.0	18.27	0	23.4	54771.2	13	2.8	902.0
75	20	0	27.9	14122.4	3	8.8	65.52	0	19.1	42274.6	11	2.6	1204.8
100	20	0	24.8	20503.8	2	8.1	73.18	0	17.7	56931.3	9	3.0	1204.9
all	20	0	38.9	13840.8	8	15.0	35.14	0	28.1	49959.2	61	3.6	962.8
all	all	1	31.3	9209.6	9	16.0	17.86	27	19.0	48029.3	135	3.3	661.5

Table 3.1: Makespan results

3.4.1 Benefits of pallet assignment

Since the novelty of the TRPASP is the inclusion of pallet assignment in the problem, we wish to show that this consideration yields better results. To do so, we take our 100 instance set, and run TRP1 on them by setting $d_i = o_i$ in the input parameters. We then compare the best result of TRP1 to our results from the algorithms developed in this paper. Table 3.2 summarises these results. The column with heading *Best* compares the best result from all of our computational testing to the result of TRP1. The column with heading H_4 compares the best makespan found by algorithm H_4 to the result of TRP1. Once again we present results broken down by n and l values of the instances.

n	l	Best (%)	H_4 (%)
20	10	23.42	23.42
30	10	20.87	20.17
50	10	9.55	9.44
75	10	4.48	3.71
100	10	4.80	4.52
all	10	12.62	12.25
20	20	43.81	43.58
30	20	28.52	28.52
50	20	25.43	25.43
75	20	17.65	17.62
100	20	17.48	17.28
all	20	26.58	26.49
all	all	19.60	19.37

Table 3.2: Average percentage improvement made to makespan by considering pallet assignment

In all instance subsets, pallet assignment has a positive effect on makespan. We see the largest improvement in instances with few jobs and a long rail. As the number of jobs increases there is a reduction in the observed improvement, but the improvements for $n = 75$ and $n = 100$ are very similar. We also see that a longer rail leads to much greater improvement when considering pallet assignment. The potential for improvement through pallet assignment is therefore greatest for instances with low n and high l.

3.5 Conclusions

We have introduced the Twin-Robot Pallet Assignment and Scheduling Problem (TRPASP), a new problem with applications in industry. We have presented four heuristic algorithms capable of solving the TRPASP, then compared their performance to conclude that a hybrid genetic algorithm yields the best results. We also compared our results to the TRPP (Thomasson et al., 2018), a similar problem which does not include consideration of pallet assignment. In this comparison we showed that pallet assignment within the problem leads to improvement in overall makespan, with the best improvements found in instances with fewer jobs and a longer rail.

Acknowledgements

This research made use of the Balena High Performance Computing (HPC) Service at the University of Bath.

Bibliography

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

Atallah, M. J. and S. R. Kosaraju (1988). Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing 17*(5), 849–869.

Berbeglia, G., J.-F. Cordeau, I. Gribkovskaia, and G. Laporte (2007). Static pickup and delivery problems: a classification scheme and survey. *TOP 15*(1), 1–31.

Boysen, N., D. Briskorn, and S. Emde (2015). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics 62*(1), 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*(1), 343–357.

Briskorn, D., S. Emde, and N. Boysen (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics 211*, 40–57.

Erdoğan, G., M. Battarra, and G. Laporte (2014). Scheduling twin robots on a line. *Naval Research Logistics 61*(2), 119–130.

Gharehgozli, A. H., G. Laporte, Y. Yu, and R. de Koster (2015). Scheduling twin yard cranes in a container block. *Transportation Science 49*(3), 686–705.

Guan, Y., K.-H. Yang, and Z. Zhou (2013). The crane scheduling problem: models and solution approaches. *Annals of Operations Research 203*(1), 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.

Jaehn, F. and A. Wiehl (2020). Approximation algorithms for the twin robot scheduling problem. *Journal of Scheduling 23*(1), 117–133.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*(1-2), 83–97.

Liu, J., Y.-w. Wan, and L. Wang (2006). Quay crane scheduling at container terminals to minimize the maximum relative tardiness of vessel departures. *Naval Research Logistics 53*(1), 60–74.

Ng, W. (2005). Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research 164*(1), 64–78.

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*(2), 273–290.

Thomasson, O., M. Battarra, G. Erdoğan, and G. Laporte (2018). Scheduling twin robots in a palletising problem. *International Journal of Production Research 56*(1-2), 518–542.

Vanderbilt, T. (2012). The single most important object in the global economy. https://slate.com/business/2012/08/pallets-the-single-most-important-object-in-the-global-economy.html. Accessed: 2020-11-02.

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, F., A. Lim, and Z. Xu (2006). The one-commodity pickup and delivery travelling salesman problem on a path or a tree. *Networks 48*(1), 24–35.

Appendix

3.A Pseudocode for two-point crossover in hybrid genetic algorithm

Algorithm 3.3 gives details of the two-point crossover used in H_4. c_1 and c_2 are the crossing points, and $q[C_x(i)]$ is the ith element of chromosome x for offspring q

Algorithm 3.3 Offspring production by two-point crossover

```
1: c_1 = U_d[1, ..., n − 1]
2: c_2 = U_d[c_1 + 1, ..., n]
3: count = 0
4: for i < c_1 do
5:     q[C_x(i)] = P_1[C_x(i)]
6:     count = count + 1
7: for c_1 ≤ i < 2n do
8:     j = i mod n
9:     inherited = false
10:    for k < n do
11:        if q[C_x(k)] = P_2[C_x(j)] then
12:            inherited = true
13:    if inherited = false then
14:        q[C_x(i)] = P_2[C_x(j)]
15:        count = count + 1
16:    if count = c_2 then
17:        break
18: for c_2 ≤ i < 2n do
19:    j = i mod n
20:    inherited = false
21:    for k < n do
22:        if q[C_x(k)] = P_1[C_x(j)] then
23:            inherited = true
24:    if inherited = false then
25:        q[C_x(i)] = P_1[C_x(j)]
26:        count = count + 1
27:    if count = n then
28:        break
```

3.B Detailed computational testing results

The following tables provide the makespan found by each algorithm in each of the ten runs. The makespans which are the best known values are presented in bold face. We also provide the average CPU time in seconds for the ten runs of each instance.

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
20	10	0	74	74	74	74	74	74	74	74	74	74	2642.6
20	10	1	73	73	74	73	73	73	74	74	74	73	2642.3
20	10	2	95	94	94	94	94	94	94	94	94	94	2642.4
20	10	3	56	56	56	58	56	56	56	56	56	56	2642.3
20	10	4	73	72	75	74	72	74	73	75	75	73	2642.2
20	10	5	70	71	71	70	70	70	70	71	70	70	2642.2
20	10	6	70	70	70	70	70	70	70	70	70	70	2642.2
20	10	7	68	67	67	66	66	67	67	67	67	67	2642.3
20	10	8	60	60	60	59	61	60	59	60	60	60	2642.5
20	10	9	60	60	59	59	59	59	59	59	59	59	2642.5
20	20	0	128	128	128	132	126	127	131	126	128	128	3842.3
20	20	1	84	84	87	88	85	86	88	86	84	88	3841.6
20	20	2	111	112	113	112	110	111	112	113	112	111	3842.0
20	20	3	98	98	98	99	98	98	99	98	99	98	3841.9
20	20	4	98	98	98	100	99	96	100	98	100	103	3842.2
20	20	5	120	121	120	120	120	121	122	120	121	121	3841.9
20	20	6	101	100	100	100	100	100	100	100	100	100	3842.3
20	20	7	101	104	105	103	103	101	103	101	103	103	3842.1
20	20	8	114	114	114	116	114	114	114	118	115	114	3842.2
20	20	9	127	127	128	128	129	128	129	130	129	128	3842.0
30	10	0	92	90	90	90	90	90	90	90	91	88	3969.6
30	10	1	113	113	113	112	112	111	112	111	110	111	3970.9
30	10	2	100	100	100	101	101	101	101	100	101	101	3970.3
30	10	3	84	84	84	84	84	84	84	84	85	84	3969.7
30	10	4	99	98	98	99	97	99	98	98	99	97	3970.6
30	10	5	93	93	92	92	92	92	93	92	92	93	3970.2
30	10	6	126	125	124	126	124	126	126	126	125	125	3970.3
30	10	7	101	100	101	101	100	100	101	101	100	101	3971.1
30	10	8	103	102	103	104	104	104	105	103	103	104	3971.8
30	10	9	88	88	85	87	89	88	88	88	88	86	3969.7

Table 3.3: H_1 Results for instances 20.10.0 to 30.10.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
30	20	0	162	164	164	164	165	164	166	166	164	164	5774.7
30	20	1	160	160	159	161	160	158	158	158	160	156	5773.9
30	20	2	146	144	148	146	146	146	148	148	146	146	5773.4
30	20	3	142	144	144	145	140	144	146	144	146	144	5773.6
30	20	4	160	160	157	159	155	156	158	158	160	152	5773.1
30	20	5	154	155	152	153	157	151	153	153	152	154	5772.9
30	20	6	180	180	174	181	180	180	180	175	180	178	5773.0
30	20	7	170	170	168	166	170	170	168	170	169	168	5772.2
30	20	8	144	140	143	142	143	145	144	140	144	142	5771.3
30	20	9	139	138	136	139	138	137	138	139	139	138	5772.7
50	10	0	157	156	157	159	157	159	158	157	155	156	6660.4
50	10	1	160	159	161	160	160	159	162	162	163	158	6653.4
50	10	2	149	154	152	157	151	149	151	152	153	151	6660.3
50	10	3	164	166	165	162	165	166	168	165	166	165	6667.5
50	10	4	167	168	166	163	169	165	167	167	166	167	6667.8
50	10	5	146	146	145	147	145	144	146	146	146	145	6663.3
50	10	6	167	166	166	166	166	164	164	164	167	165	6665.9
50	10	7	179	183	183	183	182	183	183	182	185	180	6671.2
50	10	8	173	172	173	172	173	171	172	172	172	172	6661.4
50	10	9	160	163	158	162	161	162	161	161	162	160	6663.6
50	20	0	275	273	269	265	272	274	271	274	269	273	9697.5
50	20	1	232	236	232	234	236	226	231	230	234	229	9680.5
50	20	2	224	232	228	226	229	232	235	228	226	226	9681.9
50	20	3	222	223	224	220	225	224	224	219	224	222	9691.2
50	20	4	247	247	242	245	242	244	240	242	237	244	9690.1
50	20	5	228	224	220	221	226	226	222	227	220	224	9696.2
50	20	6	257	252	248	247	247	250	254	253	261	253	9689.7
50	20	7	258	255	257	259	247	253	254	254	252	254	9675.5
50	20	8	247	246	252	255	251	250	248	253	251	248	9699.5
50	20	9	261	262	262	262	266	263	259	265	262	263	9690.5

Table 3.4: H_1 Results for instances 30.20.0 to 50.20.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
75	10	0	235	236	238	234	229	238	237	239	**228**	237	10278.5
75	10	1	222	221	220	220	221	222	221	222	224	218	10364.7
75	10	2	251	247	244	248	245	247	249	248	248	249	10159.3
75	10	3	262	263	263	259	262	261	260	261	264	259	10154.2
75	10	4	283	283	279	282	283	282	280	284	285	281	10046.1
75	10	5	266	266	263	264	265	266	263	267	260	261	10121.2
75	10	6	260	261	261	260	261	259	257	257	261	260	10046.1
75	10	7	310	306	309	310	306	309	305	307	312	311	10327.0
75	10	8	249	255	254	252	255	255	255	252	253	252	10094.5
75	10	9	240	241	241	238	239	240	238	237	241	238	10169.6
75	20	0	465	469	470	463	471	473	472	472	472	470	14905.7
75	20	1	411	398	405	410	411	406	407	402	403	400	14579.6
75	20	2	380	377	383	379	376	377	377	371	385	375	14819.4
75	20	3	397	396	393	385	391	395	394	397	402	397	14624.3
75	20	4	367	364	364	364	364	365	358	370	368	367	14544.3
75	20	5	357	356	354	355	350	356	355	359	362	356	14861.6
75	20	6	420	423	413	422	420	413	414	423	419	407	14664.5
75	20	7	406	412	407	408	405	410	408	413	417	411	14574.6
75	20	8	441	431	438	433	432	439	438	431	439	433	14587.5
75	20	9	362	362	359	359	357	361	356	362	358	364	14964.3
100	10	0	322	311	325	318	313	318	318	316	318	317	14055.3
100	10	1	332	324	325	325	329	328	331	325	328	329	14136.9
100	10	2	336	343	342	337	333	331	333	332	336	333	14222.7
100	10	3	388	390	388	392	391	385	389	385	388	385	14331.0
100	10	4	325	328	323	322	325	323	322	323	322	323	13999.5
100	10	5	345	350	351	348	351	350	352	353	352	351	14064.4
100	10	6	302	301	301	301	301	295	301	300	301	296	13846.4
100	10	7	322	327	328	323	324	329	326	322	326	324	13944.5
100	10	8	402	398	403	402	398	397	403	400	402	402	14498.6
100	10	9	322	323	318	320	320	324	321	322	323	324	14125.0
100	20	0	513	510	508	506	512	509	520	499	506	515	20558.0
100	20	1	542	540	543	545	547	538	546	548	543	547	20548.0
100	20	2	568	569	571	567	572	575	570	573	572	567	20805.6
100	20	3	520	492	498	513	514	511	511	510	507	512	20401.2
100	20	4	529	527	532	538	528	525	530	519	526	527	20344.4
100	20	5	518	508	516	506	508	515	515	506	516	519	20404.6
100	20	6	538	536	539	534	527	539	545	540	535	543	20511.4
100	20	7	551	556	554	552	555	552	558	555	547	556	20598.5
100	20	8	536	534	540	541	541	525	535	533	537	535	20578.2
100	20	9	542	542	544	537	535	542	530	533	536	529	20288.4

Table 3.5: H_1 Results for instances 75.10.0 to 100.20.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
20	10	0	60	56	62	62	78	62	62	63	66	64	0.05
20	10	1	66	60	58	67	64	64	62	62	60	62	0.04
20	10	2	68	75	70	72	74	78	70	68	77	74	0.04
20	10	3	57	66	61	66	62	62	66	60	62	62	0.04
20	10	4	52	54	52	54	56	50	54	52	50	58	0.04
20	10	5	69	66	46	44	48	46	50	68	59	49	0.04
20	10	6	66	65	71	69	66	65	73	62	64	69	0.04
20	10	7	70	70	66	66	71	66	66	68	65	68	0.04
20	10	8	70	69	74	68	69	64	72	68	62	69	0.04
20	10	9	65	63	64	60	63	73	60	64	54	64	0.04
20	20	0	66	106	72	83	82	70	74	82	84	78	0.05
20	20	1	**62**	76	78	73	69	78	70	82	89	72	0.05
20	20	2	89	104	86	82	98	98	82	**69**	90	78	0.04
20	20	3	84	74	84	77	68	75	88	62	66	58	0.05
20	20	4	84	99	88	90	94	**70**	76	78	93	78	0.06
20	20	5	90	101	98	92	108	91	85	92	108	82	0.05
20	20	6	70	77	72	86	74	58	76	70	63	68	0.05
20	20	7	72	79	66	80	76	78	77	70	80	72	0.05
20	20	8	88	86	75	94	84	82	100	84	94	78	0.04
20	20	9	77	78	76	70	82	76	64	74	82	76	0.05
30	10	0	79	85	93	84	82	81	78	94	92	86	0.24
30	10	1	97	93	88	93	91	96	88	88	82	82	0.22
30	10	2	99	88	94	95	87	98	85	83	94	84	0.23
30	10	3	84	86	89	94	91	89	94	90	98	87	0.22
30	10	4	102	91	85	89	92	95	89	89	97	92	0.22
30	10	5	92	95	98	91	94	89	104	83	88	96	0.22
30	10	6	110	113	111	101	115	104	104	113	106	97	0.23
30	10	7	93	98	96	88	100	94	97	98	94	108	0.18
30	10	8	98	100	91	100	97	91	99	96	97	99	0.21
30	10	9	92	89	93	89	88	92	83	92	88	94	0.22

Table 3.6: H_2 Results for instances 20.10.0 to 30.10.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
30	20	0	123	132	122	104	126	128	124	122	126	142	0.23
30	20	1	126	122	109	122	138	120	127	129	136	132	0.24
30	20	2	136	131	120	122	119	132	132	142	131	138	0.26
30	20	3	132	128	136	118	124	129	110	119	120	116	0.23
30	20	4	128	131	128	126	120	137	132	137	158	126	0.23
30	20	5	140	134	126	124	120	140	118	135	120	122	0.24
30	20	6	139	140	132	133	136	159	126	134	151	152	0.27
30	20	7	136	139	134	133	128	134	146	134	136	127	0.22
30	20	8	112	128	124	110	112	116	130	127	152	123	0.24
30	20	9	132	140	120	120	127	114	127	114	133	130	0.26
50	10	0	146	169	153	154	172	161	161	166	161	160	2.42
50	10	1	153	153	150	149	160	177	153	161	159	170	2.19
50	10	2	158	164	166	163	154	162	168	151	152	164	2.45
50	10	3	158	167	164	176	166	163	164	160	186	164	2.34
50	10	4	161	156	157	160	158	159	167	162	157	163	2.48
50	10	5	158	151	151	149	166	158	142	154	150	172	2.27
50	10	6	149	142	152	143	157	138	148	163	142	147	2.34
50	10	7	160	162	157	180	170	160	162	171	162	165	2.54
50	10	8	146	155	143	136	155	145	148	154	137	141	2.29
50	10	9	165	161	154	153	154	153	149	153	164	163	2.33
50	20	0	221	243	221	236	228	224	216	235	245	220	2.72
50	20	1	217	234	196	213	235	217	213	222	242	220	2.54
50	20	2	224	235	235	224	218	199	218	218	210	224	2.49
50	20	3	225	252	200	227	212	228	220	215	238	202	2.69
50	20	4	206	200	202	211	209	214	192	190	208	190	2.36
50	20	5	192	180	197	200	223	207	212	180	183	185	2.19
50	20	6	179	177	211	218	191	194	214	205	206	188	2.62
50	20	7	220	228	244	258	249	232	226	242	245	265	2.24
50	20	8	227	208	224	227	239	216	212	210	212	230	2.25
50	20	9	248	232	237	247	228	238	246	232	227	248	2.20

Table 3.7: H_2 Results for instances 30.20.0 to 50.20.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
75	10	0	279	269	246	254	274	259	266	279	257	261	16.36
75	10	1	231	233	247	233	248	269	247	235	249	249	16.01
75	10	2	235	245	250	241	255	247	240	250	241	249	16.62
75	10	3	243	250	247	242	255	253	261	251	251	246	15.08
75	10	4	275	255	255	251	254	248	254	239	260	256	15.30
75	10	5	258	243	247	262	249	248	250	243	250	243	17.48
75	10	6	245	230	253	232	236	241	244	235	250	241	17.19
75	10	7	243	224	233	242	256	247	237	237	237	237	17.26
75	10	8	251	265	280	264	287	262	262	264	278	278	15.37
75	10	9	250	247	245	253	265	269	268	258	247	255	16.38
75	20	0	336	337	352	330	330	335	314	340	332	328	18.46
75	20	1	335	344	332	312	319	300	326	311	309	324	18.87
75	20	2	337	349	339	356	361	360	345	340	341	331	18.76
75	20	3	337	340	328	336	310	352	345	326	360	324	18.42
75	20	4	352	332	343	329	327	330	329	332	329	334	18.27
75	20	5	319	352	348	327	352	334	327	321	351	343	18.46
75	20	6	357	358	342	343	361	340	347	340	336	368	16.88
75	20	7	383	358	403	374	**349**	379	362	370	377	360	17.59
75	20	8	342	337	341	354	345	340	325	341	**323**	352	18.91
75	20	9	349	345	324	312	339	347	334	334	**310**	351	18.09
100	10	0	340	331	339	336	326	338	318	358	330	317	69.46
100	10	1	305	323	326	307	314	304	310	332	325	312	66.55
100	10	2	333	341	327	338	337	330	347	315	349	343	66.31
100	10	3	326	323	334	313	311	315	**287**	317	312	323	62.72
100	10	4	324	335	318	333	315	307	314	299	318	323	62.35
100	10	5	302	308	325	318	322	309	310	339	323	319	66.63
100	10	6	301	329	313	298	315	299	295	299	312	297	66.20
100	10	7	310	314	329	331	331	330	331	320	320	316	65.71
100	10	8	342	346	335	347	341	343	336	341	328	340	60.73
100	10	9	321	312	310	306	309	310	318	327	325	310	68.56
100	20	0	396	426	401	413	428	407	405	404	411	398	72.28
100	20	1	440	430	476	449	449	442	432	431	432	445	73.49
100	20	2	476	471	497	466	482	471	497	498	461	489	70.94
100	20	3	474	462	**438**	451	485	478	478	462	471	486	80.14
100	20	4	467	480	464	475	468	466	471	465	480	468	80.05
100	20	5	429	471	432	452	437	478	437	424	442	416	64.87
100	20	6	449	450	452	455	465	466	440	449	449	474	69.96
100	20	7	500	490	490	518	494	495	494	500	500	495	70.50
100	20	8	495	465	475	492	**464**	467	485	473	486	513	75.71
100	20	9	480	518	469	488	497	490	491	489	468	507	73.86

Table 3.8: H_2 Results for instances 75.10.0 to 100.20.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
20	10	0	60	58	60	56	56	60	58	58	56	**54**	39873.0
20	10	1	57	**50**	60	54	60	52	52	58	60	56	39873.3
20	10	2	81	76	79	82	78	80	84	78	83	83	39881.9
20	10	3	56	54	54	54	54	52	54	54	54	54	39866.5
20	10	4	73	50	62	71	71	72	73	68	73	55	39875.0
20	10	5	50	55	42	58	50	56	51	48	46	46	39866.8
20	10	6	64	64	58	62	64	62	62	60	62	62	39871.0
20	10	7	61	61	58	59	61	64	64	60	58	61	39873.3
20	10	8	54	60	61	56	56	58	59	58	54	58	39872.9
20	10	9	54	**52**	**52**	53	53	**52**	**52**	**52**	**52**	**52**	39868.8
20	20	0	106	96	143	108	118	100	92	106	112	121	54387.4
20	20	1	93	86	80	85	82	82	82	84	82	82	54372.6
20	20	2	100	94	100	100	84	102	94	94	94	84	54374.5
20	20	3	87	88	83	80	78	80	83	87	68	84	54374.2
20	20	4	92	94	94	93	90	88	92	90	98	92	54376.9
20	20	5	90	82	94	92	106	92	86	88	86	96	54372.6
20	20	6	88	88	88	89	88	90	90	80	90	81	54376.0
20	20	7	98	88	97	93	92	91	88	89	91	90	54377.3
20	20	8	92	98	96	102	94	90	102	104	103	96	54382.3
20	20	9	118	116	115	104	80	106	106	96	109	84	54387.1
30	10	0	86	77	83	**74**	82	85	82	87	**74**	78	40354.3
30	10	1	**74**	92	92	79	83	84	86	91	82	86	40365.5
30	10	2	80	80	89	84	83	78	80	82	86	81	40360.6
30	10	3	80	82	80	86	85	83	81	80	87	**74**	40361.4
30	10	4	81	79	87	80	88	81	89	82	86	86	40351.7
30	10	5	78	86	86	84	85	82	85	85	80	84	40350.3
30	10	6	112	104	108	108	100	108	112	106	106	101	40373.7
30	10	7	86	88	86	88	85	82	84	83	92	81	40363.8
30	10	8	**81**	90	87	92	83	85	88	88	90	92	40366.0
30	10	9	82	80	88	86	78	88	82	**77**	82	88	40353.3

Table 3.9: H_3 Results for instances 20.10.0 to 30.10.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
30	20	0	136	140	133	149	141	142	150	154	133	150	55082.7
30	20	1	150	146	146	154	147	162	152	143	143	135	55062.6
30	20	2	143	137	135	132	138	142	139	143	147	138	55085.8
30	20	3	134	142	140	135	124	135	138	141	134	143	55045.8
30	20	4	138	146	152	147	154	153	157	151	151	147	55068.0
30	20	5	134	145	134	141	136	144	143	136	144	142	55049.5
30	20	6	158	159	166	165	168	151	154	152	167	165	55062.2
30	20	7	158	136	156	144	160	148	149	158	145	150	55050.9
30	20	8	134	136	134	126	121	135	134	131	131	131	55054.7
30	20	9	141	137	142	146	141	142	142	135	139	140	55065.4
50	10	0	139	**138**	141	143	148	146	142	**138**	140	147	40741.6
50	10	1	148	148	144	147	145	142	147	150	150	139	40784.9
50	10	2	153	159	148	153	159	150	153	149	154	144	40870.7
50	10	3	151	155	146	157	151	152	153	154	154	150	40897.0
50	10	4	**144**	147	151	149	151	**144**	149	146	155	**144**	40907.4
50	10	5	138	139	139	142	**134**	144	140	138	138	138	40727.1
50	10	6	136	143	144	138	139	143	139	138	139	136	40850.1
50	10	7	155	154	159	162	162	157	163	165	164	159	40931.7
50	10	8	141	137	143	**128**	144	152	140	133	143	142	40738.6
50	10	9	148	145	151	147	141	142	140	140	153	145	40788.7
50	20	0	247	222	248	256	255	249	247	248	253	248	55602.5
50	20	1	250	239	226	242	229	241	247	249	251	246	55502.6
50	20	2	219	236	231	225	210	244	230	221	240	241	55123.8
50	20	3	246	235	250	242	215	223	250	230	244	226	55361.9
50	20	4	222	211	223	219	225	223	225	220	212	229	55839.8
50	20	5	223	219	225	225	213	223	216	214	221	204	55485.7
50	20	6	259	257	259	242	243	258	247	259	259	267	55911.4
50	20	7	240	240	248	237	238	239	245	245	242	237	55204.1
50	20	8	246	224	248	235	249	231	235	243	245	250	55727.5
50	20	9	246	251	244	234	253	254	245	250	251	248	55218.0

Table 3.10: H_3 Results for instances 30.20.0 to 50.20.9

n	l	#	1	2	3	4	5	6	7	8	9	10	CPU
75	10	0	239	235	240	246	241	246	240	247	236	239	40067.4
75	10	1	227	228	**217**	227	223	226	229	223	230	228	39985.6
75	10	2	230	240	234	245	239	235	237	231	**218**	238	40032.5
75	10	3	238	227	241	234	228	235	223	232	240	241	41167.8
75	10	4	240	244	238	241	232	242	237	218	239	229	40040.0
75	10	5	236	231	230	229	230	228	227	214	231	232	40263.6
75	10	6	237	241	252	247	249	247	233	241	238	235	40085.5
75	10	7	246	256	242	242	237	234	227	234	239	258	41456.0
75	10	8	233	232	227	231	234	232	234	229	232	**226**	40058.9
75	10	9	223	237	233	234	230	234	233	224	232	232	40055.0
75	20	0	404	372	383	373	388	387	376	401	396	382	54689.3
75	20	1	377	384	374	395	383	352	365	367	374	347	54771.9
75	20	2	372	370	369	365	373	374	364	350	371	381	54760.8
75	20	3	364	357	364	361	370	367	361	363	362	358	55093.9
75	20	4	344	353	350	342	338	339	348	347	350	335	54690.2
75	20	5	378	374	373	358	376	386	367	343	367	383	54655.3
75	20	6	372	399	345	383	380	390	375	353	381	382	55010.4
75	20	7	370	381	384	367	361	383	369	371	372	377	54725.5
75	20	8	419	396	430	414	402	428	426	433	426	422	54770.4
75	20	9	346	346	348	340	336	348	343	351	346	350	54544.2
100	10	0	316	320	323	310	321	320	310	**305**	318	320	41957.1
100	10	1	298	298	298	304	309	302	312	313	289	310	42790.0
100	10	2	309	311	308	316	**300**	318	316	308	311	313	42446.6
100	10	3	311	354	305	325	313	327	323	325	299	317	42059.3
100	10	4	306	302	300	309	304	318	296	306	299	301	42556.1
100	10	5	309	317	317	321	330	321	327	328	309	335	41613.2
100	10	6	318	331	312	312	314	317	318	306	316	316	42573.4
100	10	7	310	309	312	308	309	312	303	310	308	310	42061.3
100	10	8	336	346	341	352	349	345	333	346	344	344	42747.9
100	10	9	310	300	301	304	307	305	319	295	307	298	41940.8
100	20	0	479	475	503	478	514	501	436	473	498	442	57065.2
100	20	1	499	481	467	509	530	492	484	515	452	495	56873.7
100	20	2	532	528	536	518	523	512	499	525	490	499	57381.3
100	20	3	499	494	506	489	507	493	494	507	511	490	57173.0
100	20	4	509	524	529	532	516	517	534	518	517	524	56583.8
100	20	5	496	485	469	496	498	466	486	498	471	489	56384.0
100	20	6	483	513	511	498	494	496	482	464	490	483	57479.7
100	20	7	500	534	513	513	543	521	509	532	501	525	57441.1
100	20	8	498	513	513	526	507	527	536	504	518	519	56348.1
100	20	9	501	528	522	517	521	507	533	530	524	513	56582.9

Table 3.11: H_3 Results for instances 75.10.0 to 100.20.9

n	l	#	Best	1	2	3	4	5	6	7	8	9	10	CPU
20	10	0	54	**54**	56	56	55	**54**	56	56	55	56	56	240.04
20	10	1	50	**50**	52	52	51	51	**50**	52	53	52	52	240.04
20	10	2	64	**64**	65	66	66	**64**	66	65	66	65	65	240.04
20	10	3	47	50	**47**	50	50	49	51	52	50	50	52	240.04
20	10	4	42	44	44	44	46	44	46	44	44	44	**42**	240.05
20	10	5	40	42	**40**	**40**	**40**	**40**	42	**40**	42	42	42	240.04
20	10	6	54	55	56	**54**	55	56	**54**	56	**54**	**54**	55	240.04
20	10	7	54	56	56	56	56	58	**54**	56	56	58	57	240.04
20	10	8	52	**52**	54	55	55	53	53	54	53	54	54	240.05
20	10	9	52	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	240.05
20	20	0	64	66	**64**	69	68	67	66	68	**64**	66	**64**	240.06
20	20	1	62	64	**62**	65	**62**	**62**	64	67	66	65	64	240.08
20	20	2	70	76	72	72	72	78	74	70	74	71	74	240.07
20	20	3	54	55	58	58	56	59	62	**54**	60	56	58	240.06
20	20	4	72	74	72	73	74	76	78	76	74	75	74	240.06
20	20	5	76	80	77	78	80	80	82	80	**76**	80	82	240.08
20	20	6	56	**56**	60	60	60	58	58	64	58	**56**	58	240.07
20	20	7	62	64	64	64	68	66	**62**	65	66	64	64	240.08
20	20	8	60	**60**	68	66	68	66	67	68	72	70	66	240.04
20	20	9	58	**58**	60	61	61	60	60	60	59	**58**	60	240.09
30	10	0	74	**74**	79	77	75	77	76	**74**	77	77	77	360.17
30	10	1	77	78	80	80	80	78	78	77	78	78	78	360.12
30	10	2	76	78	77	79	**76**	**76**	79	80	81	77	77	360.09
30	10	3	75	78	78	76	76	76	75	77	78	76	78	360.12
30	10	4	74	**74**	75	**74**	**74**	76	75	76	76	76	**74**	360.14
30	10	5	77	78	80	80	83	**77**	81	79	78	80	79	360.12
30	10	6	91	**91**	95	95	96	92	92	93	94	93	92	360.14
30	10	7	78	83	**78**	82	79	81	79	82	82	80	82	360.14
30	10	8	82	87	89	82	90	87	86	87	88	90	88	360.15
30	10	9	79	79	81	82	80	80	79	80	82	79	80	360.15

Table 3.12: H_4 Results for instances 20.10.0 to 30.10.9

n	l	#	Best	1	2	3	4	5	6	7	8	9	10	CPU
30	20	0	102	108	106	**102**	106	112	112	109	114	106	112	360.16
30	20	1	108	109	110	**108**	**108**	109	**108**	**108**	114	113	112	360.17
30	20	2	111	118	116	117	114	120	118	**111**	112	122	112	360.18
30	20	3	106	**106**	110	109	108	110	113	110	108	109	107	360.17
30	20	4	114	118	125	120	**114**	120	120	118	122	118	120	360.18
30	20	5	106	110	108	111	108	114	**106**	**106**	112	107	108	360.21
30	20	6	116	124	122	120	124	125	126	122	123	121	**116**	360.15
30	20	7	117	120	120	124	**117**	122	122	120	122	122	121	360.18
30	20	8	105	110	106	106	112	110	108	114	113	**105**	109	360.15
30	20	9	113	114	118	119	120	118	120	117	119	118	**113**	360.17
50	10	0	139	141	141	145	143	139	145	140	139	139	144	600.52
50	10	1	138	140	142	**138**	141	139	139	144	141	143	141	600.51
50	10	2	142	144	144	146	146	149	149	**142**	146	146	147	600.67
50	10	3	145	150	**145**	147	149	149	146	148	149	151	148	600.51
50	10	4	144	145	147	**144**	**144**	150	149	150	150	148	**144**	600.70
50	10	5	134	**134**	**134**	136	136	135	**134**	**134**	138	136	137	600.45
50	10	6	128	130	**128**	**128**	132	130	131	132	133	**128**	132	600.47
50	10	7	144	150	152	148	155	147	148	**144**	148	147	151	600.71
50	10	8	129	129	134	134	133	130	133	131	134	133	129	600.51
50	10	9	138	143	**138**	146	144	142	145	143	142	142	141	600.51
50	20	0	202	208	208	210	**202**	213	206	210	204	209	215	600.63
50	20	1	192	200	199	200	**192**	196	196	200	198	194	196	600.47
50	20	2	197	200	198	202	200	202	203	208	**197**	200	**197**	600.72
50	20	3	196	205	206	208	199	**196**	204	206	211	208	212	600.59
50	20	4	178	188	181	181	187	190	186	**178**	186	184	**178**	600.73
50	20	5	174	176	176	180	180	**174**	178	**174**	178	182	175	600.77
50	20	6	169	172	175	170	178	**169**	175	170	172	174	174	600.60
50	20	7	216	220	**216**	220	221	222	222	220	224	221	223	600.80
50	20	8	196	209	205	203	**196**	197	200	202	209	199	206	600.63
50	20	9	218	222	222	224	226	222	225	225	**218**	219	229	600.67

Table 3.13: H_4 Results for instances 30.20.0 to 50.20.9

n	l	#	Best	1	2	3	4	5	6	7	8	9	10	CPU
75	10	0	240	245	250	248	246	245	246	240	244	243	240	901.30
75	10	1	217	225	223	230	**217**	228	228	228	223	**217**	228	902.00
75	10	2	224	228	232	228	229	224	229	231	226	229	227	902.20
75	10	3	219	225	229	229	223	224	222	230	**219**	230	225	901.70
75	10	4	217	230	221	224	229	228	228	**217**	225	232	222	902.38
75	10	5	213	224	217	224	219	225	230	220	**213**	219	218	901.79
75	10	6	222	**222**	224	223	224	**222**	228	228	223	223	228	902.09
75	10	7	213	220	**213**	215	216	221	221	219	216	222	221	901.82
75	10	8	226	235	232	**226**	237	228	**226**	229	234	234	229	902.22
75	10	9	218	227	227	**218**	222	219	220	223	224	227	232	901.47
75	20	0	303	320	312	315	307	**303**	314	321	316	314	316	902.62
75	20	1	287	298	**287**	297	304	300	290	300	303	296	304	901.72
75	20	2	322	327	323	324	325	332	330	332	331	335	**322**	901.45
75	20	3	302	311	313	315	310	315	315	316	313	311	**302**	901.80
75	20	4	309	312	318	311	**309**	310	316	314	314	319	321	902.19
75	20	5	309	326	330	319	321	313	335	324	326	**309**	325	902.24
75	20	6	315	316	331	321	323	323	**315**	320	329	320	330	901.49
75	20	7	349	**349**	351	353	357	352	357	**349**	353	357	**349**	902.13
75	20	8	324	332	324	328	335	324	328	338	327	328	328	902.19
75	20	9	310	313	316	315	321	**310**	314	312	324	322	319	901.85
100	10	0	307	309	313	313	314	310	314	309	316	307	311	1204.48
100	10	1	285	287	292	**285**	296	303	293	288	288	293	295	1203.72
100	10	2	306	312	307	318	317	316	316	306	315	311	318	1205.41
100	10	3	288	291	289	290	294	296	296	288	290	295	292	1205.48
100	10	4	284	295	295	290	289	290	**284**	289	292	297	296	1204.22
100	10	5	289	296	302	301	292	295	**289**	292	291	290	295	1204.16
100	10	6	280	285	285	**280**	289	286	283	285	291	291	**280**	1204.25
100	10	7	298	304	**298**	300	306	305	301	307	302	302	299	1206.10
100	10	8	318	326	322	321	**318**	322	321	323	322	324	323	1204.78
100	10	9	281	289	285	**281**	288	295	289	287	290	291	283	1205.40
100	20	0	373	375	388	378	387	376	384	381	**373**	386	397	1204.96
100	20	1	406	412	**406**	413	**406**	429	423	431	415	409	420	1204.96
100	20	2	441	**441**	443	462	450	447	458	449	442	449	465	1204.80
100	20	3	448	452	458	460	448	449	451	458	469	456	456	1204.62
100	20	4	447	451	**447**	454	451	458	453	463	468	454	463	1204.82
100	20	5	401	427	**401**	421	411	416	426	406	425	424	416	1205.31
100	20	6	409	442	429	442	434	432	423	**409**	427	438	437	1205.42
100	20	7	466	484	485	474	479	481	472	482	**466**	473	478	1203.22
100	20	8	465	471	465	474	466	471	475	465	472	480	468	1204.86
100	20	9	448	471	**448**	458	477	463	451	475	468	459	454	1205.57

Table 3.14: H_4 Results for instances 75.10.0 to 100.20.9

Chapter 4

Algorithms for the Calzedonia Workload Allocation Problem

Statement of authorship

This declaration concerns the article entitled:
Algorithms for the Calzedonia workload allocation problem

Publication status: Published

Publication details:
Maria Battarra, Federico Fraboni, Oliver Thomasson,
Güneş Erdoğan, Gilbert Laporte, and Marco Formentini.
(2020)
Algorithms for the Calzedonia workload allocation problem.
Journal of the Operational Research Society
Published online: 23 Jun 2020
`https://doi.org/10.1080/01605682.2020.1755897`

Candidates contribution to the paper

- Discovery & exploration of idea (20%)
- Mathematical modelling (30%)
- Heuristic development (60%)
- Coding & testing (80%)
- Initial write-up (50%)
- Editing for publication (30%)

Statement from candidate
This paper reports on original research I conducted during the period of my Higher Degree by Research candidature

Signed:
[redacted]

Date:
Wednesday 7th July, 2021

4.1 Introduction

We introduce the Workload Assignment Problem (WAP) which consists of assigning an ordered sequence of $|S|$ operations to a set of workers. The order of the operations is fixed, and each operation consists of a batch of B units, hence a total of $|J| = B \times |S|$ jobs have to be performed. A worker is assigned to exactly one set of consecutive jobs. Workers have different skills, and therefore the time to complete jobs differs among workers. However, contiguous jobs may take the same time if performed by the same worker and if they belong to the same *operation*. Operations are the production steps necessary to complete a product. Each operation has to be performed once on each product of the batch, before sequentially moving to the next operation. A worker is assumed to take the same time to complete the same operation on each product in the batch. A *job* is defined as the process of completing one of the operations on one product in the batch. The objective is the minimisation of the longest *workload* among the workers, where the workload is defined as the sum of the processing times of all jobs assigned to a worker.

The closest problem in the literature is the Assembly Line Worker Assignment and Balancing Problem (ALWABP), introduced by Miralles et al. (2008). Our problem differs from ALWABP since it involves batch production and a precedence graph that reduces to a line. These differences allow us to develop exact solution methods with higher efficiency than the available ones.

This work is motivated by our collaboration with Calzedonia, an Italian apparel company. The WAP reflects the workload allocation practice in some of their production sites. We present two exact algorithms and a metaheuristic, and compare our results with those of a heuristic implemented at the company. We show the effectiveness of our metahueristic, with its short CPU time requirement and superior performance.

A formal definition of the WAP is given in Section 4.1.1. A description of the factory settings that motivated this problem is provided in Section 4.1.2. Similar scheduling problems are reviewed in Section 4.2. Section 4.3 presents a valid mathematical formulation for the WAP. Section 4.4 describes our sequential exact algorithm. Section 4.5 describes the heuristic algorithm that was implemented by Calzedonia, as well as our metaheuristic. Computational results are given in Section 4.6, followed by conclusions and future research directions in Section 4.7.

4.1.1 Problem definition

A set of workers W is responsible for the sewing operations of a single type of product on a production line. Each product is manufactured by completing an ordered sequence S of sewing operations. The workers manufacture the product in batches of size B. In order to complete a batch of products, an ordered set of jobs J has to be performed, where $|J| = B \times |S|$ (i.e., all operations in order, on all products in the batch). Each worker k requires t_{ik} units of time to complete job i. However, $t_{ik} = t_{jk}$ if jobs i and j belong to the same operation. The problem is to assign each worker to one contiguous subset of J, so that every job is assigned to exactly one worker. Note that jobs cannot be shared between workers, whereas the jobs belonging to the same operation can be shared among neighbouring workers.

The objective is to minimise the time for the slowest worker to complete their assigned jobs ($\max_{k \in W} \sum_{i \in J} t_{ik} x_{ik}$, where $x_{ik} = 1$ if worker k performs job i, 0 otherwise), which reduces the likelihood of bottlenecks, promoting a smooth flow of products in the line (or a well-paced line). This objective, which we define as z, guarantees that a batch of products is completed every z units of time. It is typically referred to as 'minimising the maximum process time' (Li et al., 2015, 2017). We therefore use this descriptor for our objective in the remainder of this paper. Finally note that the production is repetitive (i.e., a new batch of products starts being processed whenever the first worker completes his or her jobs from the previous batch), however our analysis can focus on the workload allocation for a single batch without any loss of generality.

4.1.2 Motivation

Calzedonia is an Italian fashion company established in 1986 that specialises in hosiery, garments, and beachwear for women, men, and children. The company is structured vertically, i.e., the design, production, and distribution of products are handled either directly, or through affiliates. Calzedonia sells its products in over 30 countries, and employs approximately 26,000 workers. Their factories are located in Bulgaria, Croatia, Romania, Ethiopia, Serbia, and Sri Lanka. This research originated in one of Calzedonia's Sri Lankan factories which specialises in bra production.

The Sri Lankan factory produces hosiery and hosts approximately 150 cells, each of which produces a single type of product on any given day. Each bra requires 18 to 42 sewing operations, depending on the complexity of the product. The number of workers is typically between nine and 15 in each cell. The Calzedonia headquarters provide the ordered sequence of operations to sew a product, as well as, for each operation m,

Figure 4-1: Bra production in the Calzedonia Sri Lankan factory

the estimated processing time $\bar{t}_m$ taken from the General Sewing Data (GSD, 2017). The initial workload allocation reflects the order of the sewing operations given by headquarters, so that the cell can be considered as a production line for algorithmic purposes.

Bra production is labour-intensive and requires skilled sewing of many components, which are typically small (Hardaker and Fozzard, 1997). While Sri Lankan workers achieve high levels of productivity and sewing standards, it is common to encounter absenteeism (Arai, 2006; Kelegama, 2009; Wickramasinghe and Wickramasinghe, 2011). Most of the sewing workforce is composed of young females who tend to end their career once they get married. They frequently miss work days due to family events (e.g., weddings and funerals) or personal circumstances, without informing the company. This leads to some sewing operations having no worker to complete them. It has been estimated that 7% of the workers are absent from work, without notice, on any given day. On average, this means that each production line is missing one worker every morning, and leads to a drop of 40 to 50% in productivity in the first hours of the day. The company estimated that, before our collaboration, the average time necessary to complete the factory's daily workload reallocation was one hour and 40 minutes (18% of the working day). The previous workload allocation was performed manually by line managers, each being responsible for eight production lines.

Finally, a preliminary analysis conducted by the company highlighted that the employee turnover rate is approximately 50% per annum, leading to thousands of new employees

needing to be trained every year. This adds to the variability of sewing times; a worker new to the job requires much longer on the same task than an experienced worker. Considering the variability in jobs' process times is therefore crucial to obtaining good-quality solutions.

The WAP therefore models the problem encountered every morning by line managers in the factory. Whenever one or more workers are absent from a cell, the ordered sequence of jobs has to be reassigned among the workers, considering their skill levels. Workload allocations have to be planned quickly, so that high levels of efficiency can be reached from the start of the working day, and workers can either process the work-in-progress from the previous day, or start processing a new product if there is no work-in-progress inventory.

4.2 Literature review

Brucker et al. (2011) and De Bruecker et al. (2015) provide overviews of the personnel scheduling literature. The problems reviewed capture the needs of shift creation, staffing, and scheduling. There also exists a large body of literature on workload allocation and how the workers' skills should be taken into consideration.

In our industrial setting, each cell in the factory is an assembly system, given that a batch of products and workers move from one operation to the next, until the batch of products is completely assembled. More precisely, a single-model assembly line problem is studied, because a single product is assembled. The surveys by Scholl and Becker (2006), Becker and Scholl (2006), Boysen et al. (2007), Boysen et al. (2008), and Battaïa and Dolgui (2013) provide detailed literature reviews and classification schemes for assembly line balancing problems. The WAP is a special case of the single-model assembly line problem, because the "precedence graph" among jobs reduces to a line.

Fırat et al. (2016) aim to define a stable assignment between technicians and jobs. Miralles et al. (2007) and Moreira et al. (2015) study a problem in which disabled people have to be assigned to workstations and jobs in a production line. Precedence constraints among operations are present and production times differ between workers. This problem is called the assembly line worker assignment and balancing problem, and has since been widely studied (Miralles et al., 2008; Chaves et al., 2009; Blum and Miralles, 2011; Mutlu et al., 2013; Borba and Ritt, 2014; Vilà and Pereira, 2014; Moreira et al., 2015). Zacharia and Nearchou (2016) extend the problem definition to include a multi-criteria objective function consisting of the cycle time and the

smoothness index of the workload of the line, i.e., how evenly the workload is split between workers.

A related line of research has been conducted on the *Robust Assembly Line Balancing Problem* (RALBP) and its variants, which aim to incorporate the heterogeneity of the processing times through uncertainty, usually expressed as an interval for each processing time. Although the resulting solutions are robust, the algorithms require longer computation times to deal with the additional complexity. Hazir and Dolgui (2013) study the RALBP, and provide a mathematical model as well as a decomposition-based exact solution algorithm. The authors solve instances with 29 to 70 operations, for which their algorithm requires an average of 6186.29 seconds. In recent work, Pereira (2018) studies the Robust (Minmax Regret) Assembly Line Worker Assignment and Balancing Problem that involves interval type uncertainty. The author provides a formulation based on that of Borba and Ritt (2014), and presents both exact and heuristic algorithms. Through computational experiments, the author demonstrates the performance of the algorithms for instances with over 50 tasks and 7 workers. Finally, Pereira and Álvarez Miranda (2018) provide a branch-and-bound algorithm to solve the RALBP, and demonstrate that their algorithm outperforms that of Hazir and Dolgui (2013).

The workload allocation problem of Calzedonia features some characteristics of this literature, but to the best of our knowledge, no problem exactly matches the WAP. In addition, the instance sizes we aim to handle are much larger than those in the literature. The minimum number of jobs in a real instance is 630, whereas the computational reach of the formulations of Borba and Ritt (2014) are limited to instances with up to 28 tasks, the branch-and-bound algorithm of Vilà and Pereira (2014) was tested for instances with up to 75 tasks, and the computational testing of the exact algorithms of Moreira et al. (2015) are performed on instances with up to 100 tasks.

4.3 Mathematical model

In order to solve the WAP, we develop a two-index mathematical model called $2I$. In this formulation, the two-index binary variable x_{ik} assumes value 1 if and only if worker $k \in W$ performs job $i \in J$. The formulation $2I$ is as follows:

$$
\begin{aligned}
(2I)\ \text{minimise} \quad & z && && (4.1)\\
\text{subject to} \quad & z \geq \sum_{i \in J} t_{ik} x_{ik} && k \in W && (4.2)\\
& \sum_{k \in W} x_{ik} = 1 && i \in J && (4.3)\\
& x_{ik} + x_{jk} \leq x_{hk} + 1 && h, i, j \in J : i < h < j && \\
& && k \in W && (4.4)\\
& x_{ik} \in \{0, 1\} && i \in J, k \in W && (4.5)\\
& z \geq 0. && && (4.6)
\end{aligned}
$$

The objective function (4.1) minimises the maximum process time, which must be at least equal to the sum of the times of jobs assigned to any worker by Constraints (4.2). Constraints (4.3) ensure that each job is assigned to a worker, and Constraints (4.4) ensure contiguity among the jobs assigned to a worker (a pair of jobs cannot be assigned to a worker unless all intermediate jobs are also assigned to the same worker). Constraints (4.5) require the assignment variables to be binary, and Constraint (4.6) sets the maximum process time to be non-negative.

Instances of a realistic size cannot usually be solved by a mixed integer linear programming solver using this formulation. Indeed, the number of jobs $|J|$ is typically in the order of thousands, and Constraints (4.4) are of order $\mathcal{O}(|J|^3 \times |W|)$.

4.4 A sequential exact algorithm

Once computational testing showed model $2I$ to be unable to solve medium to large sized instances, we developed an alternate exact sequential algorithm. This algorithm is based on a relaxation of the problem in which we allow workers to perform all of the jobs in an operation, a fraction of the jobs in an operation, or none of the jobs in an operation. This reduces the search space to the operations instead of the jobs, and reduces the number of constraints by multiple orders of magnitude. The compromise necessary for this reduction is that using continuous variables for the number of jobs within an operation may lead to infeasible solutions for the WAP. We therefore refer to this formulation as the Continuous Operation Relaxation Formulation (CORF).

Figure 4-2 depicts a solution representation of feasible workload allocations for a worker. Each square represents an operation type in the sequence. Let C denote an operation completely performed by the worker, and F denote an operation fractionally performed

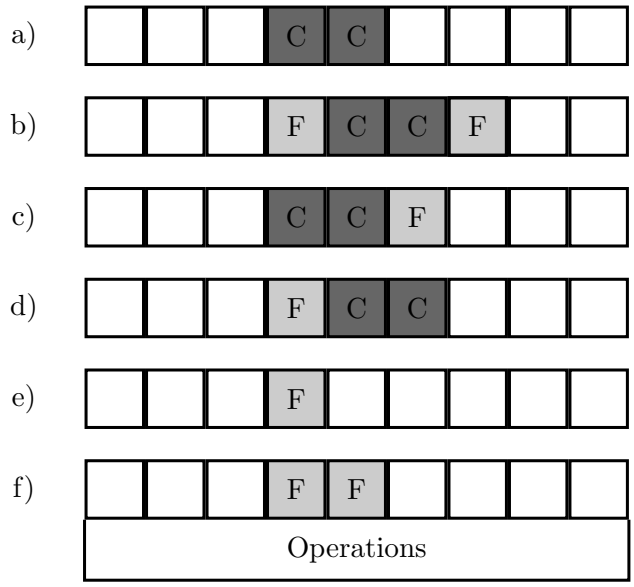

Figure 4-2: Solution representation of feasible workload allocations for a worker.

by the worker. The worker can completely perform a sequence of operations (Figure 4-2a), completely perform a sequence of operations and fractionally perform operations before or after the sequence (Figures 4-2b,c,d), or only fractionally perform one or two consecutive operations (Figures 4-2e,f).

The model CORF we have developed to solve the relaxed version of the WAP uses three new sets of variables. Let binary variable p_{mk} be equal to 1 if and only if worker $k \in W$ completely performs operation $m \in S$. The binary variable $q_{mk} = 1$ if and only if worker k fractionally performs operation m. The variable $0 \leq r_{mk} \leq 1 - \epsilon$ is the fraction of operation $m \in S$ performed by worker $k \in W$, where $\epsilon = 10^{-6}$ in our implementation. The value of r_{mk} can be greater than zero in the following two cases.

1. An operation m can be fractionally performed by worker k directly before operation $m+1$ which is completely performed by worker k, provided no operations $m-l$ (for $l \in \{1, \ldots, m-1\}$) are performed completely or fractionally by worker k. In other words, r_{mk} can be greater than zero if $p_{m+1,k} = 1$ and $\sum_{l=1}^{m-1} (p_{lk} + q_{lk}) = 0$. Vice versa, an operation m can be fractionally performed by worker k directly after operation $m-1$ which is completely performed by worker k, provided no operations $m+l$ (for $l \in \{m+1, \ldots, |S|\}$) are completely or fractionally performed by worker k. In other words, r_{mk} can be greater than zero if $p_{m-1,k} = 1$ and $\sum_{l=m+1}^{|S|} (p_{lk} + q_{lk}) = 0$. This condition allows the cases seen in Figure 4-2b,c,d.

2. An operation m can be fractionally performed by worker k if no operations are

completely performed by worker k, and at most one of the operations $m+1$ and $m-1$ is fractionally performed by k. In other words, r_{mk} can be greater than zero if $\sum_{m \in S} p_{mk} = 0$, $\sum_{l \in S \setminus \{m-1,m,m+1\}} q_{lk} = 0$, and $q_{m+1,k} + q_{m-1,k} \leq 1$. These conditions allow the cases seen in Figure 4-2e,f.

The formulation of CORF is

$$
\begin{aligned}
(CORF) \text{ minimise} \quad & z && && (4.7)\\
\text{subject to} \quad & z \geq \sum_{m \in S} (p_{mk} + r_{mk}) B \times t_{m \times B,k} && k \in W && (4.8)\\
& \sum_{k \in W} (p_{mk} + r_{mk}) = 1 && m \in S && (4.9)\\
& q_{lk} + q_{mk} + p_{lk} + p_{mk} \leq p_{ok} + 1 && l, m, o \in S && \\
& && l + 1 \leq o \leq m - 1; && \\
& && k \in W && (4.10)\\
& p_{mk} + q_{mk} \leq 1 && m \in S, k \in W && (4.11)\\
& r_{mk} \leq q_{mk} && m \in S, k \in W && (4.12)\\
& \sum_{m \in S} q_{mk} \leq 2 && k \in W && (4.13)\\
& p_{mk} \in \{0, 1\} && m \in S, k \in W && (4.14)\\
& q_{mk} \in \{0, 1\} && m \in S, k \in W && (4.15)\\
& 0 \leq r_{mk} \leq 1 - \epsilon && m \in S, k \in W && (4.16)\\
& z \geq 0. && && (4.17)
\end{aligned}
$$

The maximum process time is minimised by objective function (4.7) and bounded below by Constraint (4.8). Note the index of $t_{m \times B,k}$ instead of t_{ik} due to the fact that CORF considers operations, and t_{ik} is indexed by job. Replacing i with $m \times B$ means we are considering the time taken to complete the first job in a contiguous set of jobs of one operation. Constraints (4.9) require that each operation be performed. Constraints (4.10) impose the contiguity of operations performed for each worker and force operations fractionally performed to be either before or after a completely performed operation for worker k, if any operation has been completely performed by k. Constraints (4.11) ensure that a worker can perform an operation completely or fractionally, but not both. Constraints (4.12) link q_{mk} to the corresponding r_{mk} variable. Constraints (4.13) state that a worker can at most fractionally perform two operations.

Note that these constraints are not necessary for the validity of the formulation (they are implied by Constraints (4.10), but strengthen the formulation). The domains of the variables are defined by Constraints (4.14) to (4.17).

Proposition 1. *An optimal solution value z^*_{CORF} of CORF yields a valid lower bound LB for the WAP.*

Proof. We prove Proposition 1 by showing that any feasible solution to $2I$ is also a feasible solution of CORF. A feasible solution to $2I$ is given by a matrix of binary variables x_{ik}, where $x_{ik} = 1$ if and only if worker k performs job i (as defined in Section 4.3). For any given feasible solution, variables x_{ik} respect Constraints (4.3), (4.4), and (4.5). We can then construct a feasible solution of CORF as a combination of binary variables p_{mk} and q_{mk}, and continuous variables r_{mk}. For all $m \in S$ and $k \in W$, define $f_{mk} = \sum_{i=(m-1)B+1}^{mB} x_{ik}/B$ as the fraction of operation m undertaken by worker k. If $f_{mk} = 1$ then set $p_{mk} = 1$, $q_{mk} = 0$, and $r_{mk} = 0$. If $0 < f_{mk} < 1$ then set $p_{mk} = 0$, $q_{mk} = 1$, and $r_{mk} = f_{mk}$. If $f_{mk} = 0$ then set $p_{mk} = 0$, $q_{mk} = 0$, and $r_{mk} = 0$.

This variable assignment defines a feasible solution for CORF since all constraints are satisfied. Constraints (4.9) are satisfied since Constraints (4.2) state that each job is performed by exactly one worker. Therefore an operation can either be performed by just one worker or a combination, but the sum of assignment to the job must be one. Constraints (4.10) and (4.13) follow directly from Constraints (4.3), and Constraints (4.11) and (4.12) are satisfied by our transformation conditions. Since any solution of $2I$ is a solution to CORF, Proposition 1 holds. □

Note that CORF becomes a valid formulation for WAP if the r_{mk} variables are restricted to only values corresponding to an integer number of jobs. This can either be imposed by cutting planes or branching constraints. However, the complexity of the formulation would increase and the problem would become intractable for realistic sized instances.

4.4.1 Computing an upper bound for WAP from any feasible solution of CORF

Once a solution to CORF is found, regardless of optimality, we wish to convert the solution of the relaxation into a feasible solution to WAP. A heuristic is used to convert the best objective solution returned by CORF into an upper bound for the original problem. For each operation m, the upper bound computation heuristic (CORF-H) identifies a worker k who performs a fraction of operation m (i.e., $r_{mk} > 0$). If worker

k performs part or all of operation $m+1$, then worker k is marked as *lastWorker*, so their assignment for operation m will be rounded down to the closest integer number of jobs. Any other worker who performs part of operation m is then added to a vector A. The worker identified previously as the last worker is then added to the end of A. We then sequentially round the assignment of any worker up to the closest integer number of jobs, if it is not already integer. The corresponding fractional increment of workload is subtracted from the next worker in A. This process is reiterated for each worker in A. A pseudocode for this heuristic is shown in Algorithm 4.1. In this way we ensure that no worker can have more than one job rounded up, and the increase to the maximum process time is limited as described in Proposition 2.

Algorithm 4.1 Upper bound computation heuristic (CORF-H)

Require: r_{mk}, p_{mk}
for $m = 1, \ldots, |S|$ **do**
  $i = 1$
  for $k = 1, \ldots, |W|$ **do**
    if $r_{mk} > 0$ **then**
      if $m = |S|$ **then**
        $A_i = k$
        $i = i + 1$
      else
        if $r_{m+1,k} > 0$ or $p_{m+1,k} = 1$ **then**
          $lastWorker = k$
        else
          $A_i = k$
          $i = i + 1$
  if $m \neq |S|$ **then**
    $A_i = lastWorker$
  for $c = 1, \ldots, i$ **do**
    if $r_{mA_c} \times B \neq \lfloor r_{mA_c} \times B \rfloor$ **then**
      $r_{mA_{c+1}} = r_{mA_{c+1}} + r_{mA_c} - (\lfloor r_{mA_c} \times B \rfloor - 1)/B$
      $r_{mA_c} = (1 + \lfloor r_{mA_c} \times B \rfloor)/B$
return r_{mk}, p_{mk}

Proposition 2. *Given a feasible solution of CORF with objective function value z_{CORF}, a valid upper bound UB for WAP can be obtained, for which $UB - z_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$.*

Proof. We prove Proposition 2 by showing that Algorithm 4.1 increases the maximum process time of the solution by at most $\max_{i \in J, k \in W} \{t_{ik}\}$. As stated at the start of Section 4.4, each worker can be assigned at most two fractional jobs. Referring to Figure 4-2,

the only assignments with two fractional jobs are shown in 4-2b and 4-2f. The fractional jobs are always found at the extremities of the assignment, corresponding to the first and last jobs processed by a worker. Algorithm 4.1 will never round up the first job assigned to a worker since the algorithm searches for the worker within an operation who has jobs in the subsequent operation. The algorithm identifies and then labels this worker as *lastWorker* in lines 10 and 15 of Algorithm 4.1. By rounding up at most the last job for each worker, the maximum process time of each worker cannot increase by more than $\max_{i \in J, k \in W} \{t_{ik}\}$. The change in maximum process time from CORF to CORF-H is therefore bounded by $UB - z_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$. □

Given an optimal solution to CORF, a stronger proposition can be proven to hold.

Proposition 3. *If z^*_{CORF} is the optimal solution value of CORF and UB^* is the upper bound obtained using algorithm CORF-H, the optimal solution value of WAP lies in the interval $[z^*_{CORF}, UB^*]$ and $UB^* - z^*_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$.*

Proof. The proof follows from Propositions 1 and 2. □

4.4.2 A lower bound condition on workload duration

Given a valid upper bound UB for WAP, the workload of a given worker can be disregarded if it does not allow the remaining workers to complete their jobs in a time less than UB. More formally:

Proposition 4. *Given a valid upper bound UB for WAP, the workload of worker $k \in W$ that is assigned jobs from i to j; $i \leq j; i, j, \in J$, is suboptimal if*

$$\frac{\sum_{e=0}^{i-1} \min_{k \in W} \{t_{ek}\} + \sum_{e=j+1}^{|J|} \min_{k \in W} \{t_{ek}\}}{|W| - 1} > UB. \tag{4.18}$$

Proof. Inequality (4.18) considers the jobs not assigned to worker k, namely from $0, \ldots, i-1$ and $j+1, \ldots, |J|$. These jobs are assumed to all be completed at the speed of the fastest worker for that job (i.e., the fastest process time) by the remaining $|W| - 1$ workers. If the sum of the process times of these jobs is equally divided across the $|W| - 1$ workers, we obtain a lower bound on the workload duration of the $|W| - 1$ workers. If this lower bound on the remaining workload is greater than UB time units, then the workload assigned to k is suboptimal, because the workload of at least one of the remaining workers would exceed UB. □

4.4.3 Three-index formulation

We now introduce a three-index formulation, which we refer to as $3I$. It uses three-index binary variables y_{ijk} equal to 1 if and only if worker $k \in W$ performs jobs from i to j, $i, j \in J : i \leq j$. The formulation is then

$$
\begin{aligned}
(3I)\ \text{minimise} \quad & z && && (4.19)\\
\text{subject to} \quad & z \geq y_{ijk} \sum_{u \in \{i,\ldots,j\}} t_{uk} && i, j \in J : i \leq j && \\
& && k \in W && (4.20)\\
& \sum_{k \in W} \sum_{i \in \{1,\ldots,e\}} \sum_{j \in \{e,\ldots,|J|\}} y_{ijk} = 1 && e \in J && (4.21)\\
& \sum_{i \in J} \sum_{j \in \{i,\ldots,|J|\}} y_{ijk} = 1 && k \in W && (4.22)\\
& y_{ijk} \in \{0, 1\} && k \in W && \\
& && i, j \in J : i \leq j && (4.23)\\
& z \geq 0. && && (4.24)
\end{aligned}
$$

The objective is to minimise the maximum process time (4.19), in conjunction with Constraints (4.20), which force the duration of each sequence of jobs performed by a worker to be less than or equal to z. Constraints (4.21) impose that each job must be assigned to a single sequence of jobs. Note that overlapping sequences are suboptimal. Constraints (4.22) state that each worker is assigned to one sequence of jobs. Constraints (4.23) and (4.24) define the domains of the variables.

Model $3I$ is a valid formulation for WAP, but its large number of variables makes the formulation impractical for real-sized instances. However, the number of variables can be easily reduced based on the workload duration upper bound found by CORF-H and the workload duration lower bound defined in Section 4.4.2. More precisely, any y_{ijk} leading to a workload longer than the upper bound obtained by CORF-H is eliminated from the formulation, as well as any y_{ijk} for which inequality (4.18) is satisfied. This is done by setting the upper and lower bounds of the variable to zero in our chosen solver.

4.4.4 Exact sequential algorithms

In summary, our 3-Component Sequential Algorithm (3CSA) consists of the following steps:

1. Solve CORF and obtain a valid UB using CORF-H.
2. Find suboptimal workloads according to Inequality (4.18).
3. Solve $3I$, removing the variables with associated workload longer than UB or satisfying Inequality (4.18).

To further strengthen this algorithm, CORF can be initialised with a heuristically calculated upper bound. If this is done we refer to the algorithm as 4CSA.

4.5 Heuristics

We now provide the details of the heuristic implemented at the company, and we describe an Iterated Local Search metaheuristic we have developed.

4.5.1 Heuristic implemented by Calzedonia (IBAH)

The exact algorithms presented in Sections 4.3 and 4.4 were designed in order to assess the performance of the heuristic we implemented in Calzedonia's Sri Lankan factory. This heuristic is based on the previously used manual allocation procedure and is described below.

When performing initial workload allocation, Calzedonia headquarters calculate the size of the batch B as follows. Given the set of workers W, the set of operations S, and the estimated process time $\bar{t}_m, \forall m \in S$, given by the GSD sewing time (GSD, 2017), the batch size can be calculated as

$$B = \left\lfloor \frac{60 \times |W|}{\sum_{m \in S} \bar{t}_m} \right\rfloor . \tag{4.25}$$

Equation (4.25) estimates how many products should be fully completed in an hour if all workers sew at the GSD speed. The value of B typically varies between around 35 and 120 products.

The company allocates the workload by creating a set of sequential blocks of jobs. These blocks contain the maximum number of jobs that are estimated to be completed

in one hour, based on the GSD times. Each new block continues from the final job of the previous block. Next, blocks of jobs are assigned to workers by the line manager by solving a Linear Bottleneck Assignment Problem (Burkard et al., 2012) (LBAP). Hereafter we refer to the heuristic used in the factory as IBAH. The algorithm runs in a fraction of a second, therefore allows the line managers to obtain a workload allocation very quickly whenever they are made aware of absences. This allows the production to be effective from the early hours of the morning. However, the performance of IBAH was never assessed in comparison to other methods.

4.5.2 Bisection Iterated Local Search (BILS)

Our computational results illustrate that the algorithm used by Calzedonia does not return satisfactory solutions, but the long running times of our exact methods are prohibitive for the industrial application. We therefore developed a metaheuristic which runs in much less time than the exact methods, and provides much better solutions than IBAH.

The metaheuristic adopts a simplified solution representation by encoding only the order of workers. A sequence of workers is converted to a full solution with assigned tasks by Algorithm 4.2. This bisection algorithm sequentially allocates tasks to workers. Each worker is assigned the longest possible workload below m_k units of time. The value of m_k is iteratively set to be the average of the best known upper bound UB and lower bound LB on the maximum process time. Algorithm 4.2 runs in pseudo-polynomial time but the procedure proves fast in practice. LB is initialised as the sum of the fastest worker's process time for every task, divided by the number of workers. UB is initialised as the sum of the slowest worker's process time for each task, and is updated by setting it to the best known maximum process time.

The metaheuristic follows the structure of Iterated Local Search presented by Lourenco et al. (2003) and sketched in Algorithm 4.3. The initial solution is a random sequence of workers. The local search consists of two moves: a relocation of one worker in the sequence, and a swap of two workers' positions in the sequence. Each neighbourhood comprises of $|W|^2$ moves, however we only investigate moves affecting the position of the worker with the maximum process time. This reduces the number of moves in the swap neighbourhood to $|W|$, and in the relocate neighbourhood to $2k^*(|W|+1) - 2(k^*)^2 - 2$ where k^* is the position in the sequence of the worker with the maximum process time. Further details are provided in Appendix 4.A.

For the perturbation stage of the ILS there are four moves, the first two are a random

move in the local search neighbourhoods. The third is repositioning a contiguous subsequence of workers of minimum length two and maximum length $|W| - 1$. The final perturbation is reversing the order of a contiguous subsequence of workers of minimum length four and maximum length $|W|$. Each perturbation move has the same probability of being selected, and each move within each perturbation neighbourhood has the same chance of being implemented. We select N_p random perturbation moves at each ILS iteration.

We tested with termination time limits of 30 seconds, one minute, and two minutes. The metaheuristic is henceforth referred to as the Bisection Iterated Local Search (BILS).

Algorithm 4.2 Bisection Algorithm

Require: $UB, LB, SeqW$
$i = 1$
while $UB - LB > 0$ **do**
    Assign a capacity of $m_k = (UB + LB)/2$ to each worker $k \in W$
    for $k = 1$ to $|W|$ **do**
        while $i \leq |J|$ **and** $m_{SeqW(k)} \geq t_{i,SeqW(k)}$ **do**
            Assign job i to worker $SeqW(k)$
            $m_{SeqW(k)} = m_{SeqW(k)} - t_{i,SeqW(k)}$
            $i = i + 1$
    if $i > |J|$ **then**
        $LB = (UB + LB)/2$
    else
        $UB = ((UB + LB)/2) - \min_{k \in W} m_k$
return UB

Algorithm 4.3 Iterated Local Search

$s_0 = GenerateInitialSolution$
$s^* = LocalSearch(s_0)$
while Termination condition not met **do**
    $s' = Perturb(s^*)$
    $s^{*\prime} = LocalSearch(s')$
    $s^* = AcceptanceCriterion(s^*, s^{*\prime})$

4.6 Computational Results

To test our algorithms we generated two sets of 100 instances each, in which all parameters were based on ten sample instances provided by Calzedonia. In both sets the number of operations is $|S| \in \{18, \ldots, 42\}$ and there are four instances for each value of $|S|$. Each operation m is assigned an estimated process time $\bar{t}_m$ in minutes,

by sampling from the normal distribution $N(0.4, 0.2)$, where any values smaller than 0.1 are rounded up to 0.1. The number of workers is generated by sampling from the discrete uniform distribution $U_d[9, \ldots, 15]$. Given these values, the batch size for each instance is calculated using Equation (4.25).

The sets of instances differ in their procedure for generating the t_{ik} values. In our first set, which we name *Average Performance*, all workers are considered to perform their jobs in a time reasonably close to the estimated value $\bar{t}_m$. Our second instance set involves a greater variation in worker skill; we name this set *Mixed Performance*. The t_{ik} values for Mixed Performance are generated by first assigning a skill level to each worker by sampling from the discrete uniform distribution $U_d[0, 2]$. Table 4.1 demonstrates the rules for t_{ik} generation for each instance set, and each skill level within MP. The instances are available for download at `https://people.bath.ac.uk/mb2182/instances/`.

Instance set	Skill level	t_{ik}
Average Performance	N/A	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0, 0.5), 0.1\}$
Mixed Performance	0	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0, 0.5), 0.1\}$
	1	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(-0.1, 0.3), 0.1\}$
	2	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0.1, 0.3), 0.1\}$

Table 4.1: Calculation of t_{ik} for each instance type.

All methods were coded in C++, and CPLEX 12.7.1 was used for solving the MILPs. Tests were run using Balena High Performance Computing (HPC) Service at the University of Bath. Full technical specifications can be found at `https://www.bath.ac.uk/bucs/services/hpc/facilities/`. In what follows, we present aggregate results for the sake of readability. We provide the detailed results in Appendix B. We also note that while the algorithms for ALWABP may be used to solve the WAP, it is unlikely that they perform better, due to the differing features of the WAP that we exploit in our algorithms. A full computational comparison of the algorithms for the ALWABP and the WAP is beyond the scope of this paper.

4.6.1 Size of formulations

To give an indication of the difficulty of each method, we first present the number of variables and constraints in each of the four mathematical models. These figures are stated in Table 4.2 and report the average number of variables or constraints across all 200 instances.

Model	Number of variables	Number of constraints
$2I$	22,481	13,704,843,540
$3I$	21,266,535	21,268,348
$3I$(3CSA)	2,066,537	2,068,351
$3I$(4CSA)	1,812,106	1,813,920

Table 4.2: Number of variables and constraints in each model.

We observe that $2I$ has a number of variables at least two orders of magnitude smaller than any other method. The number of constraints, on the other hand. is the largest of any method due to Constraints (4.3). We also see that using CORF to calculate upper and lower bounds to eliminate variables from $3I$ reduces the number of variables and constraints by an order of magnitude (3CSA). The reduction is more pronounced for 4CSA, which uses the lower bound of CORF and the upper bound of IBAH.

4.6.2 Performance of the exact algorithms

Tables 4.3 and 4.4 summarise the following results:

- **Gap(%):** The average percentage gap between the best lower bound LB and best upper bound UB provided by the solver over all instances for which the solver could provide a lower bound and an upper bound, but were not solved to optimality. Gap(%) is calculated as $\frac{UB-LB}{LB} \times 100$
- **Solved:** The number of instances out of 100 for which the method was able to conclude to optimality within the given computing time.
- **Time(s):** The average time, in seconds, taken to find an optimal solution, calculated using only the instances solved optimally.
- **UB−LB:** The average time unit difference between the lower bound provided by CORF, and the upper bound provided by CORF-H.

Initial testing indicated that both $2I$ and $3I$ were unable to solve instances with the batch size B defined by Equation (4.25). We therefore ran both with batch size $\hat{B} \in \{5, 10\}$ to analyse their performance, we additionally tested $3I$ with $\hat{B} = 20$ as it had exhibited good results for $\hat{B} = 10$. Both formulations were allowed a maximum CPU time of one hour.

From Table 4.3 we see that $3I$ is able to solve more instances than $2I$, but as the batch size increases it becomes unable to converge to optimal solutions within the allowed

	Instance set	B	Gap(%)	Solved	Time(s)
$2I$	Average Performance	5	166.74	1	609.43
		10	296.14	0	N/A
	Mixed Performance	5	125.31	0	N/A
		10	230.38	0	N/A
$3I$	Average Performance	5	N/A	100	163.33
		10	2003.99	53	1401.28
		20	2857.84	0	N/A
	Mixed Performance	5	N/A	100	161.78
		10	2140.84	61	1413.05
		20	2961.99	1	3211.70

Table 4.3: Results of testing $2I$ and $3I$

time. However, the percentage gaps between bounds for unsolved instances are lower for $2I$. Algorithms 3CSA and 4CSA are able to complete instances with batch size B defined by Equation (4.25). For these algorithms we allow one hour of CPU time per formulation, so a full run of 3CSA or 4CSA is allowed two hours of CPU time.

		CORF			CORF–H	$3I$ with reduced variables		
	Instance set	Gap(%)	Solved	Time(s)	UB−LB	Gap(%)	Solved	Time(s)
3CSA	Average Performance	157.19	20	574.67	18.65	146.51	20	1644.14
	Mixed Performance	112.10	19	850.92	16.48	111.04	23	1432.50
4CSA	Average Performance	76.72	38	524.13	9.10	107.24	34	1485.89
	Mixed Performance	69.14	33	445.27	9.92	86.56	31	1549.55

Table 4.4: Results of testing 3CSA and 4CSA

Table 4.4 summarises the results of our computational tests on 3CSA, 4CSA. Clearly, initialising CORF with the solution of our heuristic as an upper bound yields a benefit in the solving of CORF, and this improvement leads to a greater number optimal solutions produced by the reduced $3I$, since the tighter bounds allow more variables to be eliminated. From both Table 4.3 and Table 4.4 we see that the performance of our exact algorithms changes little depending on the instance sets Average Performance and MP.

4.6.3 Performance of the metaheuristic

We have run BILS 10 times on each of the 200 instances. We have used three different time limits for BILS, 30 seconds, one minute, and two minutes, to better observe the trade-off between the CPU time requirement and the performance. Tables 4.5 and 4.6 summarise the results of our testing of BILS. Table 4.5 displays the deviation with respect to the best lower bound for instances of 18 to 24 operations, for which the lower

bound is tight enough to conclude optimality for most instances. The column heading "Avg." refers to the aggregate average deviation from the best known solution for each instance set, whereas "Best" refers to the average of the best deviation achieved by any run of BILS for each instance. Table 4.6 presents the deviations with respect to the best known solution, for all instances. We emphasize that BILS outperforms all of the other algorithms we have presented.

		BILS(30 seconds)		BILS(1 minute)		BILS(2 minutes)	
Instance set	IBAH	Avg	Best	Avg	Best	Avg	Best
Average Performance	483.12	2.28	0.58	1.85	0.65	1.69	0.57
Mixed Performance	345.96	2.31	1.39	1.99	1.36	1.78	1.36

Table 4.5: Average percentage deviation of the heuristic results from the best lower bound, for instances with up to 24 operations.

		BILS(30 seconds)		BILS(1 minute)		BILS(2 minutes)	
Instance set	IBAH	Avg	Best	Avg	Best	Avg	Best
Average Performance	372.48	1.76	0.05	1.35	0.04	0.90	0.02
Mixed Performance	279.02	1.05	0.05	0.84	0.07	0.48	0.00

Table 4.6: Average percentage deviation of heuristic results from the best-known solution value.

4.6.4 Overall performance

	Average solution value				Average reduction from IBAH to:	
Instance set	IBAH	CORF-H	4CSA	BILS(2 minutes)	4CSA	BILS(2 minutes)
Average Performance	121.53	29.96	29.75	27.47	75.52	77.40
Mixed Performance	110.24	33.64	33.53	30.19	69.59	72.61

Table 4.7: Comparisons of feasible solution values at each stage of 4CSA, where the solution values are in minutes.

Finally, we assess the reduction in solution value from IBAH to 4CSA or BILS with a two minute termination condition. These results are presented in Table 4.7. For the 4CSA solution value we took the average best known feasible solution at the conclusion of each stage of the algorithm for each instance set, and for BILS we present the best solution found from the ten runs. This gives the values stated under 'Average solution value (minutes)'. The results obtained from the full run of 4CSA and BILS are better than those found by IBAH, by approximately 70% in both of the instance sets. This enabled us to show Calzedonia that their workload assignment strategy yields far from optimal solutions.

4.7 Conclusions

We have introduced, modeled and solved a new workload allocation problem which arises in practice in the apparel industry. We provided details of the manual allocation methods used by Calzedonia before this study, the heuristic algorithm IBAH. We proposed a mathematical model which can be used to optimally solve the workload allocation, but the model is impractical for real instances. We therefore presented a sequential exact algorithm, based on a relaxation of the original problem, and a metaheuristic algorithm.

Computational results evaluate the efficacy of each method, along with its respective positive and negative qualities. The heuristic IBAH offers production managers the capacity to react and solve short-term issues caused by absenteeism (or other sources of uncertainty in production scheduling) by reallocating rapidly and efficiently the workload, despite not being optimal, while the exact methods can solve small to medium sized instances to optimality within two hours. The metaheuristic algorithm we have developed provides high-quality solutions within very small computing times. It is vastly superior to IBAH and can be applied in an industrial setting.

Our study provides an original contribution to the literature by analyzing a relevant problem encountered in Calzedonia's operations, which has received less attention in workload allocation/balancing studies. In addition, our study offers exact solution methods with high efficiency, and important managerial implications.

We believe that the proposed tool has the opportunity to be adapted to the different Calzedonia production contexts in order to solve country-specific issues. This adaptation represents an interesting future development for our study. Another opportunity for future research is represented by the adaptation of the proposed algorithm not only to facilitate the reallocation in production contexts characterised by an uneven balance of skills among the workers, but also as a tool to support skills improvement and additional focused training. In this context, our exact algorithms could be of practical use, given that computing times are not as stringent in the production planning phases. The algorithm development process highlighted the important role played by workers' skills in the workload reallocation. We believe that future research would provide beneficial improvements in terms of social sustainability in the textile industry, especially considering production contexts such as developing countries.

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) Service at the University of Bath. Thanks are due to the referees for their valuable comments.

Bibliography

Arai, E. (2006). Readymade garment workers in Sri Lanka: Strategy to survive in competition. In M. Murayama (Ed.), *Employment in readymade garment industry in post-MFA era: The cases of India, Bangladesh and Sri Lanka*, Chapter 2, pp. 31–52. JRP Series no. 140. Chiba: Institute of Developing Economies.

Battaïa, O. and A. Dolgui (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics 142*, 259–277.

Becker, C. and A. Scholl (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research 168*(3), 694–715.

Blum, C. and C. Miralles (2011). On solving the assembly line worker assignment and balancing problem via beam search. *Computers & Operations Research 38*(1), 328–339.

Borba, L. and M. Ritt (2014). A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. *Computers & Operations Research 45*, 87–96.

Boysen, N., M. Fliedner, and A. Scholl (2007). A classification of assembly line balancing problems. *European Journal of Operational Research 183*(2), 674–693.

Boysen, N., M. Fliedner, and A. Scholl (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics 111*(2), 509–528.

Brucker, P., R. Qu, and E. Burke (2011). Personnel scheduling: Models and complexity. *European Journal of Operational Research 210*(3), 467–473.

Burkard, R., M. Dell'Amico, and S. Martello (2012). *Assignment problems*. Philadelphia: Society for Industrial and Applied Mathematics.

Chaves, A. A., L. A. N. Lorena, and C. Miralles (2009). Hybrid metaheuristic for the

assembly line worker assignment and balancing problem. *Hybrid Metaheuristics 5818*, 1–14.

De Bruecker, P., J. Van den Bergh, J. Belien, and E. Demeulemeester (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research 243*(1), 1–16.

Fırat, M., D. Briskorn, and A. Laugier (2016). A branch-and-price algorithm for stable workforce assignments with hierarchical skills. *European Journal of Operational Research 251*(2), 676–685.

GSD (2017). General sewing data.

Hardaker, C. and G. Fozzard (1997). The bra design process: A study of professional practice. *International Journal of Clothing Science and Technology 9*(4), 311–325.

Hazir, H. and A. Dolgui (2013). Assembly line balancing under uncertainty: Robust optimization models and exact solution method. *Computers & Industrial Engineering 65*(2), 261–267.

Kelegama, S. (2009). Ready-made garment exports from Sri Lanka. *Journal of Contemporary Asia 39*(4), 579–596.

Li, W., T. Freiheit, and E. Miao (2017). A lever concept integrated with simple rules for flow shop scheduling. *International Journal of Production Research 55*(11), 3110–3125.

Li, X., H. Ishii, and M. Chen (2015). Single machine parallel scheduling problem with fuzzy due-date and fuzzy precedence relation. *International Journal of Production Research 53*(9), 2707–2717.

Lourenco, H., O. Martin, and T. Stutzle (2003). Iterated local search. In *Handbook of Metaheuristics*, pp. 320–353. Springer, Boston, MA.

Miralles, C., J. Garcia-Sabater, C. Andrés, and M. Cardos (2007). Advantages of assembly lines in sheltered work centres for disabled. A case study. *International Journal of Production Economics 110*(1), 187–197.

Miralles, C., J. Garcia-Sabater, C. Andrés, and M. Cardos (2008). Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. *Discrete Applied Mathematics 156*(3), 352–367.

Moreira, M., J.-F. Cordeau, A. Costa, and G. Laporte (2015). Robust assembly line balancing with heterogeneous workers. *Computers & Industrial Engineering 88*, 254–263.

Moreira, M., C. Miralles, and A. Costa (2015). Model and heuristics for the assembly line worker integration and balancing problem. *Computers & Operations Research 54*, 64–73.

Mutlu, O., O. Polat, and A. Supciller (2013). An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research 40*(1), 418–426.

Pereira, J. (2018). The robust (minmax regret) assembly line worker assignment and balancing problem. *Computers & Operations Research 93*, 27–40.

Pereira, J. and E. Álvarez Miranda (2018). An exact approach for the robust assembly line balancing problem. *Omega 78*, 85–98.

Scholl, A. and C. Becker (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research 168*(3), 666–693.

Vilà, M. and J. Pereira (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research 44*, 105–114.

Wickramasinghe, D. and V. Wickramasinghe (2011). Perceived organisational support, job involvement and turnover intention in lean production in Sri Lanka. *The International Journal of Advanced Manufacturing Technology 55*(5), 817–830.

Zacharia, P. and A. Nearchou (2016). A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem. *Engineering Applications of Artificial Intelligence 49*, 1–9.

Appendix

4.A Proof of neighbourhood size

We aim to count the solutions in the relocate neighbourhood which involve a change in the position of the worker with the maximum process time located at position k^*. Here, $|W|$ is the number of workers. Consider a relocate move that places the worker from position $\hat{k}$ to position $\bar{k}$, where $\hat{k} \neq \bar{k}$. The worker in position k^* is affected by the move in three cases:

1. $k^* = \hat{k}$, then $\bar{k}$ can take any value different than k^*. The number of moves to be explored is $|W| - 1$.
2. $\hat{k} < k^* \leq \bar{k}$, then $\hat{k}$ can take $k^* - 1$ values and $\bar{k}$ can take $|W| + 1 - k^*$ values. The number of moves to be explored is $(k^* - 1)(|W| + 1 - k^*)$.
3. $\bar{k} \leq k^* < \hat{k}$, then $\hat{k}$ can take $|W| - k^*$ values and $\bar{k}$ can take k^* values. The number of moves to be explored is $k^*(|W| - k^*)$.

So the total number of moves is $|W| - 1 + (k^* - 1)(|W| + 1 - k^*) + k^*(|W| - k^*)$ which simplifies to $2k^*(|W| + 1) - 2(k^*)^2 - 2$ as stated in the main text.

4.B Detailed computational results

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
1	18	15	13.39	13.45	13.83
2	19	9	**31.52**	**31.52**	**31.52**
3	20	15	19.63	19.72	20.98
4	21	15	**21.15**	**21.15**	21.31
5	22	12	**21.99**	**21.99**	22.14
6	23	10	**29.09**	**29.09**	29.14
7	24	12	**21.50**	**21.50**	21.50
8	25	14	26.76	27.06	27.86
9	26	11	**25.59**	**25.59**	25.78
10	27	13	20.67	24.40	24.40
11	28	11	18.02	26.96	26.96
12	29	14	**25.23**	**25.23**	25.44
13	30	10	24.10	31.08	31.08
14	31	14	**18.42**	**18.42**	18.87
15	32	11	23.45	31.87	31.87
16	33	15	19.35	23.41	23.42
17	34	9	20.74	32.82	32.82
18	35	13	20.73	30.77	31.35
19	36	15	16.77	23.01	23.08
20	37	13	17.43	24.73	24.75
21	38	14	20.77	28.95	29.05
22	39	15	17.92	23.98	23.98
23	40	10	18.29	42.32	42.32
24	41	10	17.42	36.59	36.59
25	42	15	24.17	32.82	33.12
26	18	10	**29.05**	**29.05**	**29.05**
27	19	12	**20.13**	**20.13**	20.50
28	20	11	**20.39**	**20.39**	**20.39**
29	21	9	**28.89**	**28.89**	**28.89**
30	22	10	**34.19**	**34.19**	**34.19**

Table 4.8: Detailed results for instances 1–30 in the Average Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
31	23	11	**30.47**	**30.47**	**30.47**
32	24	14	20.37	20.37	20.88
33	25	11	**23.26**	**23.26**	**23.26**
34	26	10	31.02	34.37	34.37
35	27	14	21.18	21.24	21.29
36	28	14	**22.82**	**22.82**	**22.82**
37	29	11	24.44	33.37	33.37
38	30	10	21.74	29.95	29.95
39	31	14	15.71	18.80	18.91
40	32	15	23.71	26.59	26.95
41	33	14	28.24	28.34	28.50
42	34	10	**36.51**	**36.51**	**36.51**
43	35	9	17.99	37.51	37.51
44	36	9	19.36	39.42	39.42
45	37	10	17.74	33.91	33.91
46	38	15	18.68	29.21	29.94
47	39	15	17.61	22.34	22.58
48	40	10	17.68	33.69	33.69
49	41	9	18.00	37.45	37.45
50	42	15	12.38	25.94	27.15
51	18	10	**30.49**	**30.49**	**30.49**
52	19	13	15.04	15.05	15.06
53	20	12	**23.61**	**23.61**	**23.61**
54	21	13	22.86	23.01	23.40
55	22	15	**19.43**	**19.43**	**19.43**
56	23	14	19.59	19.70	19.71
57	24	11	**27.16**	**27.16**	**27.16**
58	25	9	29.38	32.75	32.75
59	26	14	**20.89**	**20.89**	20.90
60	27	13	**26.18**	**26.18**	**26.18**
61	28	11	**28.14**	**28.14**	**28.14**
62	29	11	**26.19**	**26.19**	**26.19**
63	30	12	21.73	31.23	31.29
64	31	15	17.08	23.37	24.70
65	32	14	**22.77**	**22.77**	23.13
66	33	14	15.73	22.90	22.90
67	34	11	25.51	33.73	34.01
68	35	12	16.38	29.71	29.82
69	36	10	21.54	41.44	41.44
70	37	10	20.11	38.55	38.55

Table 4.9: Detailed results for instances 31–70 in the Average Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
71	38	12	22.26	32.40	33.70
72	39	11	15.50	31.37	31.37
73	40	9	18.81	40.30	40.30
74	41	14	17.63	31.04	31.25
75	42	15	12.48	28.15	28.78
76	18	11	**21.05**	**21.05**	21.42
77	19	10	**17.83**	**17.83**	**17.83**
78	20	11	**23.35**	**23.35**	23.68
79	21	11	**25.26**	**25.26**	**25.26**
80	22	15	16.67	16.74	17.11
81	23	13	**15.30**	**15.30**	15.32
82	24	14	**18.57**	**18.57**	19.49
83	25	15	20.69	20.80	21.91
84	26	15	17.66	17.70	17.91
85	27	12	**27.32**	**27.32**	**27.32**
86	28	13	15.85	23.60	23.73
87	29	9	31.29	31.29	31.29
88	30	13	23.08	25.75	26.11
89	31	13	20.32	22.29	22.31
90	32	14	16.81	21.30	21.59
91	33	15	14.28	19.50	20.40
92	34	9	21.92	41.81	41.81
93	35	13	19.27	27.80	28.26
94	36	11	15.93	32.38	32.38
95	37	12	19.15	32.64	32.81
96	38	14	19.37	26.28	26.41
97	39	14	22.09	30.39	30.58
98	40	15	14.53	27.43	28.34
99	41	14	14.47	27.10	27.50
100	42	10	19.89	46.92	46.92

Table 4.10: Detailed results for instances 71–100 in the Average Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
1	18	11	**28.41**	**28.41**	**28.41**
2	19	13	28.37	28.47	28.54
3	20	13	23.80	23.91	23.91
4	21	12	27.21	27.27	27.30
5	22	14	**18.75**	**18.75**	18.97
6	23	9	**25.91**	**25.91**	**25.91**
7	24	14	22.31	23.38	23.47
8	25	12	26.01	29.13	29.42
9	26	9	**41.67**	**41.67**	**41.67**
10	27	12	**24.63**	**24.63**	24.90
11	28	11	31.10	32.94	32.94
12	29	9	**33.52**	**33.52**	**33.52**
13	30	12	**26.88**	**26.88**	**26.88**
14	31	10	28.77	39.01	39.01
15	32	14	21.87	24.95	24.96
16	33	13	**26.80**	**26.80**	27.29
17	34	13	21.70	27.09	27.95
18	35	14	26.54	31.68	32.20
19	36	10	25.91	33.69	33.69
20	37	13	20.52	23.42	23.77
21	38	13	26.88	32.24	32.24
22	39	9	21.28	38.93	38.93
23	40	14	19.01	30.68	30.92
24	41	12	34.63	34.76	34.76
25	42	10	16.08	32.09	32.12
26	18	10	**27.33**	**27.33**	**27.33**
27	19	10	**28.66**	**28.66**	**28.66**
28	20	11	**25.60**	**25.60**	**25.60**
29	21	13	27.63	27.94	28.03
30	22	9	**30.31**	**30.31**	**30.31**

Table 4.11: Detailed results for instances 1–30 in the Mixed Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
31	23	12	31.62	31.77	31.77
32	24	14	24.98	28.46	28.63
33	25	13	**24.19**	**24.19**	24.21
34	26	11	**23.30**	**23.30**	**23.30**
35	27	11	24.95	33.80	33.80
36	28	11	**29.16**	**29.16**	**29.16**
37	29	14	22.95	23.05	23.47
38	30	13	22.86	35.93	36.28
39	31	10	29.31	36.32	36.32
40	32	14	22.24	29.39	29.65
41	33	10	**32.31**	**32.31**	**32.31**
42	34	15	**30.90**	**30.90**	31.00
43	35	11	22.77	32.19	32.19
44	36	14	21.89	32.58	33.12
45	37	11	23.37	37.88	37.88
46	38	12	18.90	31.35	31.35
47	39	9	22.49	42.94	42.94
48	40	14	18.06	26.98	27.31
49	41	15	16.65	32.34	32.93
50	42	10	17.69	37.65	37.80
51	18	11	**26.27**	**26.27**	**26.27**
52	19	11	**21.11**	**21.11**	21.15
53	20	14	23.74	23.88	24.06
54	21	14	**23.10**	**23.10**	23.27
55	22	15	21.71	21.84	21.87
56	23	13	32.03	32.17	32.61
57	24	14	**22.64**	**22.64**	**22.64**
58	25	15	34.02	34.22	35.12
59	26	11	**28.63**	**28.63**	**28.63**
60	27	11	35.35	36.68	36.68
61	28	11	**24.56**	**24.56**	**24.56**
62	29	14	18.73	22.02	22.26
63	30	11	23.88	35.40	35.40
64	31	14	22.95	26.30	26.33
65	32	12	24.25	35.54	35.89
66	33	9	25.24	43.94	43.94
67	34	15	**25.88**	**25.88**	25.98
68	35	15	17.86	25.98	26.45
69	36	13	18.84	29.88	30.07
70	37	9	**36.35**	**36.35**	**36.35**

Table 4.12: Detailed results for instances 31–70 in the Mixed Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
71	38	12	16.12	31.09	31.09
72	39	9	21.70	40.40	40.40
73	40	10	17.62	38.72	38.72
74	41	13	23.47	38.34	38.85
75	42	14	24.32	36.56	36.64
76	18	12	22.76	22.93	23.09
77	19	15	22.61	22.78	22.78
78	20	13	**22.95**	**22.95**	23.17
79	21	13	22.93	23.00	23.15
80	22	10	**31.36**	**31.36**	**31.36**
81	23	13	**32.60**	**32.60**	32.80
82	24	13	**23.45**	**23.45**	23.58
83	25	14	**22.80**	**22.80**	**22.80**
84	26	12	**29.90**	**29.90**	30.30
85	27	11	**22.40**	**22.40**	**22.40**
86	28	12	25.43	29.10	29.10
87	29	14	21.24	27.69	28.40
88	30	13	**25.16**	**25.16**	**25.16**
89	31	11	**26.52**	**26.52**	**26.52**
90	32	14	21.13	26.45	26.62
91	33	15	16.16	22.22	22.56
92	34	12	18.74	33.18	33.66
93	35	10	19.33	38.04	38.04
94	36	12	21.76	28.45	28.50
95	37	12	27.04	39.53	39.53
96	38	12	14.57	35.97	35.97
97	39	9	22.34	41.99	41.99
98	40	13	21.86	33.95	33.95
99	41	14	15.74	27.86	28.13
100	42	10	21.90	41.22	41.22

Table 4.13: Detailed results for instances 71–100 in the Mixed Performance set

Chapter 5

Conclusions and Future Research

5.1 Summary of thesis

This thesis first described the motivation and decision making behind the route I have taken to complete my PhD. I discussed the point at which I initially decided to take on a PhD, and the reasons I took on each research project. I then showed the three papers that I have produced in the course of my PhD studies.

Chapter 2 introduced the Twin-Robot Palletising Problem, in which two robots situated on the same rail had to be scheduled to pickup and deliver products from/to specified rail locations. We developed two exact algorithms, and two metaheuristic algorithms, eventually concluding that the best algorithm for practical implementation would be a Hybrid Genetic Algorithm.

Chapter 3 introduced the Twin-Robot Pallet Assignment and Scheduling Problem, which generalised the problem from Chapter 2. Instead of products having a predefined delivery location on the rail, their delivery point is left to be solved. We developed a mathematical model and four metaheuristic algorithms, again concluding that a Hybrid Genetic Algorithm would be most appropriate for implementation.

Chapter 4 introduced the Calzedonia Workload Allocation Problem, in which sewing tasks had to be assigned to a group of workers. We developed two exact algorithms: one mathematical model, and one sequential algorithm consisting of a linear relaxation, bound finding heuristics, and a mathematical model. As well, we developed two heuristics. Concluding that a Bisection based Iterated Local Search was best for Calzedonia to implement.

5.2 Key contributions

The main contribution of this thesis is that it offers methods for solving previously unsolved problems. The key contributions of each paper are as follows:

- In Chapter 2 we demonstrate that adding a second robot to a single-robot system can lead to reducing the makespan by more than 50%.
- In Chapter 3 we demonstrate that in the same twin-robot system, incorporation of the assignment of pallets into the scheduling decisions yields on average 20% improvement in makespan.
- In Chapter 4 we show the algorithm implemented by Calzedonia in their factory, and demonstrate new algorithms that could reduce their working time by over

70%.

All of these findings relate to systems that we have found to exist in industry, and could be used to dramatically improve the efficiency of either type of system. The improved efficiency could be utilised to reduce working time, energy expenditure, or costs. It could also be used to increase number of orders taken or total system throughput. Additionally, the methodological highlights that I wish to reinforce are:

- The development of the scheduling coefficients in Section 2.2.2. These coefficents were our first major breakthrough with the TRPP, and were subsequently used in every algorithm of the two papers regarding twin-robots.
- Algorithm 4CSA presented in Section 4.4. The combination of an initial heuristic to calculate an upper bound, a linear relaxation to find a lower bound, a further heuristic to redefine the bounds based on the solution to the linear relaxation, and finally an MILP to solve the bounded problem. The combination of these methods is an interesting way to go about tackling a problem of this type.

5.3 Limitations

One key limitation of the problems solved in this thesis, is that it is not practically possible to test every type of algorithm to see which is best. We drew our conclusions about which we believe is best based on the tests we could reasonably run, but could not confidently say that the final algorithms we suggest are definitely the best method for solving the problems. In the following section I discuss this fact further.

Additionally, all methods developed in this thesis are incredibly well focused on a very specific problem setup. While possible, it is unlikely that these methods would be applied to other problems. It would be an interesting next step to attempt to develop methods which have wider applicability to a group of problems, instead of a single problem structure.

In addition, as with any new researcher, each of my papers was limited by the knowledge I had and could access at each stage. The PhD has of course been a huge learning process, and if I was introduced to the first problem today, with the algorithmic knowledge I now have, I would take a different approach to it.

5.4 Future work

As with most PhD students, the task of putting together a thesis has made me question where my interests lie within a huge field of research. I have found that the specific problem I am solving is not of high importance to me, it is the fact I am solving a problem that inspires me. For this reason I realised that I am not a researcher who wishes to focus on one particular problem with granular detail. Instead, the part of research that interests me is developing methods. In each paper I have completed to date I have tried to develop more powerful algorithms than I did in the previous paper. In future research I would like to continue this trend, particularly with metaheuristics. In Chapter 3 I started to explore the implementation of parallel computation in metaheuristics, and would like to further explore these methods. My three key objectives for next steps as a researcher would be:

- Explore a greater range of metaheuristic algorithm structures. My favourite algorithm type to build at the moment is Hybrid Genetic Algorithms. I would like to explore more types of metaheuristics to broaden my abilities in this area.
- Continue to incorporate CPU parallel computation into my algorithms. I would aim to develop more complex interactions between parallel threads, to design algorithms that rely fully on the parallel elements, and could not be run in serial.
- Explore the potential of GPU parallel computation for my algorithms. GPUs usually have far more computation cores than a CPU. While these cores are often less powerful than CPU cores, the sheer number of them provides interesting opportunities for parallel computation of small tasks, with huge potential for interaction.