Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

AUTOMATICA E RICERCA OPERATIVA

Ciclo 29^

Settore Concorsuale di afferenza: A1/06

Settore Scientifico disciplinare: MAT/09

Combinatorial Optimisation Problems in Logistics and Scheduling

Presentata da: ALBERTO MARIA SANTINI

Coordinatore Dottorato

Prof. Daniele Vigo

Relatore

Prof. Silvano Martello

Relatore

Prof. Daniele Vigo

Esame finale anno 2017

PhD Thesis

Combinatorial Optimisation Problems in Logistics and Scheduling

Alberto Santini

2017

Department of Electrical, Electronic, and Information Engineering
University of Bologna

The present work is dedicated to:
my parents,
my wife and her family,
my mentors Daniele and Silvano,
all the colleagues of the O.R. group at the University of Bologna.

Contents

1 Introduction

This thesis presents a variety of problems and results in the fields of logistics and, in particular, of maritime and railways logistics. In Chapter 1 we first give a general overview of these areas in general, and of the problems discussed in this work in particular. We also aim to highlight the importance of these problems and how they contribute in achieving high-impact goals, such as reducing the environmental footprint of moving goods and people on transport networks. We then proceed to briefly review the tools used in the rest of the thesis. The problems considered, in fact, have been tackled with both exact and heuristic methods, and often with a combination of both.

The rest of the thesis presents one problem per each chapter. Each problem corresponds to a research paper, either published or submitted to peer-reviewed journals. We decided to keep the internal structure of the chapters as similar as possible to that of the original papers; in this way, each chapter is self-contained and can be read separately. This, of course, introduces some repetition between chapters, for which we apologise to the reader.

1.1 Topics

1.1.1 Maritime logistics

The International Chamber of Shipping [23] estimates that around 90% of world trade has been done by sea in 2015. This figure was of 75% in 2008 [28] and 85% in 2013 [13]. There are roughly 50 000 merchant ships operating worldwide, manned by more than one million seafarers, generating around $380 billion in annual freight rates. Even though the global economic crisis has slowed down growth in the global trade of merchandise, Asariotis et al. [1] assess that in 2015 the amount of goods shipped by sea still grew by 1.4%, surpassing for the first time the 10 billion tonnes mark. At the same time, the investment in new infrastructure is steady growing, as witnessed by the recent expansion works on the Suez and Panama canals. And investment in the global fleet do not lag behind, as in 2015 the world fleet grew by 3.5% in terms of deadweight tonnes [1].

The economical incentive to optimise the maritime supply chain, therefore, remains strong if operators want to maintain profitability in spite of lower rates. In 2015, for example, almost all shipping segments butandor — depending on the point of view — oversupply of capacity. One of the most affected segment has been that of container shipping: if it grew by 6.1% in 2014, by 2015 the growth slowed down to 2.9%, corresponding to around 175 million TEU[1] shipped [1]. At the core of this slow-down were a decrease in demand on intra-Asian and Asia-Europe routes.

[1] Twenty-foot Equivalent Unit, corresponding to the volume of a standard-sized container 20 feet long.

Excessive capacity is also a problem in container shipping: acconrdig to Davidson [11], the average ship size increased by an astounding 18.2% in 2010–2015. For example, Mærsk Line (the world's biggest container shipping operator) has introduced in 2013 a new class of vessels, the *Triple-E*, with a capacity of 18340 TEU [30]. These were the largest container ships ever built at the time of their introduction, only to be surpassed by China Shipping Container Lines' new *Globe* vessel, which can carry 19100 TEU [31]. This big vessels allow for better economies of scale, by having fewer of them and sailing more slowly. However, there is al imit on how much a ship size can grow, before it becomes impossible to operate at most ports. Vessels like the Globe, for example, can only be employed on the Asia-Europe route: no American port has enough space or drought to let them in.

Given the enormous volumes traded, and the corresponding revenue earned, it is clear that any improvement in maritime supply chain can have a big impact on the profitability of the operators. There is, however, another important reason for increasing efficiency in maritime transport: the World Shipping Council [43] estimated that 2.7% of global greenhouse gas emissions is accounted by international maritime shipping, and a quarter of this figure is due to container shipping. The minimisation of the impact of shipping on the environment has recently increased on a regulatory level, e.g., by banning particularly polluting types of fuel. But the issue has been tackled also from the point of view of maritime optimisation: the two recent reviews by Christiansen et al. [7], Wang et al. [42] dedicate a large section to problems which focus on or, at least, include the problem of the environmental impact of shipping, and the "*Green Ship Routing Problem*" [25] has been formalised and is now increasingly studied in the literature.

In this thesis we focus on two aspects of the optimisation of the maritime supply chain: **landside** and **seaside** maritime logistics. While seaside logistics is concerned with all the aspects which directly involve routing a vessel at sea, landside logistics involve all the infrastructure which forms the interface between the ship and the rest of the supply chain. Examples of landside logistic problems are defining optimal routes for ships, given a set of ports they have to call; optimising their sailing speed profile; balancing the load onboard, in order to reduce drag and improve stability, etc. Notable landside problems include the assignment of vessels to berths, the assignment of quay cranes to vessel services, and the routing of containers in the port yard.

We refer the reader to Panayides and Song [37] for an introduction to maritime logistics, to Christiansen et al. [5, 6, 7] for reviews on maritime routing and scheduling problems, and to Psaraftis and Kontovas [38] for a survey on speed optimisation in vessel routing. Regarding landside problems, we refer to the excellent survey by Steenken et al. [40] on container terminals (and to the work of Vis and De Koster [41] for transhipments at container terminals, in particular) and to Bierwirth and Meisel [3] for an overview of berth and quay crane assignment problems.

Chapters 2 and 3 deal with two landside problems, namely quay crane assignment and berth allocation; Chapters 4 and 5, on the other hand, tackle two seaside problem: the Feeder Network Design Problem, a strategic problem asking to find a set of routes for a fleet of vessels which maximises the operator's revenue, and the Travelling Salesman Problem with Draught Limits, which seeks the optimal route for a single vessel, taking into account that loading more cargo in the vessel also increases the amount of draught that it needs in order to enter

a port.

1.1.2 Railway transport

Railway transport involves the movement of people or goods on trains. It is, therefore, usually classified in two macro-areas: **passenger rail transport** and **freight rail transport**. Passenger transport is on the rise: Eurostat [15], for example, reports an average increase of +1.8% passenger-kilometres in 2015 in the European Union, with peak increases of 34% in Slovakia, 18% in Greece, and 15% in Luxembourg. Globally, China, India and Japan lead the way [35], with a comined total of about 2400 billion passenger-kilometres.

Freight rail transport statistics, however, tell a different story. A trend similar to that analysed in Section 1.1.1 for container shipping has emerged in the years following the global economic crisis, which has seen the amount of goods shipped by train decrease sharply. Eurostat [15], however, reports that growth in freight rail traffic has now restarted in 12 EU countries already in 2014, starting with Germany and France (+1.7 billions tonne-kilometres), followed by Romania (+1.4 billions). On the other hand, Finland and Sweden saw a steep contraction during the same period. In a medium-term perspective, however, rail freight has gained share in the EU-28 countries, passing from 16.9% of all inland freight transport in 2009 to 18.4% in 2014 [14]. Globally, it is the United Stats to lead the league, with their 2704 billions tonne-kilometres in 2015, followed by China, Russia, India, and Canada [35].

These numbers show both that rail transport is a crucial part of the global transportation infrastructure, and that it is increasingly the mean of transport of choice for both passengers and freight. The tranportation of passengers, in particular, involves additional challenges, because a succesfull passenger railway system has stringent requirements in terms of punctuality, frequency, connectivity, and resilience. In this thesis we are going to study the problem of rescheduling passenger trains, i.e. of finding appropriate countermeasures when an unforseen event forces the train operator to depart from its normal operating shedule. This type of problem is now increasingly studied: so much so that "train rescheduling" has become a well-defined category of optimisation problems. The reader is referred to the excellent survey by Cacchiani et al. [4] for an overview on train rescheduling algorithms.

1.2 Methodological toolbox

The three main approaches to the solution of a combinatorial optimisation problem consist in using either exact, approximate, or heuristic methods. **Exact** algorithms provide the guarantee that an optimal solution (if any) to the problem will be found in bounded time. For most interesting problems, however, this bound is often super-polynomial. Classical combinatorial problems, such as the Graph Colouring Problem, the Hamiltonian Path Problem, the Minimum Spanning Tree Problem, the Knapsack Problem, the Quadratic Assignment Problem, are all $\mathcal{NP}$-complete, meaning that the running time of any exact algorithm will grow at least exponentially with the input size.

For many $\mathcal{NP}$-hard optimisation problems, then, we often have to be content with a solution which is not optimal. Algorithms that produce such solutions are typically classified

as approximate or heuristic. An **approximate** algorithm is an algorithm that produces a solution of provable minimum quality. This means that a mathematical proof is available that the ratio between the value of the solution provided by the algorithm and the value of the optimal solution is bounded by a constant (assuming a minimisation problem), called the approximation ratio. Measures of quality for these algorithms are, e.g., the ratios in the worst or the average case.

In order to have a proof of the quality produced by an algorithm, we often need such algorithm to be simple enough to be studied from a mathematical, combinatorial, geometric, or probabilistic point of view. On the other hand, many well-perofrming non-exact algorithms for combinatorial optimisation problems are too complex to be studied in such a way. In this case, we talk about **heuristic** algorithms: they produce solutions with no theoretical quality guarantee whatsoever, but which are very (or, at least, reasonably) good in practice.

1.2.1 Exact methods

The exact algorithms used to tackle the problems presented in this thesis have, as their ultimate outcome, that of solving a Mixed-Integer Programme (MIP). The most widely used such algorithm is the **branch-and-bound** algorithm. This algorithm explores the solution space by traversing a tree. Each node of the tree represents a more constrained version of the original problem and, therefore, has to explore a smaller subset of the solution space. For example, when solving a 0-1 problem, the solution space can be partitioned into two halves, by considering the two subproblems where the value of a binary variable has been fixed, repsectively, to 0 and 1. These two subproblems will correspond to two child nodes of the root of the tree. By proceeding with further partitioning, the leaves of the tree represent solutions where all variables are fixed to specific values. Exploring the full tree, therefore, would correspond to a complete enumeration of the solution space.

The advantage of using a branch-and-bound algorithm, however, lies precisely in the fact that the whole tree need not be explored. Consider, for example, a bounded 0-1 problem (P01) in minimisation form:

$$\min \quad c^{\mathrm{t}}x \tag{1.1}$$

$$\text{s.t.} \quad Ax \geq b \tag{1.2}$$

$$x \in \{0,1\}^n \tag{1.3}$$

Notice that the objective value $c^{\mathrm{t}}\hat{x}$ of a feasible solution $\hat{x}$ to (P01) always provides an upper bound on the optimal objective value. On the other hand, the optimal solution to a relaxation of (P01) — for example, to its linear relaxation (P01L) — provides a lower bound on the optimal objective value. During the exploration of a node of the branch-and-bound tree, suppose we have obtained an uper bound UB (e.g. by reaching a leaf, or by means of a heuristic) and, solving (P01L) at the node, we obtain a lower bound LB $\geq$ UB. Since by further constraining the problem, i.e. by fixing more variables, the lower bound produced in the subtree of the current node can only increase, we are confident that we will not find any leaf in such subtree with a better upper bound than the one we already have. For this reason, we can prune the current node and its whole subtree. Analogously, we can prune the tree when we reach a node where the problem is infeasible.

This algorithm was first proposed by Land and Doig [26] and got its current name when it was applied to the solution of the Travelling Salesman Problem (TSP) by Little et al. [27]. The branch-and-bound method is extremely effective and, therefore, has not only theoretical but only practical value, being the underlying algorithm in many commercial MIP solvers.

But another crucial component in the solution of a combinatorial problem is the choice of the MIP model used to represent the problem mathematically. Two different MIP formulations for the same problem can have dramatically different mathematical and combinatorial properties (e.g. the strength of the relaxation, the presence of symmetry) that affect how effectively they can be solved by applying a branch-and-bound algorithm. The most evident of these properties is arguably the model size. With this respect, we can classify MIP formulations into compact and extended. **Compact formulations** are those for which the size of the model is polynomial in the size of input data. By size of the model we mean the size of its constraint matrix; e.g. in the case of (P01) this would be the dimension of the space to which matrix A belongs. On the other hand, **extended formulations** are those for which the model size is super-polynomial in the size of the input data. We refer the reader to, e.g., Conforti et al. [8] to a summary of the differences between these two types of formulations.

Branch-and-price

Consider the linear relaxation (P01L) of (P01), and assume we are in the case where (P01L) is bounded and the number of columns of A is exponential in the size of the input. This means that, for a large enough instance of the problem, even inputting (P01L) to a computer solver would take a considerable amount of time, let alone working towards the solution of the associated minimisation problem. In other words, even the enumeration of the columns of A is not viable.

Let K be the set of columns of $A \in \mathbb{R}^{n\times m}$ (therefore $|K| = m$). Consider a smaller subset $K' \subset K$ containing only a few of the columns of A, and let (RP01L) be the version of (P01L) where only the columns of K' are considered:

$$\min \quad \sum_{k\in K'} c_k x_k \tag{1.4}$$

$$\text{s.t.} \quad \sum_{k\in K'} a_{hk} x_k \geq b_h \qquad \forall h \in \{1,\dots,n\} \tag{1.5}$$

$$x \in \mathbb{R}^n_+ \tag{1.6}$$

Let $\hat{x} \in \mathbb{R}^n_+$ be the optimal solution of (RP01L), which we also call the *restricted* problem, and let $\pi_h \geq 0$ be the dual variables associated with Eq. (1.5) in its $\geq$-form. To solve the original unrestricted programme, we would like to identify which columns in $K \subseteq K'$ should enter the base of (RP01L) in order to improve the upper bound. We would then only add those columns, with the hope that we can prove the optimality of (P01L) by moving into K' only a small set of columns from $K \setminus K'$.

Recall from dual theory, that a column missing from the base of the primal problem corresponds to a violated inequality in the dual problem. The inequalities in the dual of (RP01L)

are

$$c_k - \sum_{h=1}^{n} a_{hk}\pi_h \geq 0 \quad \forall k \in K' \tag{1.7}$$

and therefore, a solution $k^* \in K \setminus K'$ should enter the base iff

$$\hat{c}(k^*) := c_{k^*} - \sum_{h=1}^{n} a_{hk^*}\pi_h < 0 \tag{1.8}$$

where $\hat{c}(k^*)$ is called the **reduced cost** of k^*. When we can prove that no column in $K \setminus K'$ has negative reduced cost, again from dual theory, we know that the original unrestricted problem (P01L) has been solved to optimality. In order to have a working algorithm, therefore, we also need a method to generate new columns with negative reduced cost. Such a method is called a **pricing algorithm**, and is heavily dependent on the nature of the problem we are dealing with. A desirable characteristic of the pricing algorithm is that it is able to find new columns with negative reduced cost (or to prove that none exist) in short time, and ideally in polynomial time.

By solving iteratively the linear relaxation of the reduced problem (also called the **master problem**), and the pricing problem, we obtain an algorithm for the solution of (P01L), which takes the name of a **column generation** algorithm. This method to solve a linear problem with a potentially exponential number of variables was introduced by Ford and Fulkerson [16] and succesfully employed for the first time by Gilmore and Gomory [17, 18].

When we are interested in solving an integer or mixed-integer programme, we can then embed the column generation approach within a branch-and-bound algorithm: at each node of the tree, the linear relaxation at that node is solved by means of column generation. Such a combined algorithm is called a **branch-and-price** algorithm. This combination of algorithms is not straightforward. The main problem lies, in fact, in the effect of branching decisions to the master and pricing problems. It is well known (see, e.g., Barnhart et al. [2]) that a simple branching rule that fixes the values of the variables in the master problem is often problematic to enforce in the subproblems. In many cases, therefore, alternative branching strategies have to be devised, which partition the solution space in ways other than fixing the variable values. We refer the reader to Lübbecke and Desrosiers [29] and Desrosiers and Lübbecke [12] for further introductory material on column generation and branch-and-price algorithms.

Branch-and-price algorithms are used in Chapters 3 and 4 to provide exact solutions, respectively, to the Partition Colouring Problem — also used to model a Berth Allocation Problem — and to the Feeder Network Design Problem — arising in maritime seaside logistics. In the first case, the number of columns is explonential, as it is the number of (maximal) stable sets in a graph; in the second case, because each column represents a (feasible) route of a vessel, i.e. a (feasible) sequencing of port visits.

Branch-and-cut

We now consider the related case in which (P01L) is bounded, but it's the number of rows of A to be exponential in the size of the input. We employ a similar approach, and consider only

a subset of row, i.e. a subset $N \subset \{1, \dots, n\}$ of constraints, producing formulation (CP01L):

$$\min \quad \sum_{k \in K} c_k x_k \tag{1.9}$$

$$\text{s.t.} \quad \sum_{k \in K} a_{hk} x_k \geq b_h \qquad \forall h \in N \tag{1.10}$$

$$x \in \mathbb{R}^n_+ \tag{1.11}$$

Since we removed some constraints, (CP01L) is a relaxation of the original problem (P01L). Therefore, if the optimal solution x^* to (CP01L) also satisfies the removed constraints, then it is also the optimal solution for (P01L). Otherwise, we will have to identify which removed constraint is violated by x^*; we can then add it to the model, and resolve. This iterative process is commonly called a **cutting planes** algorithm. The problem of identifying which implicit constraint is violated by a solution of (CP01L), or to prove that none are, is named the **separation problem**. As in the case of the pricing problem, we would like the separation problem to be quick (hopefully polynomial) to solve, and we have the hope that the optimal solution to (P01L) is found by separating only a small number of constraints. The cutting plane algorithm was introduced by Kelley [24].

Notice that, in principle, it is also possible to separate inequalities which are not required to produce a feasible solution, but are nonetheless valid: the idea that a linear formulation could be strenghtened by introducing extra constraints was pioneered by Gouonr [22]. These extra constraints take the name of **valid inequalities** and, depending on their number, can either be added to the original formulation, or separated using a cutting plane algorithm.

When we are solving and integer or mixed-integer programme, similarly to what done for column generation, we can embed a cutting plane algorithm into the exploration of the branch-and-bound tree, thereby using a **branch-and-cut** algorithm. Notice that the correctness of the overall algorithm is guaranteed by separating violated inequalities just for the integer solutions, but convergence can be accelerated if the separation problem can be used to derive inequalities violated by fractional solutions as well.

In the first implementations (see, e.g., Crowder and Padberg [9], Crowder et al. [10]) cutting planes were used only at the root node of the branch-and-bound tree; such approach is now called *cut-and-branch*. The first actual implementation of a branch-and-cut algorithm was presented by Padberg and Rinaldi [36] to solve the Travelling Salesman Problem (TSP). We refer the reader to Mitchell [32] for a general overview on branch-and-cut algorithms.

A branch-and-cut algorithm is used in Chapter 5 to solve a variant of the Travelling Salesman Problem. Inequalities corresponding to subtour elimination constraints are in exponential number, as there is one of them for each possible subset of the set of nodes, and are therefore added only when a violated one is found in any optimal solution to the linear relaxation of the problem. Furthermore, a number of valid inequalities are also separated and added to the model, in order to strenghten the formulation.

1.2.2 Metaheuristics

Metaheuristics are paradigms used to create heuristic algorithms. In the words of Glover and Kochenberger [21],

> "Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes."

In this work, in particular, we are going to use a variety of metaheuristic paradigms: Tabu Search (TS, Glover [19, 20]), Reduced Variable Neighbourhod Search (RVNS, Mladenovic [33], Mladenović and Hansen [34]), Adaptive Large Neighbourhood Search (ALNS, Ropke and Pisinger [39]).

These three metaheuristics offer three different solutions to the problem highlighted by Glover and Kochenberger: local search improvements lead to find solutions which are local optima, potentially very far away from the global optimum. A feasible solution x_0 is improved with local search by considering a neighbourhood $N(x_0)$ to explore, and choosing the best solution $x_1 \in N(x_0)$. If the problem involves the minimisation of an objective function $f(x)$, then x_1 can be chosen as $x_1 = \text{BEST}(N(x_0)) := \arg\min_{x \in N(x_0)}\{f(x)\}$. This procedure can be iterated by considering $x_2 = \text{BEST}(N(x_1))$, etc. When we reach a local optimum $x_k = \text{BEST}(N(x_k))$, the algorithm must then terminate.

The basic idea behind TS is that local optima can be escaped from, by allowing non-improving moves. One could set, for example, x_{k+1} as any solution in $N(x_k)$ taken at random, and not necessarily the best one (which would coincide with the local optimum x_k). This approach, however, has a clear disadvantage: most of the time we will have that $x_k = \text{BEST}(N(x_{k+1}))$, thereby cycling back to solution x_k and never escaping the "valley" surrounding the local optimum. TS proposes to overcome this limitation by introducing a short-term memory of moves to forbid, thereby placing them in a **tabu list**. The definition of *move* can be problem-dependent; it is important, however, that forbidding a move achieves the desired outcome of forbidding the return to a recently-visited local optimum. Since the memory is short-term (not to reduce too much the solution space), we only place a move in the tabu list for a certain limited number of iterations; this number is known as the **tabu tenure**.

The RVNS metaheuristic, on the other hand, aims at escaping from local minima by exploring increasingly larger neighbourhoods. In this case, instead of defining a single neighbourhood $N(x)$, we define a succession of them: $N_1(x), \ldots, N_k(x)$. These neighbourhoods are nested, i.e. for all points x of the solution space, $N_1(x) \subset N_2(x) \subset \ldots \subset N_k(x)$. Since the size of a neighbourhood N_h can become very large as h increases, they are not explored completely but rather sampled. In this work, we only consider one sample from each neighbourhood: if the sample provides a better objective value than the current solution, it is accepted; otherwise, we sample the next (larger) neighbourhood.

ALNS, finally, is rooted in the idea that, when multiple neighbourhoods are available, the same neighbourhood can be effective for one instance and ineffective for another. Therefore,

the choice of neighbourhood to use in each iteration of the heuristic should depend on its past performance during the solution process of the current specific instance. In particular, ALNS neighbourhoods are defined implicitely as $N_{dr}(x) = r(d(x))$. $d(\cdot)$ is a *destroy method* which takes a feasible solution as input, and destroys part of it, returning a potentially unfeasible one; $r(\cdot)$ is a *repair method* which takes a destroyed solution and repairs it, producing a feasible solution. If each repair method is able to repair solutions destroyed by each destroy method, then we will have one neighbourhood N_{dr} for each possible combination of destroy and repair methods. ALNS will then try to evaluate the destroy and repair methods separately, rather than giving an explicit evaluation of the neighbourhood. This is done by defining a score for each method and increasing it every time the method is involved in the production of an improving solution, while decreasing it if the solution is worse than the current one. At each iteration, then, the methods are selected randomly with a probability proportional to their score, thus favouring methods which have "behaved well" for the instance at hand.

The roles these metaheuristics play in the present work are many: we use them to generate starting solutions to exact algorithms, to efficiently explore the solution space of a problem, and we even study their methodological properties without the explicit aim of solving any particular problem. TS is used in Chapter 3 to produce initial solutions for the Partition Colouring Problem; however, we show that ALNS produces better results in a shorter time. This result is particularly interesting, because traditionally ALNS has proven effective in solving "rough landscape" problems, such as Vehicle Routing variants: problems where the number of possible objective values is very large (essentially of the same order of the number of solutions). The Partition Colouring Problem, on the other hand, has a very flat landscape with a few discrete possible values for the objective function, and moving from a solution to one with a better objective value is difficult. To this end, we employed a new acceptance criterion (a criterion to decide wether a new solution should be kept or discarded) which plays well with flat-landscape problems. TS is furthermore used in Chapter 5 to produce initial solutions to the Travelling Salesman Problem with Draft Limits.

TS and RVNS are also employed in Chapter 6 as two alternative strategies to decide in which order the subproblems of a decomposed problem should be solved, keeping in mind that the solution of a previous subproblem reduces the solution space of the following ones. In particular, the problem of rescheduling a set of trains is decomposed train-by-train; scheduling one train marks certain resourced (tracks, platform) as inaccessible for trains scheduled afterwards. The main idea of the algorithm is to produce greedy schedules for each train in sequence, and then perturb their order and re-run the greedy algorithm. TS and VNS come into play when deciding how the order pertubation should be made, in order to find a good compromise between running times (this real-time algorithm should produce a solution in under 2 seconds) and solution quality.

Finally, Chapter 7 investigates the impact of different acceptance criteria on ALNS, and reports results obtained trying the different critera on two relevant optimisation problems: the Capacitated Vehicle Routing Problem, and the Capacitated Minimum Spanning Tree Problem.

Bibliography

[1] Regina Asariotis, Hassiba Benamara, Jan Hoffmann, Anila Premti, Vincent Valentine, and Frida Youssef. Review of maritime transport, 2016. Technical report, United Nation Conference on Trade and Development, 2016.

[2] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

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

[4] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[5] Marielle Christiansen, Kjetil Fagerholt, and David Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

[6] Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Maritime transportation. *Handbooks in operations research and management science*, 14:189–284, 2007.

[7] Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228 (3):467–483, 2013.

[8] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR: A Quarterly Journal of Operations Research*, 8(1): 1–48, 2010.

[9] Harlan Crowder and Manfred W Padberg. Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509, 1980.

[10] Harlan Crowder, Ellis L Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[11] Neil Davidson. Juggling bigger ships, mega-alliances and slower growth, 2016. Terminal Operations Conference Europe, Hamburg.

[12] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[13] Drewry Maritime Research. Seaborne Trade Annual Report 2013. Technical report, Drewry, 2014.

[14] Eurostat. Energy, transport and environemtn indicators. Technical report, 2016. URL http://ec.europa.eu/eurostat/documents/3217494/7731525/KS-DK-16-001-EN-N.pdf.

[15] Eurostat. Railway passenger transport statistics: quarterly and annual data 2016. Technical report, 2016.

[16] Lester Randolph Ford and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[17] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[18] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem – Part II. *Operations research*, 11(6):863–888, 1963.

[19] Fred Glover. Tabu search — part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[20] Fred Glover. Tabu search — part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[21] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[22] RE Gouonr. Outline of an algorithm for integer solutions to linear programs. *Bull. Am. Math. Soc*, 64:3, 1958.

[23] International Chamber of Shipping. Shipping and world trade, 2017. URL http://www.ics-shipping.org/shipping-facts/shipping-and-world-trade.

[24] James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[25] Christos A Kontovas. The green ship routing and scheduling problem (gsrsp): a conceptual approach. *Transportation Research Part D: Transport and Environment*, 31:61–69, 2014.

[26] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[27] John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[28] Lloyd's Marine Intelligence Unit. Measuring Global Seaborne Trade. Technical report, Lloyd's, 2009.

[29] Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[30] Mærsk Line. Mærsk Triple-E, 2017. URL http://www.maersk.com/en/hardware/triple-e.

[31] MarineTraffic. CSCL Globe, 2017. URL http://www.marinetraffic.com/en/ais/details/ships/shipid:993261/mmsi:477712400/imo:9695121/vessel:CSCL_GLOBE.

[32] John E Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, pages 65–77, 2002.

[33] Nenad Mladenovic. A variable neighborhood algorithm — a new metaheuristic for combinatorial optimization. In *Abstract of papers presented at Optimization Days*, page 112, 1995.

[34] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[35] International Union of Railways. Railways statistics. Technical report, 2016. URL http://www.uic.org/IMG/pdf/synopsis_2015_print_5_.pdf.

[36] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[37] Photis M Panayides and Dong-Wook Song. Maritime logistics as an emerging discipline. *Maritime Policy & Management*, 40(3):295–308, 2013.

[38] Harilaos N Psaraftis and Christos A Kontovas. Speed models for energy-efficient maritime transportation: A taxonomy and survey. *Transportation Research Part C: Emerging Technologies*, 26:331–351, 2013.

[39] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4): 455–472, 2006.

[40] Dirk Steenken, Stefan Voß, and Robert Stahlbock. Container terminal operation and operations research: a classification and literature review. *OR spectrum*, 26(1):3–49, 2004.

[41] Iris FA Vis and Rene De Koster. Transshipment of containers at a container terminal: An overview. *European journal of operational research*, 147(1):1–16, 2003.

[42] Shuaian Wang, Qiang Meng, and Zhiyuan Liu. Bunker consumption optimization methods in shipping: A critical review and extensions. *Transportation Research Part E: Logistics and Transportation Review*, 53:49–62, 2013.

[43] World Shipping Council. The Liner Shipping Industry and Carbon Emission Policies. Technical report, World Shipping Council, 2009.

2 Maritime landside logistics: the quay crane assignment problem

Abstract This chapter studies the Quay Crane Scheduling Problem with non-crossing constraints, which is an operational problem that arises in container terminals. An enhancement to a mixed integer programming model for the problem is proposed and a new class of valid inequalities is introduced. Computational results show the effectiveness of these enhancements in solving the problem to optimality.

2.1 Introduction

A container terminal manager is faced with several interesting and challenging optimization problems and the topic of applying operational research methods to optimize container terminal operations has received a great amount of attention in recent years. The most important container terminal optimization problems as well as related solution methods are surveyed by Steenken et al. [7] and Stahlbock and Voß [6].

The focus of this article is on the quay crane scheduling problem (QCSP). In the QCSP a container vessel and a number of quay cranes are given and the objective is to make a schedule for the quay cranes such that the tasks that need to be performed on the vessel are carried out in a way that satisfies both the terminal manager and the vessel owner. Typically it is of primary importance to serve the vessel as quickly as possible. This is in the interest of the terminal manager, as it ensures that valuable quay space is freed up quickly and that labor cost is kept in check. It is also in the interest of the vessel owner, because it means that the ship can quickly commence its voyage, so to minimize unproductive time.

A conceptual container vessel is displayed in Figure 2.1. The figure shows that storage space on the vessel is divided into bays, rows and tiers, with a certain bay–row–tier combination pointing out a cell in the vessel that can store one forty feet container. This figure is, of course, a simplification. In practice the containers are not stored in a box-shaped vessel, the system for numbering positions on the vessel is different from what is used here and containers come in different sizes. The reader is referred to, for example, Pacino et al. [4] for a more realistic

This chapter is based on the contents of: Alberto Santini, Henrik Alsing Friberg, and Stefan Ropke. A note on a model for quay crane scheduling with non-crossing constraints. *Engineering Optimization*, 47(6):860–865, 2015. doi: 10.1080/0305215X.2014.958731.

description of a container vessel. For the purposes of this work, the simple description is sufficient since, as it is common in the QCSP literature, the assumption is made that each task consists of unloading and loading an entire bay.

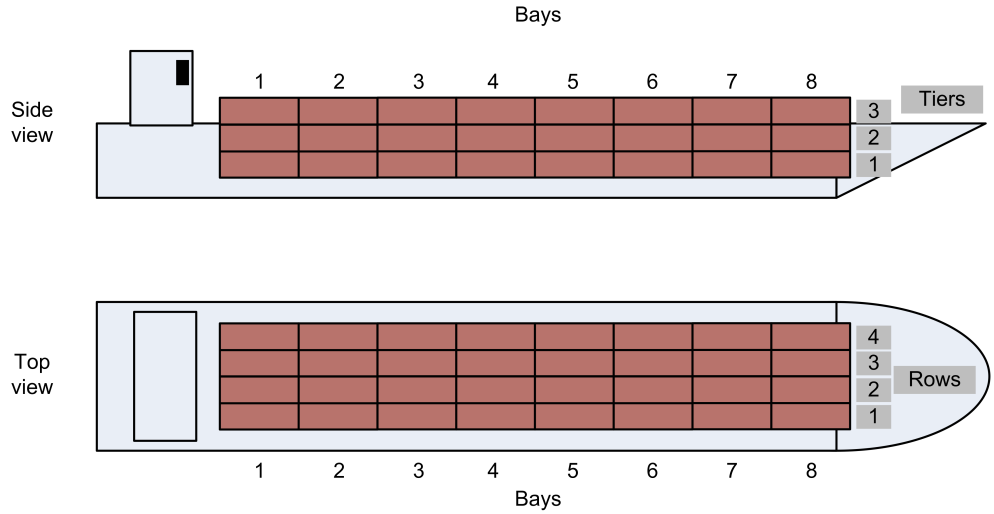


Figure 2.1: Conceptual container vessel

The QCSP model studied in this article is the one presented by Lee and Chen [3] and the contribution of the article is to show how the model, in a very simple way, can be improved to make it much more tractable for off-the-shelf solvers like CPLEX. Let $B = \{1, ..., n\}$ be the set of bays, $K = \{1, ..., m\}$ the set of quay cranes and p_b the processing time of bay $b \in B$. Each crane can process one bay at a time. Once the processing has started it has to run to its end. Cranes are running on rails, so they cannot overtake each other. The dimensions of bays and cranes are such that it is impossible to place two or more cranes at any bay simultaneously. it must be decided which crane should process which bay and at what time, while respecting the non-crossing constraint and the necessary time for processing each bay. It is assumed that the time for moving the crane between bays is negligible compared to the time for processing each bay. The objective is to minimize the make-span of the entire operation; that is, to minimise the ending time for the crane that ends the latest.

A classification scheme for QCSP formulations as well as a survey of contributions to the problem are presented by Bierwirth and Meisel [1]. QCSP formulations are classified according to four attributes: 1) task attribute, 2) crane attribute, 3) interference attribute and 4) performance attribute. The QCSP studied in this article is classified as “Bay | – | cross | max(compl)” which means that 1) each individual task is a bay — as opposed to a group of bays or a single container at the two extremes, 2) there are no special attributes associated with cranes, 3) the non-crossing of cranes is respected and 4) the maximum completion time of all tasks is minimized.

2.2 Mathematical model

The mathematical model is based on that of Lee and Chen [3] which in turn is an improved version of the model presented by Lee et al. [2]. The model uses the binary variable x_{bk} which is 1 if and only if bay $b \in B$ is served by crane $k \in K$, the binary variable $y_{bb'}$ is 1 if and only if work on bay $b \in B$ is finished before work on bay $b' \in B$ starts. The variables c_b indicate the completion time of bay $b \in B$ and c is the overall makespan. Using these variables and letting M be a sufficiently large positive integer number, the model is:

$$\begin{aligned}
\min \quad & c && && (2.1)\\
\text{subject to} \quad & c \geq c_b && \forall b \in B && (2.2)\\
& c_b \geq p_b && \forall b \in B && (2.3)\\
& \sum_{k \in K} x_{bk} = 1 && \forall b \in B && (2.4)\\
& c_b \leq c_{b'} - p_{b'} + M(1 - y_{bb'}) && \forall b, b' \in B, b \neq b' && (2.5)\\
& \sum_{k \in K} k x_{bk} - \sum_{k \in K} k x_{b'k} + 1 \leq M(y_{bb'} + y_{b'b}) && \forall b, b' \in B, b < b' && (2.6)\\
& \sum_{k \in K} k x_{b'k} - \sum_{k \in K} k x_{bk} \leq b' - b + M(y_{bb'} + y_{b'b}) && \forall b, b' \in B, b < b' && (2.7)\\
& x_{bk} = 0 && \forall b \in B, k \in K, k > b && (2.8)\\
& x_{bk} = 0 && \forall b \in B, k \in K, n - b < m - k && (2.9)\\
& x_{bk} \in \{0, 1\} && \forall b \in B, k \in K && (2.10)\\
& y_{bb'} \in \{0, 1\} && \forall b, b' \in B, b \neq b' && (2.11)\\
& c_b \in \mathbb{R} && \forall b \in B && (2.12)\\
& c \in \mathbb{R} && && (2.13)
\end{aligned}$$

The objective function (2.1) minimizes the total make-span of the process. Constraint (2.2) together with the minimization of the objective function ensures that c is equal to the largest of all completion times. Constraint (2.3) makes sure that the completion time of each bay is greater than its processing time. Constraint (2.4) ensures that every bay is served by exactly one crane. Constraint (2.5) links the $y_{bb'}$ and c_b variables. It forces $y_{bb'}$ to zero whenever $c_b > c_{b'} - p_{b'}$, that is, when b' is started before b finishes. Constraint (2.6) makes sure that the cranes do not cross and that each crane is working at one bay at a time. Constraint (2.7) ensures that there is always is enough space between two cranes (e.g. that crane 1 and 3 never are servicing two adjacent bays simultaneously). Constraints (2.8) and (2.9) ensure that no crane is pushed outside the bounds of the ship. This is illustrated in Figure 2.2 that shows an example with 8 bays and 3 quay cranes. In this example it is only crane 1 that is feasible for bay one; crane 2 and 3 are not feasible since that would imply that crane 1 is pushed further left and there may not be space for that since another vessel may be moored directly to the left of the current vessel or the vessel may be at the end of the quay. Similarly it is only crane 2 and 3 that can serve bay 7 since serving it by crane 1 would imply that crane 3 is pushed out of bounds. In the example, constraint (2.8) fixes x_{12}, x_{13} and x_{23} to zero and thereby ensures that no crane is pushed too far left. Constraint (2.9) fixes x_{71}, x_{81} and x_{82} to zero implying that no crane is pushed too far right.

Bay number	1	2	3	4	5	6	7	8
Feasible quay cranes	1	1 2	1 2 3	1 2 3	1 2 3	1 2 3	2 3	3

Figure 2.2: Bays and feasible quay cranes

The model is different from that of Lee and Chen [3] in two ways. Lee and Chen [3] creates two dummy bays and two dummy cranes in order to avoid cranes being pushed out of bounds. The dummy bays are situated at each end of the ship and the dummy cranes are locked to serving the two dummy bays during the entire planning period. As explained earlier, in this model the same issue is handled by the variable fixing done in (2.8) and (2.9). This modeling approach is preferred, as it requires fewer decision variables and constraints, while making the model easier to understand as well.

The second difference is that constraint (17) of Lee and Chen [3] has been left out. Using the notation introduced earlier, the constraint is

$$c_b + M y_{bb'} \geq c_{b'} - p_{b'} \quad \forall b, b' \in B,\ b \neq b'$$

It forces $y_{bb'}$ to 1 when $c_b < c_{b'} - p_{b'}$, that is, when b' starts after b finishes. Forcing the $y_{bb'}$ variable to one has no impact on the solution of the model since the only other place where $y_{bb'}$ occurs is in constraints 2.6 and 2.7 and here a value of one implies that the constraint will never be binding. The only drawback is that the $y_{bb'}$ sometimes can have a value 0 in the final solution when the value logically should be 1, but that is not an issue as the only interest is in the values of the x_{bk}, c_b and c variables.

The following simple family of valid inequalities has been introduced and its significant impact on computing experiments will be later shown:

$$c \geq \sum_{b \in B} x_{bk} p_b \quad \forall k \in K \tag{2.14}$$

Inequality (2.14) simply forces the overall make-span to be greater than the sum of all the processing times of the bays served by the same crane.

2.3 Computational results

The purpose of the computational results is to show the impact of inequality (2.14) when solving model (2.1) – (2.13). The computational tests were performed using a 2.93 GHz Intel Core i7 model 940 that has 4 cores. The MIP model was solved using CPLEX 12.4 which was allowed to use all cores of the computer and was allotted one hour per run. Table 2.1 shows results on the 24 instances used by Lee and Chen [3] and compare results with and

without constraint (2.14), as well as the results reported in [3]. The authors obtained the original data set from Lee and Chen and conducted the experiments using these instances.

The first column in the table reports the instance name, the first number gives the number of bays while the second gives the number of quay cranes. The next 6 columns report results from the mathematical model, including constraint (2.14). The first three of these columns report the lower and upper bounds when CPLEX terminated and the corresponding gap is calculated as (UB-LB)/LB · 100%. The next columns report the time spent by CPLEX, where a dash indicates that the solver timed out. The last two of the six columns report if the problem was solved to optimality and the number of branch and bound nodes explored. The following six columns show the same information for the model without constraint (2.14). The second to last column reports the best solution found by Lee and Chen [3]. Values marked with superscript "A" were found using CPLEX, while values marked with superscript "B" were found using a heuristic. The last column reports if the instance was solved to optimality in [3].

A first observation is that the valid inequality has a tremendous impact on the model. Consider for example the first instance. Without the inequality, CPLEX needs about 90 times as much time and needs to explore around 290 times as many nodes in the branch and bound tree in order to solve it to optimality. CPLEX is able to solve 15 instances to optimality when using the inequality and only 4 instances without the inequality. For the instances that none of the models can solve to optimality, the gap is much lower for the model using the inequalities.

When comparing to the results reported by Lee and Chen [3], it can be noticed that even the model without the valid inequality is able to solve more instances to optimality. This has been attributed to the fact that the experiment reported in the present work are using a faster computer and a more recent version of CPLEX. Lee and Chen [3] used a 3 GHz Pentium IV computer and did not report which version of CPLEX they used. The authors do not believe that the fact that they are using slightly fewer variables and constraints in their model has a great impact on CPLEX's ability to solve the problem.

The optimal results obtained with the proposed valid inequalities are often substantially better than the heuristic solutions reported in [3] and for most of the instances that were not solved to optimality, CPLEX is still able to find a better solution than Chen and Lee's heuristic. On the other hand, their heuristic is much faster and never uses more than 15 seconds.

The heuristic is also able to find a better solution than CPLEX for the largest instances with 100 bays. However, no container ship has 100 bays so such an instance is not realistic: one of the largest container ships currently in operation, Emma Maersk, has approximately 23 bays (based on inspection of photos). It is therefore possible to conclude that the enhanced model, within one hour, is able to solve most of the realistic sized instances to optimality.

2.4 Conclusions

In this article the quay crane scheduling model proposed by Lee and Chen [3] has been revisited. A simple family of inequalities has been introduced and this has been shown to have a great impact on the ability to solve the model to optimality. Computational results

	With constraints (2.14)					Without constraints (2.14)					Lee and Chen [3]	
Instance	LB	UB	Gap %	Time (s)	Opt	BB Nodes	LB	UB	Gap %	Time (s)	Opt	BB Nodes	Best Solution	Opt
16-4	726.0	726	0.0	3.4	✓	12878	726.0	726	0.0	305.4	✓	3726726	726[A]	✓
16-5	586.0	586	0.0	1.1	✓	5288	586.0	586	0.0	8.8	✓	52274	610[A]	
17-4	741.0	741	0.0	34.6	✓	74285	698.0	741	6.2	—		23593606	746[A]	
17-5	600.0	600	0.0	44.9	✓	76721	600.0	600	0.0	330.5	✓	3435505	604[A]	
18-4	720.0	720	0.0	41.2	✓	84801	687.0	720	4.8	—		29252698	737[A]	
18-5	579.0	579	0.0	35.9	✓	26854	579.0	579	0.0	233.7	✓	2587339	595[A]	
19-4	702.0	702	0.0	255.0	✓	317052	578.0	702	21.5	—		14824165	711[B]	
19-5	567.0	567	0.0	414.2	✓	404794	542.0	567	4.6	—		21702526	580[A]	
20-4	925.0	925	0.0	579.4	✓	1043278	679.0	925	36.2	—		9578048	949[A]	
20-5	739.0	739	0.0	271.4	✓	323089	677.0	749	10.6	—		14664665	781[B]	
21-4	759.0	759	0.0	1151.1	✓	1516153	540.1	759	40.5	—		6954439	801[B]	
21-5	612.0	612	0.0	3320.3	✓	3703084	524.0	612	16.8	—		8713794	622[B]	
22-4	757.0	757	0.0	1203.2	✓	2042917	612.0	759	24.0	—		6820790	766[B]	
22-5	611.0	611	0.0	2715.0	✓	6097888	544.0	611	12.3	—		12185027	636[A]	
23-4	886.0	886	0.0	1916.2	✓	2168284	562.0	889	58.2	—		5371125	910[B]	
23-5	708.8	719	1.4	—		3148789	546.9	713	30.4	—		5754008	740[B]	
24-4	857.5	860	0.3	—		3722243	600.0	861	43.5	—		4996441	874[B]	
24-5	686.0	698	1.7	—		5210886	525.7	693	31.2	—		3865710	712[B]	
25-4	1083.5	1087	0.3	—		2085828	579.0	1089	88.1	—		1020039	1129[B]	
25-5	866.8	871	0.5	—		2651609	560.1	877	56.6	—		2023905	921[B]	
50-8	998.3	1025	2.7	—		357369	427.0	1013	137.2	—		62120	1046[B]	
50-10	798.6	830	3.9	—		204419	448.0	839	87.3	—		2396552	897[B]	
100-8	2064.9	2132	3.2	—		319813	365.0	—	—	—		673507	2124[B]	
100-10	1651.9	1757	6.4	—		275514	345.4	—	—	—		561873	1747[B]	

Table 2.1: Computational results

showed that the improved model is able to solve most instances with realistic size to optimality. The authors believe that the model can provide inspiration for further work in this and related areas and that the computational results provided can be used as a basis for comparison for future heuristics for the problem.

Bibliography

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

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

[3] Der-Horng Lee and Jiang Hang Chen. An improved approach for quay crane scheduling with non-crossing constraints. *Engineering Optimization*, 42(1):1–15, 2010. ISSN 0305-215X.

[4] D. Pacino, A. Delgado, R.M. Jensen, and T. Bebbington. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. *Lecture Notes in Computer Science*, 6971:286–301, 2011.

[5] Alberto Santini, Henrik Alsing Friberg, and Stefan Ropke. A note on a model for quay crane scheduling with non-crossing constraints. *Engineering Optimization*, 47(6):860–865, 2015. doi: 10.1080/0305215X.2014.958731.

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

[7] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research – a classification and literature review. *OR Spectrum*, 26:3–49, 2004.

3 Maritime landside logistics: is the berth allocation problem solvable by partition colouring?

Abstract This chapter presents a study of the Partition Coloring Problem (PCP), a generalization of the Vertex Coloring Problem where the vertex set is partitioned, and analyses a claim by Demange et al. [7] that the PCP can be used to solve the Berth Allocation Problem (BAP). The PCP asks to select one vertex for each subset of the partition in such a way that the chromatic number of the induced graph is minimum. We propose a new Integer Linear Programming formulation with an exponential number of variables. To solve this formulation to optimality, we design an effective Branch-and-Price algorithm. We propose and compare several meta-heuristic algorithms capable of finding excellent quality solutions in short computing time. Extensive computational experiments on a benchmark test of instances from the literature show that our Branch-and-Price algorithm, combined with the new meta-heuristic algorithms, is able to outperform the state-of-the-art exact approaches for the PCP. After having established that the proposed method is a suitable tool to solve the PCP, we generated BAP instances, transformed them into PCP instances, and assessed the feasibility of solving the BAP as a PCP.

3.1 Introduction

Graph coloring problems are among the most studied ones in both graph theory and combinatorial optimization. Given an undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, the classical *Vertex Coloring Problem* (VCP) consists of assigning a color to each vertex of the graph in such a way that two adjacent vertices do not share the same color and the total number of colors is minimized. The *chromatic number* of G, denoted by $\chi(G)$, is the minimum number of colors in a coloring of G.

This chapter is based on the contents of: Fabio Furini, Enrico Malaguti, and Alberto Santini. Exact and euristic algorithms for the Partition Colouring Problem. *Submitted to Computers & Operations Resarch*, pages 1–17, 2017.

The VCP is an $\mathcal{NP}$-hard problem and it has a variety of applications, among which: scheduling, register allocation, seating plan design, timetabling, frequency assignment, sport league design, and many others (we refer the interested readers to Pardalos et al. [29], Marx [26], Lewis [21]). The VCP and its variants are very challenging from a computational viewpoint; the best performing exact algorithms are usually based on exponential-size Set Covering formulations, and require Branch-and-Price techniques to be solved (see, e.g., Malaguti et al. [25], Gualandi and Malucelli [13], Held et al. [15], Furini and Malaguti [11]). For dense graphs, good results are obtained by advanced Integer Linear Programming (ILP) compact formulations, like the so-called *representatives formulation* (see Campêlo et al. [3], Cornaz et al. [5]), which are able to remove the symmetry affecting classical descriptive compact ILP models.

In this manuscript we study the *Partition Coloring Problem* (PCP) which is a generalisation of the VCP where the vertex set is partitioned and exactly one vertex of each subset of the partition has to be colored. The PCP asks to select one vertex for each subset of the partition in such a way that the chromatic number of the induced graph is minimum. The PCP is $\mathcal{NP}$-hard since it generalizes the VCP and it is also known in the literature as the *Selective Graph Coloring Problem*.

Formally, let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a k-partition of the vertex set V of G. A *stable set* is a subset $S \subseteq V$ of non-adjacent vertices, i.e., $\forall u, v \in S, uv \notin E$. A *partial coloring* $\tilde{C}$ of G is a partition of a subset of vertices $\tilde{V} \subset V$ into h non-empty stable sets or colors ($\tilde{C} = \{\tilde{V}_1, \ldots, \tilde{V}_h\}$), while the remaining vertices $V \setminus \tilde{V}$ are uncolored. Let $f(v)$ be a function which returns the color of a colored vertex v ($v \in \tilde{V}$). The PCP consists of finding a partial coloring $\tilde{C}$ such that:

(i) $|\tilde{V} \cap P_i| = 1$ for $i = 1, 2, \ldots, k$;

(ii) $f(v) \neq f(w)$ for all $v, w \in \tilde{V}$, $vw \in E$;

(iii) h is minimum.

The minimum number of colors used in any optimal PCP solution is denoted in the rest of this manuscript as *Partition Chromatic Number* $\chi_{\mathrm{P}}(G, \mathcal{P})$.

Let us introduce an example, called *Example 1*. In the left part of Figure 3.1, we depict a graph G of ten vertices and thirteen edges. The graph is partitioned in five subsets ($k = 5$), each subset is composed by two vertices; the dotted lines are used to identify the subsets of the partition. In the right part of Figure 3.1, we depict a feasible partial coloring $\tilde{C}$ using two colors (gray and black). For each subset of the partition exactly one vertex is colored. The colored vertices, i.e., the vertices $v \in \tilde{V}$, are colored with the corresponding color (gray or black) while the uncolored ones are white.

The PCP models many real-world applications (see Demange et al. [7]) including: routing and wavelength assignment, dichotomy-based constraint encoding, antenna positioning and frequency assignment, as well as a wide variety of scheduling problems (timetabling, quality test) and a variant of the classical Travelling Salesman Problem. Furhtermore, Demange et al. [7] propose to model the Berth Allocation Problem (BAP) as a PCP, but provide no computational evidence on whether this is a practicable solution method for the BAP. Part of the aim of the present work is to provide an answer to this question.

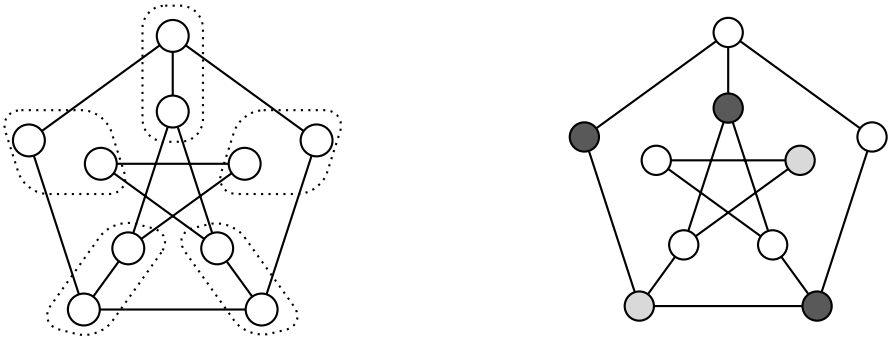


Figure 3.1: Example: (left) a graph G and a partition of its vertices in 5 subsets ($k = 5$); (right) a feasible partition coloring of G with two colors (gray and black).

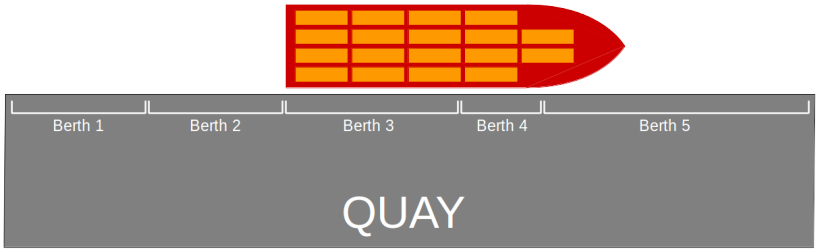


Figure 3.2: A ship docket at berth 3, occupying berths 3, 4, and 5.

3.1.1 Modelling the Berth Allocation Problem

In the considered version of the BAP (see, e.g., Türkoğulları et al. [33]) the terminal operator has a list of ships he will have to receive and dock during a certain time horizon. The quay is divided in berths where each ship can dock if the berth, or eventually the adjacent ones (depending on the size of the ship) are not occupied. Figure 3.2 shows a ship docking at berth 3. Because of the ship's size, no other vessel can use berths 3, 4, and 5 while the ship is docked.

More formally, let U be the set of ships. Each ship is identified by a length l_u and an amount of time t_u which is needed to load or unload the ship. Let B be the set of berths, aligned along a quay of total length L. Each berth has a length $\tilde{l}_b$, and starts at a distance d_b from the leftmost point of the quay. The first berth, therefore, will have $d_1 = 0$, the second will have $d_2 = \tilde{l}_1$, the third $d_3 = \tilde{l}_1 + \tilde{l}_2$, and so on. The time horizon T of duration $t_{\max}$ is divided into time intervals, and each ship can dock at a berth b at time interval t if $d_b + l_u \leq L$ and $t + t_l \leq t_{\max}$.

A feasible solution to the problem is an assignment of each ship to a berth and a time instant, such that no two ships occupy the same berth at the same time. We will now show how this problem can be modelled as a PCP on a graph $G_{\text{BAP}} = (V_{\text{BAP}}, E_{\text{BAP}})$. Consider the

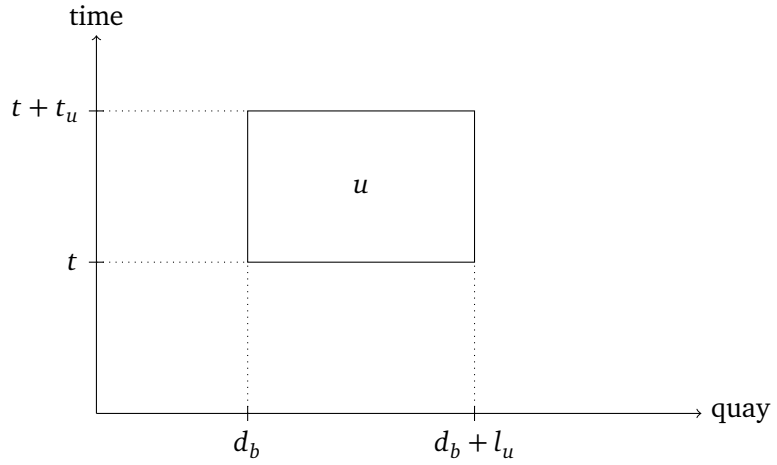


Figure 3.3: A representation of the planning horizon, showing a ship u docket at berth b from time t to time $t + t_u$.

vertex set:

$$V_{\text{BAP}} = \{(u, b, t) \in U \times B \times T \ : \ d_b + l_u \leq L, \ t + t_u \leq t_{\max}\} \tag{3.1}$$

Notice that if there are limitations on the arrival time of a ship, e.g. if we know that a ship u cannot arrive before time interval $\bar{t}$, the node set can be pruned accordingly, by removing all vertices of the type (u, b, t) for $t < \bar{t}$. The vertex set is partitioned into a partition $\mathscr{P}$ of $|U|$ clusters, one for each ship, as follows:

$$P_{\bar{u}} = \{(u, b, t) \in V_{\text{BAP}} \ : \ u = \bar{u}\} \tag{3.2}$$

The arc set E contains an edge between all pair of vertices (u, b, t) and (u', b', t') such that if ship u docks at berth b at time t, than it is not possible for ship u' to dock at berth b' at time t'. If we represent the planning horizon on a cartesian plane, where the x axis corresponds to the length of the quay, and the y axis corresponds to time, each node can be represented by a rectangle, as shown in Figure 3.3. Two nodes are then marked as incompatible (and an edge is drawn between them) if the two corresponding rectangles overlap.

We see, then, that the BAP has a feasible solution if and only if $\chi_{\text{P}}(G_{\text{BAP}}, \mathscr{P}) = 1$. In this case, the corresponding coloring gives a feasible docking plan for the time horizon. If $\chi_{\text{P}}(G_{\text{BAP}}, \mathscr{P}) > 1$, on the other hand, not all ships are serviceable within the time horizon.

3.1.2 Literature review: the PCP

The PCP has been introduced in Li and Simha [22] to model wavelength routing and assignment problems. Three heuristic algorithms for the VCP, i.e., the *Largest-First*, the *Smallest-Last* and the *Color-Degree* have been adapted to tackle the PCP. In Li and Simha [22], a first set of benchmark instances for the PCP has been proposed, representing mesh optical networks and the National Science Foundation Net (called `nsf` in the following). A memetic heuristic algorithm was proposed by Pop et al. [30], which combines genetic operators with a local search phase.

Theorethical results on the complexity of the PCP on particular classes of graphs have been obtained in Demange et al. [6] and Demange et al. [8].

To the best of our knowledge, only two works proposed exact algorithms for the PCP: Frota et al. [10] and Hoshino et al. [16]. The first one proposes a branch-and-cut algorithm based on the asymmetric representatives formulation introduced by Campêlo et al. [3, 2] for the VCP. A number of valid inequalities are proposed and used within a branch-and-cut framework. A Tabu Search heuristic algorithm has also been proposed to initialize the formulation. Computational tests are reported on randomly generated instances (called `random`), VCP instances from the literature, and instances derived from the routing and wavelength assignment literature (including the `nsf` instances, and a new set of instances called `ring`).

The second exact algorithm, i.e., the one presented in Hoshino et al. [16], is branch-and-price algorithm based on the Dantzig-Wolfe reformulation of the representatives formulation. In order to deal with an exponential number of variables, a column generation scheme has been proposed which is based on a set of pricing problems, one for each "representative" vertex. The authors show how to adapt to the valid inequalities used by Frota et al. [10] to the reformulated model. However, since the inequalities did not prove to be computationally effective, they were not added to the model. Several heuristic algorithms has also been proposed in Hoshino et al. [16]. Computational results on the `random`, `nsf`, and `ring` instances showed that the branch-and-price algorithm of Hoshino et al. [16] outperforms the branch-and-cut algorithm of Frota et al. [10].

3.1.3 Literature review: the BAP

Many variants of the Berth Allocation Problem exist in the literature. The first distinction is made between static and dynamic problem. In the static version (see, e.g., Imai et al. [17]), such as the one we consider in this chapter, the ship arrivals are known beforehand to the terminal operator. In the dinamic version (introduced by Imai et al. [18]), on the other hand, this information is only partially known at the initial planning time.

Another distinction is often made relative to the possible docking positions of the ship. Imai et al. [20] consider the case when the position can be chosen arbitrarily along the quayside,and is therefore represented by a real value. The majority of works, however, discretise the berth into segments and impose that each segment can be used by at most one ship at a time (see, e.g., Guan and Cheung [14]).

Finally, a further difference involves the objective function. The BAP can aim to the minimisation of the total makespan, i.e. the moment at which the last serviced ship is released, or the sum of the waiting times, i.e. the difference between a ship's arrival and docking times. Finally, if the service time is dependend from the berthing position (notice that our model can handle this case: the width of the rectangle in Figure 3.3 would then be dependent on the x-coordinate of its left side) a sum of waiting and handling time can be considered [4, 28]. Guan and Cheung [14], furthermore, consider a further generalisation of this objective function in which each vessel's term is multiplied by a different weight. Finally, Imai et al. [19] consider a version of the dynamic BAP in which certain vessels can be given priority over others. In our work, following the approach proposed by Demange et al. [7] we use a simpler

approach, as we only study the feasibility of the docking plan, without any consideration relative to wait or service times.

The BAP has been modelled using variation of other known combinatorial problems. The particular case of the discrete version of the problem where each ship only occupies one berth segment can be modelled as an assignment problem, which can be solved in polynomial time with the Hungarian method [18]. The more general discrete version can be modelled as an unrelated parallel machine scheduling problem, in both the static and dynamic variants [4]. The continuous version can be modelled as a cutting-stock problem [20]. This is, to the best of our knolwedge, the first time that the Berth Allocation Problem is solved via graph coloring.

3.1.4 Paper Contribution

In Section 3.2 we introduce a new formulation for the PCP with an exponential number of variables and in Section 3.3 we design a Brach-and-Price algorithm to solve it to proven optimality. Based on study of the mathematical structures of the formulation, we managed to design a pricing phase based on a unique pricing problem. This is a main improvement with respect to the state-of-the-art branch-and-price algorithm of Hoshino et al. [16], which requires instead to solve several pricing problems, one for each "representative" vertex. In order to obtain feasible integer solutions, two different branching strategies are also presented in Section 3.3. To effectively initialize our branch-and-price algorithm, new meta-heuristic algorithms are presented in Section 3.4. Several instances of the considered test bed have been solved to proven optimality at the root node, i.e., no branching is required, thanks to the quality of the heuristic solutions and the strength of the lower bound provided by the linear programming relaxation of the new formulation. In Section 3.5 we present extensive computational experiments comparing the new exact and heuristic algorithms with the state-of-the-art approaches. We also present results relative to the Berth Allocation Problem instances. Finally, in Section 3.7, we draw some conclusions and depict further possible lines of research on the topic.

3.2 Integer Linear Programming Formulations

In this section we first introduce a natural ILP formulation for the PCP and then we derive a new extended formulation based on the Dantzig-Wolfe reformulation of the natural formulation. A trivial upper bound on the number of colors used in any optimal PCP solution is given by the number k of subsets of the partition. We can then introduce a set of binary variables y with the following meaning:

$$y_c = \begin{cases} 1 & \text{if color } c \text{ is used} \\ 0 & \text{otherwise} \end{cases} \qquad c = 1, 2, \ldots, k;$$

and a set of binary variables x with the following meaning:

$$x_{vc} = \begin{cases} 1 & \text{if vertex } v \text{ is colored with color } c \\ 0 & \text{otherwise} \end{cases} \qquad v \in V,\ c = 1, 2, \ldots, k.$$

The first natural ILP formulation (called ILP$^{\mathrm{N}}$) reads:

$$(\mathrm{ILP}^{\mathrm{N}}) \qquad \min \quad \sum_{c=1}^{k} y_c \tag{3.3}$$

$$\sum_{c=1}^{k} \sum_{v \in P_i} x_{vc} = 1 \qquad i = 1, 2, \ldots, k \tag{3.4}$$

$$x_{vc} + x_{uc} \leq y_c \qquad uv \in E,\ c = 1, 2, \ldots, k \tag{3.5}$$

$$x_{vc} \in \{0, 1\} \qquad v \in V,\ c = 1, 2, \ldots, k \tag{3.6}$$

$$y_c \in \{0, 1\} \qquad c = 1, 2, \ldots, k, \tag{3.7}$$

where the objective function (3.3) minimizes the number of used colors, constraints (3.4) impose that one vertex per subset of the partition is colored, and constraints (3.5) impose that adjacent vertices do not receive the same color. Finally, constraints (3.6) and (3.7) define the variables of the formulation.

By replacing constraints (3.6) and (3.7) with

$$x_{vc} \geq 0 \qquad v \in V,\ c = 1, 2, \ldots, k \tag{3.8}$$

$$y_c \geq 0 \qquad c = 1, 2, \ldots, k, \tag{3.9}$$

we obtain the Linear Programming relaxation of ILP$^{\mathrm{N}}$, that will be denoted as LP$^{\mathrm{N}}$ in what follows.

Descriptive natural models for coloring problems are known to produce weak linear programming relaxations and are affected by symmetry (see Malaguti and Toth [23], Cornaz et al. [5]), hence, in general they can be solved to optimality only for small graphs. In order to improve the strength of the linear programming relaxation, and to remove the symmetry of model (3.3)–(3.7), we convexify constraints (3.5) through Dantzing-Wolfe decomposition (see [9]). Let us introduce the following exponential-size collection $\mathscr{S}$ of stable sets of G which intersect each subset of the partition at most once:

$$\mathscr{S} = \{S \subseteq V \ : \ uv \notin E,\ \forall u, v \in S\ ;\ |S \cap P_i| \leq 1,\ i = 1, \ldots, k\}. \tag{3.10}$$

A valid model for the PCP can be obtained by introducing, for each subset $S \in \mathscr{S}$, a binary variable ξ_S with the following meaning:

$$\xi_S = \begin{cases} 1 & \text{if vertices in } S \text{ take the same color} \\ 0 & \text{otherwise} \end{cases} \qquad S \in \mathscr{S}$$

then the extended ILP formulation reads as follows:

$$(\mathrm{ILP}^{\mathrm{E}}) \qquad \min \quad \sum_{S \in \mathscr{S}} \xi_S \tag{3.11}$$

$$\sum_{S \in \mathscr{S} : |S \cap P_i| = 1} \xi_S = 1 \qquad i = 1, \ldots, k \tag{3.12}$$

$$\xi_S \in \{0,1\} \qquad S \in \mathscr{S}, \tag{3.13}$$

where the objective function (3.11) minimizes the number of stable sets (colors), whereas constraints (3.12) ensure that exactly one vertex of each subset of the partition is colored. Finally constraints (3.13) impose all variables be binary. It is worth noticing that constraint (3.12) can be rewritten as follows:

$$\sum_{S \in \mathscr{S}:|S \cap P_i|=1} \xi_S \geq 1 \qquad i = 1, \ldots, k, \tag{3.14}$$

since it is always possible to transform a solution of model (3.11), (3.14) and (3.13) into a solution of model (3.11)–(3.13) of same value. Constraint (3.14) ensures that the associated dual variables take non negative values and this fact helps stabilizing our exact algorithm (see the next section for further details). The resulting formulation (3.11)-(3.14)-(3.13) is denoted as ILP^{E} in the following.

Finally, by relaxing the integrality of constraints (3.13) to

$$\xi_S \geq 0 \qquad S \in \mathscr{S}, \tag{3.15}$$

we obtain the Linear Programming relaxation of ILP^{E}, that is denoted as LP^{E} in what follows.

By observing that ILP^{E} is obtained by applying Dantzig-Wolfe decomposition of constraints (3.5) of ILP^{N} and since constraints (3.5) do not form a totally unimodular matrix, it follows that the quality of the lower bound obtained solving the LP relaxation of ILP^{N} is dominated by its counterpart associated with ILP^{E}:

Observation 3.2.1. *Model ILP^{E} dominates ILP^{N} in terms of Linear Programming relaxation.*

Proof. Proving the observation for the specific PCP models give more insight on the structure of the LP relaxation optimal solutions for the ILP^{N} and ILP^{E} models.

We first show that any feasible solution for LP^{E} can be converted to a solution that is feasible for LP^{N}. Given a function $p(v)$ which returns the corresponding index i ($i = 1, 2, \ldots, k$) of the subset of the partition of a vertex v ($v \in V$), we can uniquely define the color $c(S)$ of any $S \in \mathscr{S}$ as $\min_{v \in S} p(v)$. Let ξ^* denote a feasible solution to LP^{E} and assume, without loss of generality, that no subset of the partition is covered by more than one selected subset $S \in \mathscr{S}$. Let us define a solution (x^*, y^*) as follows: for each color c set

$$y^*_c = \sum_{S \in \mathscr{S} \,:\, c=c(S)} \xi^*_S \quad \text{and} \quad x^*_{vc} = \sum_{\substack{S \in \mathscr{S} \,:\, c=c(S), \\ v \in S}} \xi^*_S. \tag{3.16}$$

Thus, inequalities (3.14) ensure that constraints (3.4) are satisfied. Observe that, by construction, for each edge $uv \in E$ and for each color $c = 1, 2, \ldots, k$ we have $x^*_{vc} + x^*_{uc} < y^*_c$; thus, (x^*, y^*) is feasible to LP^{N}.

We then show a case where the optimal value of LP^{E} is strictly larger than the optimal value of LP^{N}. Consider the instance of Figure 3.4, where we depict a graph G of ten vertices and twenty edges. The graph is partitioned into five subsets ($k = 5$), and each subset is composed

by two vertices. As in Figure 3.1, the dotted lines define the subsets of the vertex partition. The figure report also a numbering of the vertices of the graph. The optimal solution of LP^E is $\xi^*_{S_1} = \xi^*_{S_2} = \xi^*_{S_3} = \xi^*_{S_4} = \xi^*_{S_5} = 0.5$ where the five stable sets are $S_1 = \{1, 8\}$, $S_2 = \{1, 9\}$, $S_3 = \{2, 9\}$, $S_4 = \{2, 10\}$, and $S_5 = \{8, 10\}$. Thus, the optimal solution value is 2.5, i.e., it is larger than the value of the LP relaxations of LP^N, which is 2. □

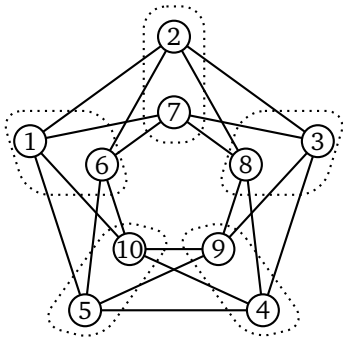


Figure 3.4: Example: a graph G of 10 vertices and a partition of its vertices in 5 subsets ($k = 5$).

Model ILP^E has exponentially many ξ_S variables ($S \in \mathscr{S}$), which cannot be explicitly enumerated for large-size instances. *Column Generation* (CG) techniques are then necessary to efficiently solve ILP^E. In the following we present a new Branch-and-Price framework for ILP^E, and refer the interested reader to [9] for further details on CG.

3.3 A New Branch-and-Price Algorithm

Two are the main ingredients of a Branch-and-Price algorithm, i.e., a CG algorithm to solve the Linear Programming Relaxation of the exponential-size integer model, and a branching scheme. We discuss separately these two aspects in the next sections.

3.3.1 Solving the Linear Programming Relaxation of ILP^E

Model (3.11), (3.14) and (3.15), initialized with a subset of variables containing a feasible solution, is called the *Restricted Master Problem* (RMP). Additional new variables, needed to solve LP^E to optimality, can be obtained by separating the following dual constraints:

$$\sum_{\substack{i=1,2,\ldots,k\,:\\ |P_i \cap S|=1}} \pi_i \leq 1 \qquad S \in \mathscr{S}, \tag{3.17}$$

where π_i ($i = 1, 2, \ldots, k$) is the dual variable associated with the i-th constraint (3.14). Accordingly, the CG performs a number of iterations, where violated dual constraints are added to the RMP in form of primal variables, and the RMP is re-optimized, until no violated dual constraint exist. At each iteration, the so-called *Pricing Problem* (PP) is solved. This

problem asks to determine (if any) a stable set $S^* \in \mathscr{S}$ for which the associated dual constraint (3.17) is violated, i.e., such that

$$\sum_{\substack{i=1,2,\ldots,k\ :\\ |P_i \cap S^*|=1}} \pi_i^* > 1, \tag{3.18}$$

where π^* is the optimal vector of dual variables for the current RMP.

At each iteration, the pricing problem can be modeled as a *Maximum Weight Stable Set Problem* (MWSSP) on an *auxiliary graph* $\hat{G} = (V, \hat{E})$, constructed as follows: the vertex set of $\hat{G}$ coincides with the vertex set of G while the edge set $\hat{E}$ is constructed from the edge set of G and its partition $\mathscr{P} = \{P_1, \ldots, P_k\}$:

$$\hat{E} = E \cup \{uv \ : \ u, v \in P_i,\ i = 1, \ldots, k\}. \tag{3.19}$$

In other words, each subset of the partition of G is transformed to a clique in $\hat{G}$. Given a weight vector $c \in \mathbb{R}_+^{|V|}$, where the weight c_v of the vertex $v \in P_i$ is set to the value π_i^* associated with the i-th subset of the partition, the pricing problem corresponds then to a MWSSP in $\hat{G}$, that is, to determine a stable set S of $\hat{G}$ maximizing $\sum_{v \in S} c_v$.

Notice that since each partition subset has been turned into a clique, such a stable set contains at most one vertex per subset P_i and therefore collects each profit π_i at most once. The MWSS can be solved by means of a specialized combinatorial Branch-and-Bound algorithm (see Section 3.5).

If a stable set S^* has total weight larger than one (that is, the reduced cost is negative), the associated column is added to the RMP and the problem is re-optimized. If, on the other hand, the total weight is not larger than 1, by linear programming optimality conditions no column can improve the objective function of the RMP and therefore we have solved LP^{E} to optimality.

3.3.2 Branching scheme for ILP^{E}

The design of a branching scheme is crucial for the performance of a branch-and-price algorithm [34]. In the following we describe the branching scheme adopted in our new Branch-and-Price framework. Two are its main properties. Firstly, it is a complete scheme, i.e., it ensures that integrality can be imposed in all cases. Secondly, it does not require modifications neither on the master problem nor the pricing algorithm. The latter means that our branching does not alter the structure of the pricing problem so that the same algorithm can be applied during the entire search.

Consider a fractional solution ξ^* to LP^{E}, at a given node of the branching tree, and let $\hat{\mathscr{S}} \subseteq \mathscr{S}$ be the set of columns in the RMP at the node. We propose a branching scheme composed of two rules applied in sequence, i.e., when the branching condition for the first rule fails, the second is applied.

The *first branching rule* is designed to impose that exactly one vertex is colored for each subset. Constraints (3.14) impose that the sum of the values of the variables associated with stables sets intersecting each subset is at least one, but in a fractional solution these stable sets can include different vertices in the same subset of the partition. A given subset P_i has

more than one (partially) colored vertex if:

$$|\{v \in P_i \ : \ \sum_{S \in \hat{\mathcal{S}}, v \in S} \xi_S^* > 0\}| > 1 \tag{3.20}$$

In case more than one such subset exists, we select the subset i with the largest number of (partially) colored vertices, breaking ties by size of the subsets (preferring smaller subsets and breaking further ties randomly). We then branch on the vertex $v \in P_i$ with the largest value of $\sum_{S \in \hat{\mathcal{S}}, v \in S} \xi_S^*$. Two children nodes are then created:

- in the first node we impose that v is the colored vertex for subset P_i;
- in the second node, we forbid that v is the colored vertex for subset P_i.

This branching rule can be enforced without any additional constraint neither for the RMP nor for the pricing problem. To force the coloring of v in the children nodes of the branching scheme, we remove from the graph G all other vertices $u \in P_i$ $(u \neq v)$; to forbid the coloring of v, we simply remove the vertex from the graph G. This first branching rule is not complete since it may happen that a vertex (partially) belongs to more that one stable set in the solution ξ^*.

If this happens for a vertex v, there must be another vertex u (belonging to a different subset of the partition) such that:

$$\sum_{S \in \hat{\mathcal{S}} : v,u \in S} \xi_S^* = \gamma, \ \gamma \text{is fractional.} \tag{3.21}$$

We say that v and u are a *fractionally colored* pair of vertices.

The *second branching rule* is designed to impose that each pair of (colored) vertices either takes the same color, or the two vertices of the pair take different colors. This rule has been proposed for the VCP by Zykov [35] and used to derive several effective Branch-and-Price algorithms for the VCP, starting from the seminal work by Mehrotra and Trick [27], see, e.g., [25, 13, 15]. In case more pairs of fractionally colored vertices exist, we select the pair v and u with the largest γ value. Two children nodes are then created:

- in the first node we force vertices v and u to take the same color;
- in the second node we force vertices v and u to take different colors.

The second branching rule can also be enforced without any additional constraint neither for the RMP nor for the pricing problem. To force different colors for a pair of vertices v and u in the children nodes of the branching scheme, we add the edge vu to E. On the other hand, to force v and u to take the same color, we remove v and u from the graph G and replace them with a new vertex z; we add edges zw for all $w \in V$ such that either $uw \in E$ or $vw \in E$. We then consider a stable set containing the vertex z coloring both $P_{p(v)}$ and $P_{p(u)}$, where the function $p(v)$ $(v \in V)$ returns the index of the subset of the partition containing vertex v.

In our Branch-and-Price algorithm we first define the vertices for each subset of the partition to be colored, i.e., we apply the first branching rule. Then, in case the solutions are still fractional, we apply the second branching rule in order to obtain integer solutions.

After branching, the variables that are incompatible with the branching decision are removed from the children nodes. The following observation states that the two proposed branching rules define a complete branching scheme for the formulation ILP$^{\mathrm{E}}$:

Observation 3.3.1. *The two branching rules applied in sequence provide a complete branching scheme for model ILP^{E}.*

Proof. After the application of the first branching rule, the colored vertex in each subset of the partition is determined. In [1] it is proved that for any 0-1 constraint matrix A (as for the case of LP$^{\mathrm{E}}$), if a basic solution ξ^* to $A\xi = 1$ is fractional, then there exist two rows i and j such that:

$$0 < \sum_{S \in \mathscr{S} \,:\, i,j \in S} \xi^*_S < 1 \tag{3.22}$$

This result allows us to conclude that if a solution is fractional then we can determine two subsets of the partition such that (3.22) holds. The same holds for the case in which $A\xi^* > 1$: in any optimal fractional solution to LP$^{\mathrm{E}}$, the rows for which covering constraints are satisfied with equality must be covered by at least two columns with associated fractional variables, and the previous result applies. By picking the colored vertex from the first and the colored vertex from the second subset, the two vertices constitute a fractionally colored pair of vertices on which to apply the second branching rule. □

3.4 Heuristic algorithms

We devised three algorithms based on meta-heuristics: a Tabu Search, inspired by that of Malaguti et al. [24]; a heuristic based on the Adaptive Large Neighbourhood Search (ALNS, first introduced by Ropke and Pisinger [31]) heuristic; a variation of the ALNS-based heuristic, improved by a Local Search phase. These algorithms are used to initialize the Branch-and-Bound algorithm with a feasible solution of good quality. Basic initial solutions are provided to the heuristics. They are created by a simple greedy procedure that constructs stable sets one at the time. Starting from a stable set composed by a vertex from an uncolored subset, the procedure keeps adding the least connected vertices of uncolored subsets to the current stable set. When this is not possible anymore, it starts a new stable set.

3.4.1 Tabu Search

The Tabu Search algorithm aims to find solutions to the PCP that use exactly k colors. Once such a solution has been found, the heuristic is restarted, trying to find a solution of $k-1$ colors, and so on, until a stopping criterion intervenes.

The algorithm considers $k+1$ buckets $B_1, \ldots, B_{k+1}$. Each of the first k buckets represents a feasible stable set of $\mathscr{S}$. The $(k+1)$-th bucket contains all other vertices. A feasible solution is reached when the stable sets $B_1, \ldots, B_k$ form a selective coloring for the graph G.

At each iteration, we randomly select an uncolored subset P_i and a random vertex $v \in P_i$ (from B_{k+1}). We try to insert v in each of the first k buckets and compute a score for each

insertion, given by the sum of the external degrees of all the vertices that would have to leave B_i, when v enters it. The external degree of a vertex w is the number of vertices u such that $w, u \in E$ and $p(w) \neq p(u)$. After evaluating each possible insertion, we perform the one which has the lowest score. Notice that, if there is a bucket i where we can place v without having to remove any other vertex, this insertion has score 0 and is always the preferred one. When inserting vertex v in bucket B_i, we add the couple (i, v) to a tabu list, meaning that, for the next I iterations (where I is a parameter) if v exits bucket B_i, it cannot re-enter it. In our experiments, we set $I = 150$; the algorithm was run for 50000 iterations.

3.4.2 ALNS-based heuristic

The basic idea behind ALNS is to explore the solution space using a large collection of neighbourhoods. At each iteration, the neighbourhood to explore is chosen randomly, with a probability proportional to a given score. The score, in turn, reflects the past performance of the neighbourhood during the solution process. Algorithm 1 shows the general framework of ALNS.

In Line 1 and Line 2 the current and best solutions are initialized; Line 3 initializes the iteration counter, and Line 4 initializes the neighbourhood scores. The algorithm is run for $maxiter$ iterations. At each iteration, a neighbourhood is selected (Line 6) using a roulette-wheel selection mechanism, with probabilities proportional to the scores. Since the neighbourhood size is often exponential, N is often not explored completely, but just sampled, in order to produce a new solution x' (Line 7). Next, in Line 8, the new solution is evaluated and either accepted or rejected, according to an *acceptance criterion*. The acceptance criterion uses a set of parameters that can change during the solution process: for example, accepting worsening solutions might be more likely at the beginning of the process than at the end. The current (Line 9) and best (Line 12) solutions are possibly updated, and finally the scores (Line 14), the acceptance criterion parameters (Line 15) and the iteration counter (Line 16) are updated, and the best known solution is returned on Line 18.

In our algorithm, the set $\mathcal{N}$ of neighbourhoods is not explicitly enumerated. Rather, we give a set of destroy methods and a set of repair methods. The former transform a feasible solution into an unfeasible one, and the latter transform an unfeasible solution into a feasible one. Each combination of a destroy method, followed by a repair method gives rise to a neighbourhood. Rather than keeping scores for the neighbourhoods, then, we keep the scores of the individual destroy and repair methods and perform two independent roulette-wheel selections. Notice that this approach can only work if (as in our case) all destroy and repair methods are compatible, meaning that it is possible to repair a destroyed solution produced by any destroy method, with any repair method.

In our implementation, we devised the destroy and repair methods described below. In order to compact the exposition, some similar methods have been grouped together and their distinctive elements are listed in curly braces.

- **Destroy methods**
  1. Select {a random, the smallest, the biggest} stable set of the solution, and remove a random vertex from that set.

Algorithm 1: ALNS Framework

Input : Initial solution: x_0
Input : List of neighbourhoods: $\mathcal{N}$
Input : Neighbourhood scores: λ_N for $N \in \mathcal{N}$
Input : Acceptance parameters: p
Input : Objective to minimize: $f(\cdot)$

$x = x_0$
$x^* = x_0$
$i = 1$
$\lambda_N = 1, \quad \forall N \in \mathcal{N}$

while $i \leq maxiter$ **do**
  Choose neighbourhood $N \in \mathcal{N}$ with probability proportional to λ_N
  Select $x' \in N(x)$
  if *Accept new solution* x' *(using parameters* p*)* **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update scores λ
  Update acceptance parameters p
  $i = i + 1$
end
return x^*

2. Select the colored vertex with {smallest, largest} external degree, and remove it from the stable set it belongs to.
3. Select the colored vertex with {smallest, largest} color degree, and remove it from the stable set it belongs to. The color degree of a vertex v is the number of vertices w such that $\{v, w\} \in E$ and w is colored in the current solution.
4. As in Items 2 and 3 but the vertex to be removed is chosen with a roulette wheel method, in which the probability of being chosen is {directely, inversely} proportional to its degree.
5. Select {a random, the smallest} stable set of the solution, remove the stable set.
6. As in Item 5, but the criterion used to choose the set is that it has the smallest cumulative {external, color} degree, defined as the sum of the degrees of its vertices.
7. As in Item 6, but where the set is chosen with a roulette wheel method, in which the probability of being chosen is inversely proportional to the cumulative degree.

- **Repair methods**
  1. For each uncolored subset, select a random vertex from the subset and add it to {a random, the smallest, the largest} feasible stable set of the current solution. If

it is not possible to put the vertex in any existing stable set, define a new stable set.

2. As in Item 1, but for each uncolored subset, we select the vertex with smallest {external, color} degree.

The scores of the destroy and repair heuristics are updated at each iteration as follows: if a method produced a new best solution, its score is increased by 0.5; otherwise, if a method produced a solution accepted by the acceptance criterion, its score is increased by 0.1; otherwise, its score is reduced by 0.5%.

The classical acceptance criterion used within ALNS is Simulated Annealing, in which a solution is accepted with probability $\exp((f(x)-f(x'))/T)$, where T is a parameter (called *temperature*) that decreases exponentially during the solution process. However, the objective function $f(\cdot)$ we consider simply counts the number of used colors, and therefore it only assumes a very limited range of discrete values, while moving from one value to the next (i.e., reducing the number of colors by one) is a relatively rare occurence. For these reason, an acceptance criterion that accepts a solution based on its objective value does not seem particularly suited for the PCP. We, therefore, decided to use the "*Worse Accept*" criterion, proposed by Santini et al. [32], which accepts a new solution x' if either it uses strictly fewer colors than the current one, or otherwise with a certain probability p, which starts at a high value, and decreases linearly to reach 0 at the end of the solution process. Notice that p does not depend on the value $f(x')$ of the new solution. In our implementation, we used the start value $p = 0.05$, and the algorithm was run for 20000 iterations.

3.4.3 Local Search refinement

The local search is a heuristic procedure that can be applied each time a new solution is generated by the destroy and repair heuristics, before the solution is evaluated. Although applying the refinement to all generated solutions certainly increases the running time of the algorithm, it also produces solutions of higher quality, and gives an important improvement on the overall quality of the algorithm, as outlined by the computational experiments reported in Section 3.5.2.

The local search operator tries to reduce the number of colors used in a solution by one unit, by emptying the smallest cardinality stable set in the solution. Assume the current solution uses k colors $S_1, \ldots, S_k$ and, without loss of generality, that S_k is the smallest cardinality stable set. The local search heuristic first uncolors all vertices of S_k. It then considers each uncolored partition, and tries to color any vertex (say v) of the partition by inserting it in one of $S_1, \ldots, S_{k-1}$.

If there is a stable set S_i such that $S_i \cup \{v\}$ is still a stable set, v is placed in S_i. Otherwise, the procedure tries to insert v in one stable set S_i by removing all vertices $w_1, \ldots, w_r$ in S_i that are not compatible with v, i.e., $v, w_j \in E$ for $j = 1, \ldots, r$. If it is possible to greedily relocate all vertices w_j in other stable sets, the vertices are relocated, and v is inserted in S_i. If there is no stable set where vertex v can be inserted, the procedure tries to color another vertex from the same uncolored partition. The uncolored partitions, the vertices v from the uncolored partition and stable sets S_i and are considered in random order.

Difference with χ_P	Tabu	ALNS	ALNS + LS
0	45	72	160
1	66	66	10
2	23	17	1
3	19	8	0
4	12	6	0
≥ 5	6	2	0

Table 3.1: Quality of the solution produced by Tabu search, ALNS, and ALNS enhanced with local search.

If, for some uncolored partition, no vertex can be inserted in a stable set $S_1, \ldots, S_{n-1}$, local search is stopped. On the other hand, if the local search manages to recolor one vertex for each uncolored partition, it has reduced the number of colors in the solution by one.

3.5 Computational Results: PCP

The experiments have been performed on a computer with a 3.10 GHz 4-core Intel Xeon processor and 8Gb RAM, running a 64 bits Linux operating system. The algorithms were coded in `C++` and all the codes were compiled with `gcc 6.2` and `-O3` optimizations. At each iteration of the Column Generation procedure (see Section 3.3), we used `Cplex 12.6.1` as a Linear Programming solver (ran single-threaded). The pricing MWSS subproblems were solved using the open-source implementation of the algorithm described in Held et al. [15] and available at `https://github.com/heldstephan/exactcolors`.

3.5.1 Instances

In order to compare our results with the ones present in the literature, we tested our approach on the instance classes `random`, `nsfnet` and `ring` presented in Section 3.1.2. In their work, Hoshino et al. [16] consider a subset of 187 out of a total of 199 instances, removing those instances solved to optimality in less than a second by either their algorithm or that of Frota et al. [10].

We, in turn, removed 12 instances of the `ring` class, as we realised that they were identical copies of the same three basic instances. In particular, instances `n10_p1.0_1` to `n10_p1.0_5` all correspond to the same instance (therefore only 1 out of the 5 instances has been kept), as do the analogous instances of base type `n15` and `n20`. This reduced the total number of instances to 175. In particular, we used 56 `random`, 32 `nsfnet`, and 87 `ring` instances. We also note that in `ring` instances, all elements of the partition have cardinality 2.

3.5.2 Initial Heuristics

This section compares the three initial heuristics presented in Section 3.4. Out of the 175 instances we considered, we know the optimal result (either from our branch-and-price algorithm, or from that of Hoshino et al. [16]) of 170 of them. For these instances, we

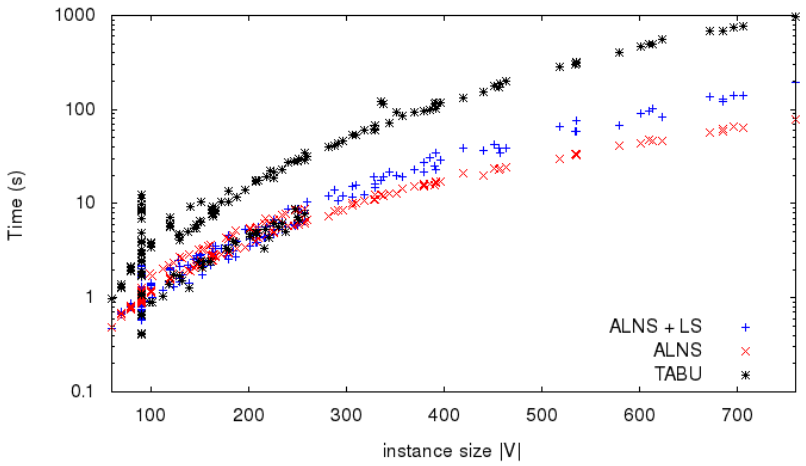


Figure 3.5: Running time of the three heuristics, versus instance size (measured in number of vertices of the graph).

can compute how well the initial heuristics fare, with respect to the optimal solution value. Table 3.1 shows the number of instances for which each of the three heuristics has found: the optimal solution; a solution with one, two, three or four colors more than the optimum; a solution with at least five colors more than the optimum. The table shows that the ALNS-based heuristics outperform Tabu search and that the introduction of a local search phase greatly enhances the effectiveness of ALNS. Figure 3.5 shows computational times for the three heuristics. Notice that the time axis is logarithmic, showing an exponential dependency of the running time to the number of vertices of the graph. For smaller graphs, furthermore, we also recorded a dependency of the Tabu Search algorithm to the graph density; this explains, for example, the two clusters of points visible in the area relative to graphs with fewer than 300 vertices. With respect to ALNS, the introduction of local search slightly increases the computational time of ALNS, but the improved solution quality definitely justifies the increase.

3.5.3 Branch-and-price Algorithm

In this section, we report the results obtained by our new branch-and-price algorithm with a time limit of 1 hour. The time limit includes the runtime of the ALNS+LS heuristic, which is used to generate an initial feasible solution.

Since linear relaxation bounds are generally tight, the overall performance of our algorithms very much depends on the ability to find good feasible solutions (upper bounds) early in the branching tree. Therefore, we solve the restricted master program as an integer model with `Cplex` in a subset of the branching nodes aiming at improving the incumbent solutions. Solving these MIP models can be computationally expensive, for this reason we limit the attempts of improving the upper bounds. We solve the MIP associated with the current pool of columns at the root and at all nodes until the pool contains no more than 400 columns. Then, we solve the MIP every 1000^{th} node explored. In order to avoid unnecessary computational

Instance	$\|V\|$	$\|E\|$	χ_P	Cols	Time (s)	Instance	$\|V\|$	$\|E\|$	χ_P	Cols	Time (s)
n100p5t2s1	100	2515	7	180	0.7	ring_n10p0.9s4	164	5337	12	159	1.1
n100p5t2s2	100	2460	7	192	0.8	ring_n10p0.9s5	166	5497	13	122	0.6
n100p5t2s4	100	2468	7	172	0.8	ring_n10p1.0s1	180	6450	13	162	2.2
n100p5t2s5	100	2524	7	188	0.5	ring_n15p0.3s3	130	3430	13	77	0.1
n60p5t2s1	60	924	5	96	0.2	ring_n15p0.3s4	140	3970	12	108	0.4
n80p5t2s1	80	1616	6	137	0.4	ring_n15p0.3s5	150	4525	12	116	0.3
n80p5t2s2	80	1570	6	144	0.6	ring_n15p0.4s1	162	5343	15	85	0.2
n80p5t2s5	80	1634	6	158	0.3	ring_n15p0.4s2	188	7099	14	104	0.1
n90p1t2s4	90	435	2	323	7.1	ring_n15p0.4s3	164	5484	15	106	0.2
n90p2t2s2	90	837	3	254	2.6	ring_n15p0.4s4	180	6549	14	115	0.3
n90p5t2s4	90	2040	7	154	0.4	ring_n15p0.4s5	196	7761	16	160	0.8
n90p5t2s5	90	2082	7	154	0.3	ring_n15p0.5s1	208	8760	18	119	0.2
n90p6t2s1	90	2462	8	145	0.3	ring_n15p0.5s2	226	10280	18	157	0.4
n90p6t2s2	90	2403	8	150	0.3	ring_n15p0.5s3	208	8828	19	131	0.1
n90p6t2s3	90	2463	8	145	0.3	ring_n15p0.5s4	210	8974	17	184	1.7
n90p8t2s1	90	3268	12	131	0.2	ring_n15p0.5s5	226	10281	18	136	0.2
n90p8t2s3	90	3282	12	133	0.5	ring_n15p0.6s1	250	12567	19	180	0.9
n90p8t2s5	90	3239	12	133	0.2	ring_n15p0.6s2	258	13395	19	160	0.6
n90p9t2s1	90	3637	16	139	0.2	ring_n15p0.6s3	258	13529	22	163	0.1
n90p9t2s2	90	3619	16	129	0.2	ring_n15p0.6s4	260	13724	20	210	4.6
n90p9t2s3	90	3621	16	142	0.4	ring_n15p0.6s5	252	12808	21	176	0.7
n90p9t2s5	90	3640	16	126	0.2	ring_n15p0.7s1	288	16745	21	202	2.4
nsf_p0.4_s3	101	625	6	54	0.1	ring_n15p0.7s2	330	21921	25	191	1.1
nsf_p0.4_s4	99	579	7	60	0.2	ring_n15p0.7s3	306	18974	23	175	0.4
nsf_p0.4_s5	112	769	6	127	1.0	ring_n15p0.7s4	310	19455	23	169	0.8
nsf_p0.5_s2	130	1041	8	79	0.2	ring_n15p0.7s5	282	16126	23	179	1.1
nsf_p0.5_s4	118	828	7	76	0.6	ring_n15p0.8s2	352	24949	25	198	1.2
nsf_p0.5_s5	132	1099	7	111	0.6	ring_n15p0.8s3	338	23111	26	208	1.2
nsf_p0.6_s1	149	1506	9	99	0.3	ring_n15p0.8s4	344	23984	25	204	1.6
nsf_p0.6_s2	154	1443	9	77	0.4	ring_n15p0.8s5	328	21736	25	233	4.6
nsf_p0.6_s3	153	1404	9	108	0.3	ring_n15p0.9s2	386	30115	27	318	33.0
nsf_p0.6_s4	161	1650	9	125	0.4	ring_n15p0.9s3	380	29257	28	243	5.6
nsf_p0.6_s5	139	1211	9	78	0.2	ring_n15p0.9s4	380	29233	27	231	2.6
nsf_p0.7_s1	180	2157	10	146	0.9	ring_n20p0.2s1	128	3345	10	80	0.1
nsf_p0.7_s2	202	2718	11	112	0.4	ring_n20p0.2s2	172	6040	15	133	0.3
nsf_p0.7_s3	177	1966	10	124	0.5	ring_n20p0.2s3	162	5293	13	100	0.2
nsf_p0.7_s4	187	2191	11	143	0.7	ring_n20p0.2s4	152	4727	13	106	0.1
nsf_p0.7_s5	159	1642	9	138	1.6	ring_n20p0.2s5	162	5449	16	137	0.2
nsf_p0.8_s1	201	2594	11	136	1.4	ring_n20p0.3s1	218	9618	18	143	0.4
nsf_p0.8_s2	221	3167	11	147	2.5	ring_n20p0.3s2	246	12431	24	149	0.1
nsf_p0.8_s3	208	2743	11	125	0.7	ring_n20p0.3s3	234	11068	18	145	0.6
nsf_p0.8_s4	209	2876	11	158	2.2	ring_n20p0.3s4	222	10169	20	148	0.1
nsf_p0.8_s5	184	2049	11	100	0.3	ring_n20p0.3s5	240	11922	21	161	0.4
nsf_p0.9_s1	216	2966	12	155	1.2	ring_n20p0.4s1	306	18925	24	175	0.3
nsf_p0.9_s2	231	3370	12	131	0.9	ring_n20p0.4s2	318	20852	28	209	0.8
nsf_p0.9_s3	217	2889	12	127	0.7	ring_n20p0.4s3	296	17718	21	205	2.9
nsf_p0.9_s4	226	3169	12	202	5.1	ring_n20p0.4s4	292	17451	25	192	0.8
nsf_p0.9_s5	234	3424	12	176	2.4	ring_n20p0.4s5	330	22422	29	203	0.8
nsf_p1.0_s1	250	3996	13	161	3.5	ring_n20p0.5s1	392	31120	32	205	0.1
nsf_p1.0_s2	251	4026	13	209	21.8	ring_n20p0.5s2	396	32349	32	234	0.9
nsf_p1.0_s3	238	3465	13	163	2.6	ring_n20p0.5s3	380	29251	27	220	1.2
nsf_p1.0_s4	257	4320	13	208	15.9	ring_n20p0.5s4	358	26197	28	221	1.5
nsf_p1.0_s5	248	3879	13	161	3.1	ring_n20p0.5s5	390	31285	32	228	0.2
ring_n10p0.7s3	130	3353	10	75	0.1	ring_n20p0.6s1	458	42545	36	271	1.1
ring_n10p0.8s1	146	4221	11	123	0.4	ring_n20p0.6s2	464	44339	36	321	7.5
ring_n10p0.8s2	152	4605	12	97	0.2	ring_n20p0.6s5	440	39665	34	239	0.3
ring_n10p0.8s3	144	4115	11	78	0.1	ring_n20p0.7s1	536	58410	39	286	3.8
ring_n10p0.8s4	146	4233	11	81	0.2	ring_n20p0.7s2	580	68948	43	336	9.2
ring_n10p0.8s5	138	3785	11	104	0.3	ring_n20p0.8s2	624	79813	46	347	7.2
ring_n10p0.9s1	162	5225	12	97	0.2	ring_n20p0.8s3	614	76597	43	380	59.0
ring_n10p0.9s2	164	5360	12	87	0.2	ring_n20p0.8s5	602	74191	43	392	36.7
ring_n10p0.9s3	166	5479	13	115	0.2	ring_n20p0.9s1	672	92075	48	393	54.9

Table 3.2: Computational results for instances solved at the root node.

			B&P [16]		B&P (new)						
Instance	$\|V\|$	$\|E\|$	LB	UB	hUB	rLB	UB	Nodes	Cols	rTime (s)	Time (s)
n100p5t2s3	100	2532	7	7	8	7	8	68655	16093	0.6	tl
n120p5t2s1	120	3616	8	8	9	8	8	61712	16045	1.6	1100.0
n120p5t2s2	120	3563	8	8	8	7	8	35	610	1.34	5.4
n120p5t2s3	120	3638	8	8	9	8	8	64801	15330	1.6	1366.0
n120p5t2s4	120	3565	8	8	9	7	8	59603	17100	1.74	1604.0
n120p5t2s5	120	3653	8	8	9	8	8	56675	17553	1.81	1660.1
n70p5t2s2	70	1204	6	6	6	5	6	21	285	0.32	0.6
n70p5t2s3	70	1218	6	6	6	5	6	261	1236	0.49	2.4
n70p5t2s4	70	1217	6	6	6	5	6	35	404	0.25	0.6
n80p5t2s3	80	1611	6	6	7	6	6	16441	8460	0.46	313.0
n80p5t2s4	80	1595	6	6	7	6	6	7579	5424	0.31	91.4
n90p1t2s1	90	445	2	3	3	2	3	509	17079	6.78	195.2
n90p1t2s2	90	442	2	3	3	2	3	1581	42500	7.04	837.4
n90p1t2s3	90	465	3	3	3	2	3	41	1807	3.37	26.2
n90p1t2s5	90	485	3	3	3	2	3	21	1237	4.88	23.7
n90p2t2s1	90	823	3	4	4	3	4	6598	61174	3.32	tl
n90p2t2s3	90	869	3	4	4	3	4	283	4485	3.09	63.4
n90p2t2s4	90	821	3	4	4	3	4	6943	59331	3.75	tl
n90p2t2s5	90	862	3	4	4	3	4	981	13679	3.51	201.5
n90p3t2s1	90	1215	4	5	5	4	5	9149	29746	1.18	1393.3
n90p3t2s2	90	1234	4	5	5	4	5	9661	29767	1.06	1466.1
n90p3t2s3	90	1275	5	5	5	4	5	187	1898	1.27	17.3
n90p3t2s4	90	1211	4	5	5	4	5	13961	42572	1.8	tl
n90p3t2s5	90	1268	5	5	5	4	5	877	6064	1.48	57.9
n90p4t2s1	90	1624	5	6	6	5	6	12535	18276	0.74	852.6
n90p4t2s2	90	1600	5	6	6	5	5	4088	9745	0.89	145.6
n90p4t2s3	90	1650	6	6	6	5	6	889	3663	0.83	25.6
n90p4t2s4	90	1638	6	6	6	5	6	1763	6076	0.74	51.1
n90p4t2s5	90	1671	6	6	6	5	6	55	697	0.73	3.5
n90p5t2s1	90	2039	7	7	7	6	7	81	694	0.33	2.1
n90p5t2s2	90	1988	7	7	7	6	7	6173	7172	0.5	125.3
n90p5t2s3	90	2064	7	7	7	6	7	23	369	0.65	1.4
n90p6t2s5	90	2478	9	9	9	8	9	2071	2056	0.41	13.0
n90p8t2s2	90	3200	12	12	12	11	12	191	506	0.18	0.8
ring_n15p0.9s1	370	27654	26	26	27	26	26	74	1858	24.91	352.0
ring_n15p0.9s5	392	31121	27	27	28	27	27	82	2532	101.19	902.4
ring_n15p1.0s1	420	35700	28	29	30	28	28	70	2527	363.73	3392.1
ring_n20p0.6s3	456	42134	32	32	33	32	32	114	2848	19.49	732.5
ring_n20p0.6s4	452	41756	32	32	33	32	32	89	2526	108.18	1314.2
ring_n20p0.7s3	534	57846	37	37	38	37	37	125	3282	130.56	2808.3
ring_n20p0.7s4	536	58641	38	38	39	38	38	126	3263	168.56	3544.7
ring_n20p0.7s5	518	55043	38	38	39	38	38	144	3200	258.86	1962.4
ring_n20p0.8s1	614	76875	44	44	45	44	44	157	3562	59.77	2418.7
ring_n20p0.8s4	610	76046	43	43	44	43	44	45	1358	225.97	tl
ring_n20p0.9s2	696	99162	49	49	50	49	50	16	890	1264.21	tl
ring_n20p0.9s3	686	95915	-	48	49	47	47	11	815	1405	3493.7
ring_n20p0.9s4	686	96132	-	48	49	47	47	7	824	1953.06	3544.6
ring_n20p0.9s5	706	101953	-	49	50	49	50	12	904	1648.07	tl
ring_n20p1.0s1	760	117800	-	-	53	39	50	0	797	tl	tl
Optimal #				33			41				

Table 3.3: Computational results for instances not solved at the root node.

Class	Instances	Solved [16] (any)	Solved [16] (best)	Solved B&P
`random`	56	45	42	**52**
`nsfnet`	32	**32**	**32**	**32**
`ring`	87	82	76	**83**
Total	175	159	150	**167**

Table 3.4: Summary of the computational results from Hoshino et al. [16] and our branch-and-price algorithm.

efforts, we also set a maximum time limit of 30 seconds for each attempt.

The results are divided in two batches. In the first batch, we considered those instances that we were able to solve at the root node. In the second batch, we include those instances for which the absolute gap at the root node was > 1, and one instance for which we were not able to fully explore the root node within the time limit.

Table 3.2 presents the results relative to the instances closed at the root node. Columns "$|V|$" and "$|E|$" report, respectively, the number of vertices and edges in the graph. Column "χ_{P}" is the chromatic number of the graph. Finally, column "Cols" reports the final size of the column pool, and column "Time (s)" is the solution time in seconds.

Table 3.3 presents the results relative to the instances that were not closed at the root node, either because the root node bound did not match the optimal solution value, or because the optimal solution value was not discovered by the initial heuristics, or both. We report under "B&P [16]" the best results obtained by any of the four implementations of Hoshino et al. [16], while under "New B&P" the results obtained by our algorithm. Columns "LB" and "UB" are the final lower and upper bounds, while column "hUB" is the upper bound obtained by the heuristic algorithm. Column "Nodes" displays the number of explored Branch-and-Bound nodes, while column "Cols" reports the number of columns generated. Finally, columns "rTime (s)" and "Time (s)" list, respectively, the root node (not including the initial heuristic) and the overall solution time (including the initial heuristic), in seconds. We do not report explicitly the final lower bound for "New B&P" because it is equal to the root lower bound "rLB" for all open instances.

Notice that we could not solve the root node of one instance (`ring_n20p1.0s1`), for which we provide a lagrangean lower bound $\mathrm{LB} = \lceil z_{\mathrm{LP^E}}/z_{\mathrm{Viol}} \rceil$, where $z_{\mathrm{LP^E}}$ is the solution of the last linear relaxation of the reduced master problem solved, and z_{Viol} is the last solution value found by the pricing problem. In summary, we managed to find the optimal solution to 41 of these 49 instances, and in 34 cases the solution was found in less than half an hour (1800s).

Table 3.4 concisely lists the number of instances solved to optimality, in each of the three classes, by the best algorithm in the literature (that of Hoshino et al. [16]) and by our branch-and-price algorithm. Column "Instances" lists the number of instances considered. In column "Solved [16] (any)" we report the number of instances solved by at least one of the four implementations of Hoshino et al. [16]. Column "Solved [16] (best)" reports the number of instances solved by the best of the four implementations, as listed in Table 5 of Hoshino et al. [16] (after removing the duplicate instances). Finally, column "Solved B&P"

Set	|U|	|B|	|T|
BAP1	20	10	12
BAP2	20	10	24
BAP3	40	10	24
BAP4	60	10	24

Table 3.5: Caracteristics of the generated BAP instances.

lists the number of instances solved by our branch-and-price algorithm.

3.6 Computational results: BAP

The same experimental setting used for the PCP has been used for the BAP. Solving the BAP, as noticed in Section 3.1.1, corresponds to solving the decision version of the PCP where we are asking whether or not a partitioned graph can be selectively colored with one color. For this reason, when solving the BAP, the algorithm performs an early termination if either the initial constructive heuristic or the ALNS produce a solution with one color; in this case, the decision problem has affirmative answer. On the other hand, if the initial solution uses multiple colors, the branch-and-price algorithm is started. If during the exploration of the branch-and-bound tree the lower bound rises above 2.0, however, then the algorithm is again terminated early; in this case, the decision problem has negative answer.

3.6.1 Instances

We generated four set of random instances. The distinctive features of each set are the number $|U|$ of ships (i.e., partitions), the number $|B|$ of berths, and the time horizon length $|T|$, as the vertex set is $V_{\text{BAP}} \subseteq U \times B \times T$. Table 3.5 reports these value for each instance set. Ten instances were generated in each set, according to the following procedure.

Each berth length was chosen uniformly at random in the interval $[2.0, 4.0]$; the ship lengths were chosen uniformly at random in the interval $[1.0, 6.0]$. The ship handling times were first drawn from the interval $[2, 6]$ proportionally to the ship length; successively, an uniform random number was drawn for each ship from the set $\{-1, 0, 1\}$ and was added to the handling time. The arrival time of each ship was chosen uniformly at random in the interval $[0, 8]$ for the instances with a 12-hours time horizon, and $[0, 18]$ for those with a 24-hours time horizon.

3.6.2 Algorithm

In this section we analyse the results ottained applying the heuristics and the branch-and-price algorithm for the PCP to the BAP instances. As shown in Table 3.6 the resulting graphs are of considerable dimensions with respect to both number of vertices and edges. Instances of sets BAP1, BAP2, and BAP3 were all solved to optimality by the initial heuristic, giving affirmative answer ($\chi_{\text{P}} = 1$) and therefore producing a feasible berth allocation schedule.

Instances of class BAP4, on the other hand, were much harder to solve. The combination of the constructive heuristic and the ALNS algorithm could not complete within the 1-hour time limit and was, therefore, truncated. For all these instances the heuristic found a colouring using two colours. In order to test the exact algorithm, this initial solution was fed to the branch-and-price solver with an additional hour of available computing time. However, in no case the solver was able to explore the root node within the time limit, and therefore the instances remain open.

From an analysis of the results, we can roughly group BAP instances into two categories: those for which a feasible berthing plan exists and could be found heuristically, and for which a berthing plan either does not exist or could not be found heuristically. The results seem to hint that there is little hope to solve this latter group of instances within a reasonable time limit.

We could solve to optimality instances with up to 40 ships and 10 berths. By comparison, Cordeau et al. [4] solved to optimality instances with up to 25 ships and 5 berths over a time horizon of one week, using a MIP model ran with a two-hour time limit; their objective was the minimisation of the total waiting and service time for the ships, considering that it also depends on which berth the ship is moored. Guan and Cheung [14] could solve to optmaility instances with up to 4 vessels using a MIP model, and up to 15 vessels with a Tree Search algorithm; the time horizon was of 1 week and the objective function minimised the waiting times. Monaco and Sammarra [28] solved instances with up to 30 ships and 7 berths with a compact MIP formulation, minimising the waiting and service times of the ships.

A direct comparison of these result would bare little value, as the objective functions are tremendously different. In general, we can state that — under certain circumstances — it seems feasible to solve the decision version of the BAP, modelling it as a PCP. This is especially useful for highly trafficked ports where a large number of ships need be served in a short time. The other algorithms proposed in the literature generally fail to provide optimal answers in these scenarios, being more useful on sparse instances where fewer vessels arrive over a larger time horizon.

3.7 Conclusions

In this manuscript we have studied the Partition Coloring Problem (PCP), a generalization of the classical Vertex Coloring Problem with several real world applications in telecommunications and scheduling. For the PCP, we propose a new ILP formulation with an exponential number of variables and a new Branch-and-Price algorithm to effectively tackle it. In order to obtain good quality feasible solutions in short computational time, we have developed a battery of heuristic algorithms. Thanks to the new exact algorithm, which exploit the heuristic solutions in its initialization phase, we were able to solve to proven optimality 167 out of 175 PCP instances from the literature. Extensive computational results have proven that the new Branch-and-Price framework improves on the previous state-of-the-art exact approaches from the literature.

We also applied the above algorithm to Berth Allocation Problem (BAP) instances, in order to assess the feasibility of solving the BAP as a PCP, as proposed by Demange et al. [7]. The

Instance	$\lvert V_{BAP}\rvert$	$\lvert E_{BAP}\rvert$	χ_P	Heur Time	BP Time
BAP1-0	465	19908	1	0.20	—
BAP1-1	252	8709	1	0.02	—
BAP1-2	191	4844	1	0.01	—
BAP1-3	578	28952	1	0.39	—
BAP1-4	390	19201	1	0.10	—
BAP1-5	530	27067	1	0.41	—
BAP1-6	550	27625	1	0.48	—
BAP1-7	672	35340	1	0.96	—
BAP1-8	324	13042	1	0.06	—
BAP2-0	530	27718	1	0.50	—
BAP2-1	2290	247238	1	55.15	—
BAP2-2	1809	185514	1	32.60	—
BAP2-3	1680	154170	1	21.00	—
BAP2-4	2422	266620	1	73.50	—
BAP2-5	1955	235677	1	46.18	—
BAP2-6	2420	318054	1	72.49	—
BAP2-7	2590	339640	1	83.49	—
BAP2-8	2565	311996	1	80.53	—
BAP2-9	2490	325782	1	93.41	—
BAP3-0	5003	861868	1	934.36	—
BAP3-1	4266	696475	1	559.37	—
BAP3-2	4789	750146	1	756.38	—
BAP3-3	4233	651228	1	762.15	—
BAP3-4	4810	947037	1	1930.58	—
BAP3-5	4329	668180	1	583.10	—
BAP3-6	4800	911949	1	858.80	—
BAP3-7	4484	799176	1	726.38	—
BAP3-8	4819	888290	1	868.67	—
BAP3-9	4590	878543	1	1569.36	—
BAP4-0	7390	1834851	$\{1,2\}$	3600	3600
BAP4-1	6470	1505789	$\{1,2\}$	3600	3600
BAP4-2	6860	1725947	$\{1,2\}$	3600	3600
BAP4-3	6309	1354479	$\{1,2\}$	3600	3600
BAP4-4	6922	1726307	$\{1,2\}$	3600	3600
BAP4-5	6066	1450902	$\{1,2\}$	3600	3600
BAP4-6	6792	1641599	$\{1,2\}$	3600	3600
BAP4-7	7410	1796181	$\{1,2\}$	3600	3600
BAP4-8	6984	1650984	$\{1,2\}$	3600	3600
BAP5-9	6492	1527453	$\{1,2\}$	3600	3600

Table 3.6: Computational results on the Berth Allocation Problem instances.

problem we consider is in some sense easier than other versions of the BAP from the literature, as we are only asking whether or not it is feasible to assign a set of ships to a set of berths, without taking into consideration the minimisation of waiting times or the total makespan. Under this simplifying assumption, we could solve instances with up to 40 vessels and 10 berths.

Finally, as a future research topic, it could be interesting to analyse the performances of the PCP model when it is adapted to take into account the classical objective function for the BAP. Let, for a berth allocation plan $S \in \mathscr{S}$, τ_S be the total waiting and service time for the ships allocated by S. The objective function (3.11) can then be modified in a bi-level fashion as $\sum_{S \in \mathscr{S}} (t_{\max} + 1 + \tau_S)\xi_S$ in order to first look for a solution which uses just one berth allocation plan and, in that case, look for the allocation plan which minimises the sum of the waiting and service time for the ships.

3.8 Acknowledgments

The authors thank Stefan Held for making the source code for the MWSS problem available online, and Edna Hoshino for providing detailed computational results for the branch-and-price algorithm of [16]. Enrico Malaguti is partially supported by MIUR (Italy), Grant PRIN 2015.

Bibliography

[1] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[2] Manoel Campêlo, Ricardo Corrêa, and Yuri Frota. Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89(4):159–164, 2004.

[3] Manoel Campêlo, Victor A Campos, and Ricardo C Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156(7):1097–1111, 2008.

[4] Jean-François Cordeau, Gilbert Laporte, Pasquale Legato, and Luigi Moccia. Models and tabu search heuristics for the berth-allocation problem. *Transportation science*, 39 (4):526–538, 2005.

[5] Denis Cornaz, Fabio Furini, and Enrico Malaguti. Solving coloring problems as maximum weight stable set problems. *Discrete Applied Mathematics*, 2016. (to appear).

[6] Marc Demange, Jérôme Monnot, Petrica Pop, and Bernard Ries. On the complexity of the selective graph coloring problem in some special classes of graphs. *Theoretical Computer Science*, 540:89–102, 2014.

[7] Marc Demange, Tınaz Ekim, Bernard Ries, and Cerasela Tanasescu. On some applications of the selective graph coloring problem. *European Journal of Operational Research*, 240(2):307–314, 2015.

[8] Marc Demange, Tınaz Ekim, and Bernard Ries. On the minimum and maximum selective graph coloring problems in some graph classes. *Discrete Applied Mathematics*, 204:77–89, 2016.

[9] Guy Desaulniers, Jacques Desrosiers, and Marius Solomon, editors. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[10] Yuri Frota, Nelson Maculan, Thiago F Noronha, and Celso C Ribeiro. A branch-and-cut algorithm for partition coloring. *Networks*, 55(3):194–204, 2010.

[11] Fabio Furini and Enrico Malaguti. Exact weighted vertex coloring via branch-and-price. *Discrete Optimization*, 9(2):130 – 136, 2012. ISSN 1572-5286.

[12] Fabio Furini, Enrico Malaguti, and Alberto Santini. Exact and euristic algorithms for the Partition Colouring Problem. *Submitted to Computers & Operations Resarch*, pages 1–17, 2017.

[13] Stefano Gualandi and Federico Malucelli. Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, 24(1):81–100, 2012.

[14] Yongpei Guan and Raymond K Cheung. The berth allocation problem: models and solution methods. *OR Spectrum*, 26(1):75–92, 2004.

[15] Stefan Held, William Cook, and Edward Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4): 363–381, 2012.

[16] Edna A Hoshino, Yuri A Frota, and Cid C De Souza. A branch-and-price approach for the partition coloring problem. *Operations Research Letters*, 39(2):132–137, 2011.

[17] Akio Imai, Ken'Ichiro Nagaiwa, and Chan Weng Tat. Efficient planning of berth allocation for container terminals in asia. *Journal of advanced transportation*, 31(1):75–94, 1997.

[18] Akio Imai, Etsuko Nishimura, and Stratos Papadimitriou. The dynamic berth allocation problem for a container port. *Transportation Research Part B: Methodological*, 35(4): 401–417, 2001.

[19] Akio Imai, Etsuko Nishimura, and Stratos Papadimitriou. Berth allocation with service priority. *Transportation Research Part B: Methodological*, 37(5):437–457, 2003.

[20] Akio Imai, Xin Sun, Etsuko Nishimura, and Stratos Papadimitriou. Berth allocation in a container port: using a continuous location space approach. *Transportation Research Part B: Methodological*, 39(3):199–221, 2005.

[21] Rhyd MR Lewis. *A Guide to Graph Colouring*. Springer, 2015.

[22] Guangzhi Li and Rahul Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, page 1. Citeseer, 2000.

[23] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17:1–34, 2010.

[24] Enrico Malaguti, Michele Monaci, and Paolo Toth. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, 20(2):302–316, 2008.

[25] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174–190, 2011.

[26] Dániel Marx. Graph colouring problems and their applications in scheduling. *Periodica Polytech., Electr. Eng*, 48(1-2):11–16, 2004.

[27] Anuj Mehrotra and Michael A Trick. A column generation approach for graph coloring. *informs Journal on Computing*, 8(4):344–354, 1996.

[28] M Flavia Monaco and Marcello Sammarra. The berth allocation problem: a strong formulation solved by a lagrangean approach. *Transportation Science*, 41(2):265–280, 2007.

[29] Panos M Pardalos, Thelma Mavridou, and Jue Xue. The graph coloring problem: A bibliographic survey. In Panos M Pardalos and Ding-Zhu Du, editors, *Handbook of combinatorial optimization*, pages 1077–1141. Springer, 1998.

[30] Petrica C Pop, Bin Hu, and Günther R Raidl. A memetic algorithm for the partition graph coloring problem. In *Extended Abstracts of the 14th International Conference on Computer Aided Systems Theory, Gran Canaria, Spain*, pages 167–169, 2013.

[31] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4): 455–472, 2006.

[32] Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, Submitted, 2016.

[33] Yavuz B Türkoğulları, Z Caner Taşkın, Necati Aras, and İ Kuban Altınel. Optimal berth allocation and time-invariant quay crane assignment in container terminals. *European Journal of Operational Research*, 235(1):88–101, 2014.

[34] François Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130(2):249–294, 2011. doi: 10.1007/s10107-009-0334-1.

[35] Alexander Aleksandrovich Zykov. On some properties of linear complexes. *Matematicheskii sbornik*, 66(2):163–188, 1949.

4 Maritime seaside logistics: the feeder network design problem

Abstract In this chapter the design of a container liner shipping feeder network is decided, by choosing which port to serve during many rotations which start and end at a hub, and can take an arbitrary shape, potentially visiting the hub multiple times during a fixed time horizon. We take into account many operational constraints, such as variable speeds and cargo transit times, and accordingly generated realistic instances, based on the LinerLib benchmark suite. We solved the problem with a branch-and-price algorithm, which is able to solve most instances to optimality. Furthermore, we performed a comprehensive scenario analysis to evaluate the sensitivity of the solutions to changes in external conditions and internal policies, and to formulate practical guidelines for network planners.

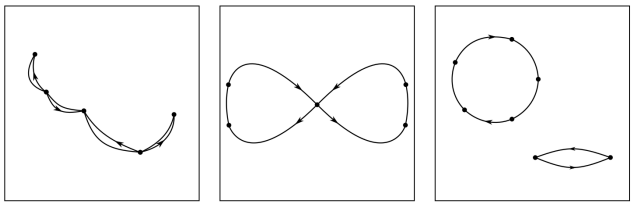
4.1 Introduction

Container liner shipping is the main freight transportation service used to move large quantities of cargo over long distances. As opposed to tramp shipping, that describes individual ships operated to fulfil the transportation requests as they come, and that decide which one is more convenient for them to accept, liner ships operate along fixed routes and according to a published schedule.

According to Lloyd's [22], 75% of internationally traded goods by volume was transported by sea in 2008 and this figure is set to increase: it was already an estimated 85% during 2013, according to Drewry [17]. Containerised goods account for a relatively small portion of the shipped volume, but their value per volume unit is higher than that of any other kind of goods, and around 52% of maritime commerce by value is shipped in containers.

Furthermore, the environmental impact of moving goods by sea is certainly non-negligible: World Shipping Council [38] estimated that 2.7% of the global greenhouse gas emissions is accounted by international maritime shipping, and a quarter of this figure is due to container shipping.

This chapter is based on the contents of: Alberto Santini, Stefan Ropke, and Christian E.M. Plum. A branch-and-price approach to the Feeder Network Design Problem. *European Journal of Operational Research (under revision)*, pages 1–16, 2017.



(a) Conveyor Belt (b) Butterfly (c) Cycle and Pendulum

Figure 4.1: Different types of rotations. Reprinted from Andersen [2] with permission.

These data make clear that the impact of employing OR tools for the optimisation of liner container vessels operations can yield enormous results in terms of both business and environmental value. And yet, operational research methods have not frequently focussed on liner shipping, especially when comparing the amount of literature relevant to this field versus other fields of logistics and transportation (see, e.g., Christiansen et al. [11]).

In this paper, we consider the Feeder Network Design Problem (FNDP), which arises when planning the routes of liner container vessels in regional feeder networks. Intercontinental container routes are operated by big vessels (up to 18000 TEU, Twenty-foot Equivalent Units) that only call the main hub ports. These hubs are characterised by an extensive infrastructure that makes them suitable to operate with enormous quantities of containers and to efficiently load and unload extremely big vessels. Networks of smaller vessels load the containers delivered at the hubs and transport them to smaller ports in a particular region. At the same time, they collect containers at the small ports and unload them at the hubs, where they will later be shipped to their final destination on the intercontinental routes. In short, liner shipping is organised in *hub-and-spoke* networks. While more than one hub can exist in the same region, in this work we focus on single-hub feeder networks.

Since liner vessels operate according to a fixed schedule, the operator issues a public timetable composed by a sequence of port calls together with their day and time. It is clearly convenient that such a schedule be periodic, so that each port is called on a certain day of the week that does not change. For this reason, the time horizon considered when planning feeder routes is a multiple of one week.

Due to the nature of the feeder network, moreover, each schedule starts and ends at the hub port (even though it can visit the hub more than once). For this reason, we usually refer to the route taken by a vessel as a *rotation*. Rotations are sometimes classified according to their structure: Figure 4.1, reprinted from Andersen [2], depicts cycles, pendulum routes, butterfly routes, and conveyor belts. In our work, we allow the creation of routes with arbitrary shapes, and that visit the hub any number of times within the time horizon. Therefore, all rotation types presented in Figure 4.1 are allowed to be constructed, as particular cases of our routes.

The network designer is also faced with the decision of selecting which ports to serve. The liner company earns revenue for each container it moves at each served port. The company might also be requested to pay a penalty when not serving a port, if it had contracts in place with terminal or logistic operators. Another possibility is that the company decides to outsource the service to another company (for example, by buying capacity on a competitor's

vessel). As we will show in the following sections, our model is flexible enough to allow for each of these scenarios.

We assume that both the delivery and pickup demands of a port are determined and known in advance. In practice, the data we use comes from forecast of future demand, possibly based on historical observation or on contracts in place. We also assume that, for each port, the network designer has two decisions to make: whether or not to pickup demand at the port, and whether or not to deliver demand to the port (these decisions are independent). Once the decision is made to serve, e.g. the pickup demand at the port, the total amount of cargo will be picked up in one visit. In other words, we do not allow split pickup and delivery.

The quantity to maximise is the total revenue from served cargoes, minus the cost of the rotations and the eventual penalties or outsourcing costs. The cost of rotations includes port taxes and calling fees and the vessel's bunker cost, i.e. the cost of the fuel used by the ship. This latter cost is particularly important for two reasons: first, it makes up a considerable share of an operator expenses, while its price remains very volatile; second, it is greatly impacted by the steaming speed of a vessel, with the relationship between the two being approximately cubic:

$$\text{cost}(s) = \left(\frac{s}{s^*}\right)^3 \cdot \text{cost}(s^*)$$

where s is the steaming speed, $\text{cost}(s)$ is the cost incurred to sail for a unit of time at speed s, s^* is the design speed, and $\text{cost}(s^*)$ is the cost to sail for a unit of time at the design speed. Both s^* and $\text{cost}(s^*)$ are known in advance.

The constraints the operator is faced with are the limited capacity and number of vessels available, the fact that ports observe certain closing time windows (e.g. ports that are closed for operations at night), that certain goods might have a maximum time span during which they can travel (e.g. perishable goods) and that ports have a maximum draught and therefore not every vessel can enter every port.

In short, the FNDP requires us to come up with certain routes for a fleet of vehicles (vessels, in this case) that abide particular constraints. The problem is, therefore, related to the well-known Vehicle Routing Problem (VRP) and many of its variants: it combines elements of the *capacitated* VRP since each vessel has a maximum container capacity, the VRP with *time windows*, the VRP with *heterogeneous fleet* since each vessel can differ from the others, the VRP with *pickups and deliveries* since we have a flow of cargo both from and to the hub, the *multi-period* VRP because the vehicles can get back to the hub multiple times. We refer the reader to the recent book by Toth and Vigo [35] for a comprehensive review of various VRP variants.

In these variants of the VRP, however, the set of customers to be served is given and no selection need be performed. With this respect, the FNDP is then related to the family of Orienteering Problems (OP), and in particular to the multi-vehicle variants which ares present in the literature under various names, including Team Orienteering Problem (TOP), VRP with Profit Collection, or VRP with Customer Selection. The TOP has received considerable attention in the recent years (see, for a review, Archetti et al. [4]). While exact algorithms based on branch-and-price exist for the TOP [7] and for its capacitated version [3], to the best of our knowledge no exact algorithm has been developed for cases which include time windows, or a pickup-and-delivery component.

The main contributions of this paper are three-fold. First, we introduce a mathematical model and an extended formulation for the Feeder Network Design Problem, taking into account the principal real-life constraints that container vessel operators have to face. Second, we provide a state-of-the-art algorithm, which is able to solve to optimality (or with very small gaps) realistic instances. Third, we perform a wide variety of scenario analyses, from which we dystill general principles an operator can apply at the strategic, tactical, and operational levels.

4.2 Literature review

In this section we summarise recent literature on two important aspects of maritime optimisation: liner network design, and speed optimisation. For more general reviews of research in maritime optimisation, we refer the reader to Christiansen et al. [11], Christiansen et al. [12] and Christiansen et al. [13].

Liner network design

An introduction to liner shipping is given in Brouer et al. [8], where the authors also present a benchmark model for LinerLib (see Løfstedt et al. [23]), the main instance library used for liner shipping problems. The benchmark model for the Liner Network Design Problem (LNDP), is solved by means of a heuristic based on Tabu search and column generation. A broad introduction to operational research methods in container liner shipping is given in the survey by Meng et al. [25].

A heuristic, a column generation algorithm and a Benders decomposition algorithm, all coupled with an iterative search routine, are presented by Agarwal and Ergun [1] to solve the combined ship scheduling and cargo routing problem, with weekly frequencies and transhipments. The authors allow multiple visits to the same port (as long as they happen in different days of the week), but do not consider transhipment costs. They test their approach on instances with at most 20 ports and 100 ships.

An exact method is presented by Reinhardt and Pisinger [31] to solve the network design and fleet assignment problem. The authors use a branch-and-cut algorithm to solve instances up to 15 ports, while considering transhipments, a heterogeneous fleet and allowing butterfly routes.

Mulder and Dekker [26] propose a heuristic approach to the combined problem of fleet design, ship scheduling and cargo routing. The authors first cluster ports by proximity and then design a hub-and-spoke hierarchy, in which each cluster is served by a feeder network. They test their approach on the Asia-Europe trade lane, comprising 58 ports.

Plum et al. [28] propose to switch the classic arc-flow formulation for the LNDP, with a service-flow based formulation. Similarly to our work, they consider the Baltic and Western African scenarios of the LinerLib and propose a mixed-integer linear formulation to maximise the operator's profit, under a weekly frequency constraint and allowing multiple calls to the same port (thereby allowing an arbitrary number of butterfly ports). The model is solved with a commercial solver.

Criterion	Value	Criterion	Value
Optimisation Criterion	Profit	Shipping Market	Liner
Decision maker	Owner	Explicit fuel price	Yes
Freight rate	Yes	Fuel consumption	Cubic
Leg-by-leg optimal speed	Yes	Speed function of payload	No
Logistical context	Pickup and delivery	Size of fleet	Multiple ships
More ships an option	Yes	Inventory costs	No
Modal split	No	Ports included	Yes

Table 4.1: Classification of the present paper under the taxonomy proposed by Psaraftis and Kontovas [29].

Another heuristic method, based on column generation, for the network design and cargo flow problem in liner shipping is presented by Wang and Meng [37]. The authors apply the column generation algorithm to a mixed-integer non-linear non-convex formulation, which models weekly services and maximum transit times (which they refer to as *deadlines*). They test their approach on the Asia-Europe trade lane, considering 12 ports.

Plum et al. [27] presents arc-flow and path-flow models for the single-vessel liner shipping service design, in which an operator has to optimise the best-paying demand (pickups and deliveries) for a round-trip route operated by a single vessel, taking into account maximum transit times. The authors propose a branch-and-cut algorithm, which is able to solve instances with up to 25 ports.

Speed optimisation

A general review and a taxonomy of existing literature related to speed optimisation for efficient and green maritime transportation is given in Psaraftis and Kontovas [29]. Table 4.1 shows the classification of our work under the proposed taxonomy.

Chang and Wang [10] and Cariou [9] study the economical impact and sustainability of slow steaming in liner shipping. In the first work, the authors conclude that speed optimisation is best employed as a dynamic process, which depends on both charter rates and fuel prices. In the second work, the author concludes that slow steaming is a long-term sustainable strategy on main container trade lanes, if the bunker price is at least $350-$400.

More details on the relation between sailing speed, incurred costs and emissions is given in Psaraftis and Kontovas [30] and Kontovas [21], in which the authors study the trade-offs involved in speed optimisation in broader shipping scenarios. In particular, Psaraftis and Kontovas [30] consider many real-life factors impacting on the relationship between speed and costs, stressing the importance of considering hotel costs to adjust the purely cubic cost function, the dependency of the cost function on the payload, the impact of external events such as weather conditions, and of the maintenance state of the ship (e.g. hull condition). They also mention the importance of *en route* inventory costs, which clearly tend to increase at lower speeds on long-distance routes. Kontovas [21], on the other hand, focuses on the relationship between fuel consumption and CO_2 emissions.

The work of Wang and Meng [36] studies the leg-by-leg optimisation of sailing speeds, once the routes have already been fixed. The author consider transhipments and container

routing, and propose a Mixed-integer non-linear convex model, for which they give an outer approximation scheme. They test their approach on an instance with 46 ports and 11 fixed routes.

4.3 Model

The time in the planning horizon is divided into m time intervals, and it is modelled by the set $T = \{1, \dots, m\}$. The size of this discretisation depends on the size of the region considered. As mentioned in Section 4.1, it is convenient that the time horizon be a multiple of one week. If, for example, we consider one month as planning horizon, we can then deploy four ships on each rotation spacing them out by one week, so that each port is visited weekly.

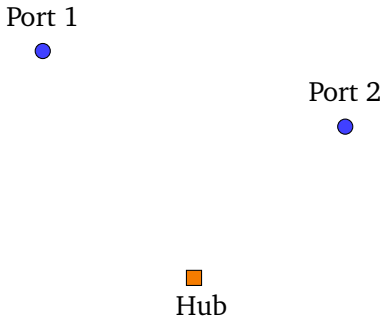
Let $n \in \mathbb{N}$ be the number of ports in the region, excluding the hub. We model each port twice, once as a pickup and once as a delivery port. The set of all ports is denoted by $P' = \{0, 1, \dots, n, n+1, \dots, 2n\}$ where 0 represents the hub, $P^- = \{1, \dots, n\}$ is the set of delivery ports, and $P^+ = \{n+1, \dots, 2n\}$ is the set of pickup ports. Ports i and $n+i$ represent the same physical port.

We also define the set $P = P' \setminus \{0\}$ of all ports excluding the hub. Each port $i \in P^-$ has a delivery demand $d_i \geq 0$ and each port $j \in P^+$ has a pickup demand $p_j \geq 0$. Furthermore, each port has a maximum draught $m_i^{\mathrm{p}} > 0$, meaning that vessels with a draught greater than m_i^{p} cannot enter the port. Each port has a handling time $h_i \geq 0$, that is the number of time intervals needed to load (if $i \in P^+$) or unload (if $i \in P^-$) containers at that port. Since the quantity of containers to be moved at each port is known in advance, an accurate estimate of h_i can be given for each port. We assume that when the vessel visits the hub, its handling time h_0 can be estimated only based on the vessel type, independently from the quantity of cargo it carries.

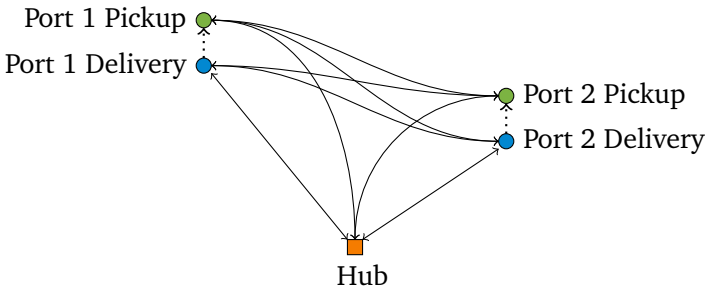
Each port has a number of closing time windows which can be used not only to model times at which the port is closed, but also for many other purposes. Here we give a couple of examples.

- If i is a pickup port that receives most fo the goods it exports (for example, via a freight train) at a certain time $t_0 > 1$, then i can be marked as closed until a time $t_1 \geq t_0$ at which we can assume the goods are ready to be loaded onto a ship.
- If we use a time horizon representing 2 weeks, and have a delivery port j which is served weekly by a freight train (at times t_0, t_1), and which imports perishable goods that can stay in the yard at most t' time units before they are loaded on the train, we can mark the port as open only during time windows $[t_0 - t', t_0]$ and $[t_1 - t', t_1]$. A rotation will visit j, for example, during the first time window; the second ship deployed on the same rotation will visit it during the second time window, thereby guaranteeing a weekly visit, and that the goods are not spoiled by staying too long in the yard.

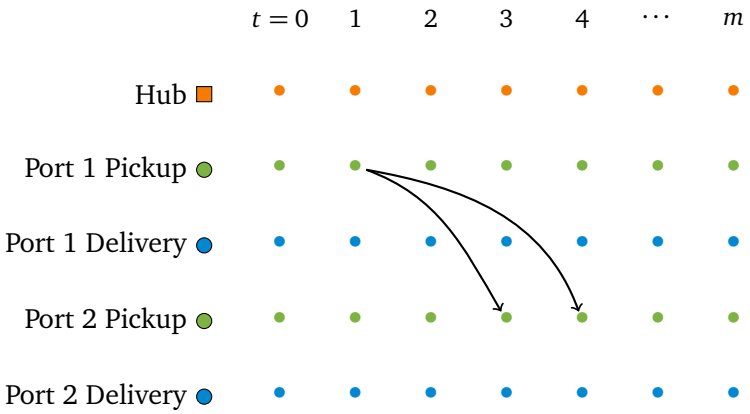
For every pair of ports i, j we have a symmetric distance $\Delta_{i,j} \geq 0$. The distances satisfy the triangle inequality, and are zero for elements modelling the same port, i.e. $\Delta_{i,i} = 0$ for all $i \in P'$, and $\Delta_{i,n+i} = 0$ for all $i \in P^-$.



(a) Configuration of the instance, with the ports in their geographical positions.



(b) In this graph, the ports have been duplicated (for pickup and delivery) and arcs represent the possibility to sail from one port to another.



(c) The time-expanded graph for a particular vessel, where each node is copied once for each time instant. Two arcs are depicted, representing sailing between two ports at different speeds.

Figure 4.2: Modelling of a simple instance, with one hub and two ports.

We deal with a heterogeneous fleet of k vessels, modelled by set $V = \{1, \dots, k\}$. Each vessel v has a capacity $Q_v > 0$ and a draught $m_v^{\mathrm{v}} > 0$. We consider the draught of a vessel as fixed, even though in reality it can change slightly with the amount of load a vessel is carrying. We refer the reader to, e.g., Glomvik Rakke et al. [19], Battarra et al. [6], Malaguti et al. [24] for recent works on maritime routing under draught limits. Each vessel v has a set of possible sailing speeds $\Omega^v = \{\omega_1^v, \dots, \omega_{s_v}^v\}$ expressed in nautical miles per discrete time unit (i.e., the speed is expressed in multiple of knots); $s_v \in \mathbb{N}$ is the number of speeds associated with vessel v.

We take into account the following cost components:

- Time charter cost: a fixed daily cost that the operator pays to charter the vessel. We associate to each vessel v the time charter cost Γ_v^{TC} (in dollars per time interval).
- Hotel cost: the cost incurred when keeping a vessel moored, but operational. We denote the hotel cost for vessel v as Γ_v^{H} (in dollars per time interval).
- Bunker cost: the sailing cost, which depends on the cruising speed of the vessel. Given a vessel v and a sailing speed $\omega_k^v \in \Omega^v$, we consider the associated cost Γ_{vk}^{B} (in dollars per time interval).
- Movement costs: the unit costs relative to the handling of the cargo at a port. Given a vessel v and a port i, since we know the quantity of cargo that needs to be handled in i, we can compute the total movement cost and denote it as Γ_{vi}^{M} (in dollars).
- Fixed port call fee: a flat cost to pay when any vessel calls a port i. This is denoted as Γ_i^{PF} (in dollars).
- Variable port call fee: a cost to pay when calling a port, but dependent on the capacity of the vessel. A vessel v calling port i will pay $\Gamma_{vi}^{\mathrm{PV}}$ (in dollars).

Serving requests generates revenue, as the operator earns a certain fee per Forty-Foot Equivalent Unit (FFE) picked up or delivered. The revenue is, for commercial reasons, not uniform and delivering an FFE to a certain port could earn more or less than delivering to another port. Therefore, we associate to each port i a revenue $R_i > 0$ (in dollars) which express the total earned by the operator when a vessel serves the port. Analogously, we associate a penalty $\alpha_i \geq 0$ (in dollars) to be paid by the shipping operator if it decides to skip service at the port. This penalty can be also used to model the cost of outsourcing the service.

Figure 4.2a shows a simple instance of the problem, with one hub and two ports. In Figure 4.2b we create one node for each element of P' and draw solid arcs to represent that a ship can sail from any port to any other port. The dotted arcs represent a ship that, at the same port, first performs a delivery and then a pickup.

4.3.1 Graphs

For each vessel v we create a space-time graph $G_v = (N, A_v)$. The node set is common to all graphs and is defined as $N = \{\sigma, \tau\} \cup P' \times T$ where σ and τ are, respectively, a source and

sink node. All other nodes (i,t) represent a port at a specific time instant; G_v is, therefore, a time-expanded graph. The arcs A_v can be partitioned in four sets:

- Starting arcs of type $(\sigma,(0,t))$, that link the source node with a node representing the hub.
- Ending arcs of type $((0,t),\tau)$, that link a node representing the hub with the sink node.
- Delivery-to-pickup arcs of type $((i,t),(i+n,t'))$, that link the delivery and pickup operations at the same physical port.
- Sailing arcs of the type $((i,t),(j,t'))$, with $j \neq i+n$ if $i,j \in P$, that represent the sailing from port $i \in P'$ to port $j \in P'$.

The time used for the operations at the destination node of the arc is precomputed and included in the arc itself. To understand how this is done, consider the example of a sailing arc from port $i \in P$ to port $j \in P$, modelling the sailing of vessel $v \in V$, starting at time t and sailing at speed $\omega_k^v \in \Omega^v$. Assume that the draught limits are respected ($m_i^{\mathrm{p}} \geq m_v^{\mathrm{v}}$ and $m_j^{\mathrm{p}} \geq m_v^{\mathrm{v}}$), that i is open at time t, and that visiting j immediately after i does not violate capacity constraints ($p_i + p_j < Q_v$ if $i,j \in P^+$, $d_i + d_j < Q_v$ if $i,j \in P^-$, $p_i + d_j < Q_v$ if $i \in P^+, j \in P^-$). If these conditions do not hold, then the arc is not created.

The arrival time instant of vessel v will be $t_{\mathrm{arr}} = t + \Delta_{ij}/\omega_k^v$. Let t' be the time instant in which the handling operations at j can be completed; if t_{arr} corresponds to a time when port j is open, then $t' = t_{\mathrm{arr}} + h_j$, otherwise a waiting time has to be factored in, as the ship cannot berth or be operated on, while the port is closed. We can then create the arc $((i,t),(j,t'))$.

This process is repeated for each pair of ports, for each vessel $v \in V$, and for each speed $\omega \in \Omega^v$. The delivery-to-pickup arcs are created similarly, while the starting and ending arcs are created for all nodes representing the hub. A starting arc of type $(\sigma,(0,t))$ means that the vessel is leaving the hub at time t and therefore has been waiting idly at the hub during the time period $[0,t]$. Analogously, an ending arc of type $((0,t),\tau)$ represents a ship concluding its rotation at time t and waiting idly during time period $[t+1,m]$.

Notice that all costs, as well as the revenue, can be modelled on the arcs. For example, the cost of a sailing arc $a = ((i,t),(j,t'))$ is set to:

$$c_a = \Gamma_v^{\mathrm{TC}}(t'-t) + \Gamma_v^{\mathrm{H}}(t'-t_{\mathrm{arr}}) + \Gamma_{vk}^{\mathrm{B}}(t_{\mathrm{arr}}-t) + \Gamma_{vj}^{\mathrm{M}} + \Gamma_j^{\mathrm{PF}} + \Gamma_{vj}^{\mathrm{PV}} - R_j \tag{4.1}$$

Figure 4.2c shows the time-expanded version of the graph depicted in Figure 4.2b, for a particular vessel (for simplicity, we omit vertices σ,τ). In the figure, we only draw two arcs: they represent a ship sailing from "Port 1 Pickup" to "Port 2 Pickup". We can imagine, for example, that the first arc represents the ship leaving Port 1 at time 1, arriving at Port 2 at time 2, and finishing the loading operations at time 3. The second arc again models leaving Port 1 at time 1 but, sailing more slowly, Port 2 is reached only at time 3, and the loading operations are completed at time 4.

4.3.2 Integer formulation

We define a route for vessel v as a succession of consecutive arcs in A_v such that the first starts at σ and the last ends at node τ. If every port $i \in P$ is visited at most once, and if the capacity constraint is not violated, we say that the route is feasible. The set of feasible routes for vessel v is denoted as R_v and the set of all feasible routes as $R = \bigcup_{v \in V} R_v$. Notice that it is possible to assign naturally a cost c_r to each route r by summing the costs of the arcs that compose the route. Finally, let $\varepsilon_{ri} \in \{0,1\}$ be a parameter with value 1 iff port i is visited by route r. We can formulate a model for the FNDP by considering binary variables x_r taking value 1 iff route $r \in R$ is part of the solution:

$$\text{(MP)} \qquad \min \sum_{r \in R} c_r x_r + \sum_{i \in P} \alpha_i \left(1 - \sum_{r \in R} \varepsilon_{ri} x_r\right) \tag{4.2}$$

$$\text{s.t.} \sum_{r \in R} \varepsilon_{ri} x_r \leq 1 \qquad \forall i \in P \tag{4.3}$$

$$\sum_{r \in R_v} x_r \leq 1 \qquad \forall v \in V \tag{4.4}$$

$$x_r \in \{0,1\} \qquad \forall r \in R \tag{4.5}$$

The objective function (4.2) minimises the total cost of selected routes and the penalties paid. (A solution yelds a profit if the value of the objective function is negative.) Constraint (4.3) ensures that each port is visited by at most one vessel, constraint (4.4) guarantees that we do not use more vessels than available, and finally constraint (4.5) specifies the variables' domain. Notice that, unlike problems in the VRP family, where the requirement that all customers need be served leads to set-covering $\geq$-constraints, our model has $\leq$-constraints in (4.3).

Problem (MP) is called the *master problem*. An obvious drawback of (MP) is that sets R_v is too large to enumerate in practice, since the number of feasible routes grows exponentially with the size of the graphs. We then consider a version of (MP) where the sets R_v are substituted with much smaller sets R'_v; this new problem is called the *restricted master problem* (RMP). The idea behind the branch-and-price algorithm we propose is to solve (RMP) by branch-and-bound: at each node the linear relaxation of (RMP) is solved and new columns are added to R'_v. In order to find promising columns to add, we solve a *pricing subproblem* $(\text{SP})_v$ for each vessel v, that will produce new columns with negative reduced cost. From dual theory, we know that columns with negative reduced cost entering the base will improve the objective function of the relaxed (RMP). A node is considered explored when it is not possible to find any negative reduced cost column. Let $\pi_i \geq 0$ be the dual variables of (4.3) and $\mu_v \geq 0$ be the dual variables of (4.4). Notice that the objective function can be rewritten as

$$\sum_{r \in R} \left(c_r - \sum_{i \in P} \alpha_i \varepsilon_{ri}\right) x_r + \sum_{i \in P} \alpha_i \tag{4.6}$$

and the last sum is a constant. Therefore, the dual cost of a column corresponding to route

$r \in R_v$ is:

$$\hat{c}_r = c_r + \sum_{i \in P} (\pi_i - \alpha_i)\varepsilon_{ri} + \mu_v \tag{4.7}$$

The dual cost is given by the original route cost; then, for each visited port $i \in P$, the dual price π_i is added and the corresponding penalty α_i is removed; finally, a dual price μ_v is paid, only depending on the used vessel $v \in V$.

4.4 Solution of the pricing subproblem

The pricing subproblem $(\text{SP})_v$ is a shortest path problem with resource constraints (SPPRC), as routes for vessel v are paths in G_v from the source node σ to the sink node τ, and the resource constraints are used to ensure the routes' feasibility. In some sense, this problem can also be seen as an elementary SPPRC (ESPPRC). Graphs G_v are acyclic, and therefore any path would be elementary. In our case, however, we require that every port $i \in P$ is visited at most once, i.e. that every subset of nodes $N_i = \{(i,t) \mid t \in T\}$ has at most one inbound and one outbound arc. In the rest of this paper we will refer to this property when we speak of elementariness.

Notice that the elementariness requirement can be dropped, and routes that visit the same port multiple times can be generated, as long as the corresponding columns are then removed by suitable branching rules in the master problem (as we will explain in Section 4.5.3). For this reason, in this section we propose algorithms to solve both the ESPPRC and the SPPRC.

4.4.1 Greedy-randomised heuristic for the ESPPRC

We can attempt to find negative-reduced-cost elementary path with a simple greedy algorithm that builds a path starting in σ and then proceeds by choosing one random arc among the K_1 outgoing arcs of least reduced cost that do not close a cycle, until τ is reached. The same procedure can also be applied backward, starting in τ and ending at σ. After the path is constructed, we can check that the capacity constraint has not been violated by a single pass over the list of visited ports; if the path is not capacity-feasible, or if its reduced cost is non-negative, it is discarded. The algorithm can be applied in both directions many times, as its computational time is very small, thereby increasing the chances that a feasible path of negative reduced cost is found.

4.4.2 Exact dynamic programming algorithm for the ESPPRC

The ESPPRC can be solved exactly via dynamic programming by using a label-setting algorithm (see, e.g., Irnich et al. [20] for a review on solution methods for shortest path problems with resource constraints). We associate a label L to each partial path from σ to a node (i,t). The label components are similar to those introduced by Dell'Amico et al. [15] in the context of a Vehicle Routing Problem with simultaneous distribution and collection:

- $v \in N$, the current node (i,t) the path is visiting.

- $\Pi \in \mathbb{N}$, the amount of cargo that vessel v can pick up after visiting the current port.
- $\Delta \in \mathbb{N}$, the amount of cargo that vessel v can deliver after visiting the current port.
- $\vec{V} \in \{0,1\}^P$, a binary vector whose j-th component is 0 iff port $j \in P$ can be visited at some point after the current port. This vector is used to keep track of visited ports so to ensure that the route is elementary.
- $C \in \mathbb{R}$, the cost associated with the partial path. This is given by the sum of the costs of the arcs traversed, plus the prices π_j paid and minus the penalties α_i avoided at visited ports. If $v = \tau$ we also add the dual price μ_v.

An initial label is created with $v = \sigma$, $\Pi = \Delta = Q_v$, $\vec{V} = \vec{0}$, and $C = 0$. Notice that a label L can then be thought of as an element of the space $S = N \times \{0, \dots, Q_v\}^2 \times \{0,1\}^{|P|}$ equipped with a cost function $C : S \to \mathbb{R}$.

In the following we will use the convenient convention that $p_i = 0$ for all $i \in P^-$ and $d_j = 0$ for all $j \in P^+$. Let us describe how to update a label L associated with a partial path to (i,t) when the path is extended to a node (j,t'), with $j \in P$, along a sailing arc. We will call the new label $L' = (v', \Pi', \Delta', \vec{V}')$ and its cost C'. First of all we check if such an extension is feasible. This is the case if there is an arc a connecting (i,t) with (j,t'), if j is a visitable port (i.e. the j-th component of $\vec{V}$ is 0) and if there is enough available capacity on the vessel (i.e. $\Pi \geq p_j$ and $\Delta \geq d_j$). If the extension is feasible, we set the components of new label L' as follows: component v' will be set to (j,t'); $\vec{V}'$ will be equal to $\vec{V}$ except for the j-th coordinate, that will be set to 1, as it is not possible to visit port j again; the cost component will be updated as $C' = C + c_{av} + \pi_j - \alpha_j$. Let us now consider component Π': since collecting cargo consumes the associated resource "amount of load that can be picked up after visiting j" of a number of units equal to the amount of cargo we collect, while delivering it does not, it will updated as $\Pi' = \Pi - p_j$. Component Δ', on the other hand, represent the resource "amount of load that can be delivered after visiting j" and both a pickup and a delivery operation reduce this resource by a number of units equal to the amount of cargo picked up or delivered. Therefore, it is updated as $\Delta' = \min\{\Pi - p_j, \Delta - d_j\}$.

Similar rules are followed when extending along other type of arcs. In particular, an extension to the hub $(0,t)$ is always possible, and will have the effect of resetting components Π and Δ to their original value of Q_v. Finally, we charge the dual cost μ_v associated with the vessel on the arcs used to reach the sink (i.e. on the "ending arcs" described in Section 4.3.1); the cost of the corresponding label will be updated as $C' = C + \mu_v$.

We now define a dominance criterion to compare two labels $L_1 = (v, \Pi_1, \Delta_1, \vec{V}_1, C_1)$ and $L_2 = (v, \Pi_2, \Delta_2, \vec{V}_2, C_2)$ representing partial paths up to the same node. We say that label L_1 dominates L_2, and we write $L_1 \prec L_2$, if: (a) $C_1 \leq C_2$; (b) $\vec{V}_1 \leq \vec{V}_2$ component-wise; (c) $\Delta_1 \geq \Delta_2$; (d) $\Pi_1 \geq \Pi_2$; (e) at least one of the previous inequalities is strict.

4.4.3 Exact dynamic programming algorithm for the SPPRC

Since the ESPPRC is an $\mathcal{NP}$-hard problem, we can consider the labelling algorithm on a relaxed state space (state here is a synonym for label). We project the state space S into a

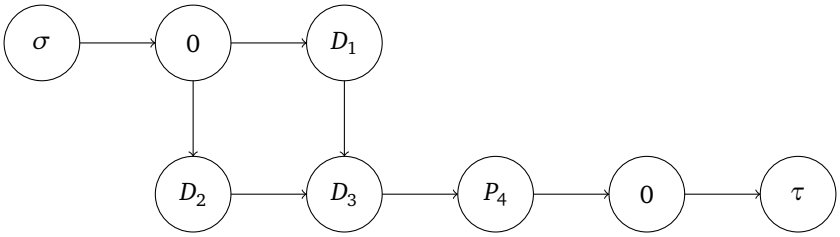


Figure 4.3: Example graph that shows the importance of correctly ordering the relaxed labels.

lower-dimensional space $S' = N \times \{0, \dots, Q_v\}^2$. The projection function sends an element $L = (v, \Pi, \Delta, \vec{V}) \in S$ to the element $L' := \text{proj}(L) = (v, \Pi, \Delta) \in S'$. As shown by Christofides et al. [14], the interesting property of state space relaxation is that the cost of the new label is $C(L') \leq \min_{L \in \text{proj}^{-1}(L')} C(L)$, and therefore provides a lower bound on the minimum cost of the corresponding label in the original space. The drawback is that S' contains labels that would be infeasible for the ESPPRC and, in particular, there are labels that correspond to paths that are not elementary. The monotonically decreasing nature of the resources, however, guarantees that eventual cycles will be finite. The state-space-relaxed labelling algorithm, therefore, solves the SPPRC. Since $|S'| = |N| \cdot (Q_v + 1)^2$, the algorithm has pseudo-polynomial complexity. The extension function of this algorithm will omit the missing component and the dominance criterion will omit condition (b). We will denote the problem solved by the state space relaxed version of our labelling algorithm as $(\text{SP'})_v$.

Label extension order It is important to observe that the fundamental assumption of label-setting algorithms (if a label L_1 dominates a label L_2, then all extensions of L_1 dominate the corresponding extensions of L_2) is valid for the relaxed state space, only if we process the labels in a particular order. To see why this is the case, consider the graph in Figure 4.3 where, for simplicity, we omitted the time component and all arcs have cost 1. Ports D_1, D_2, D_3 are delivery ports with demands, respectively, 5, 4 and 5. Port P_4 is a pickup port with demand 10. The vessel's capacity is 10. Consider two paths: $r_1 = (\sigma, 0, D_1, D_3, P_4, 0, \tau)$ and $r_2 = (\sigma, 0, D_2, D_3, P_4, 0, \tau)$. The labels associated respectively with these two paths at node D_3 are $L_1 = (D_3, 10, 0)$ and $L_2 = (D_3, 10, 1)$. Having both the same cost, we should conclude that L_2 dominates L_1. However, if we consider their extensions to port P_4, we see that both labels would be extended into $L_1' = L_2' = (P_4, 0, 0)$. Therefore it is not true that any extension of L_2 dominates the corresponding extension of L_1. This observation also applies to the state relaxation used in Dell'Amico et al. [15], where the authors projected component $\vec{V}$ into $\sum_{i \in P} \vec{V}_i$, as both partial paths in D_3 have visited the same number of ports. To overcome this limitation, we equip S with a suitable total order relation, and make sure that a label is not extended before all its predecessors are. The order we use in our algorithm is a lexicographic comparison, starting with component Δ (lowest first), followed by Π (lowest first) and finally by the cost (highest first).

4.4.4 Acceleration techniques

Both labelling algorithms for the ESPPRC and the SPPRC can be accelerated heuristically, by pruning the time-expanded graph. We propse two strategies for arc removal. The first strategy involves sorting all arcs in each graph by their reduced cost, and only keep the $K_2 \cdot |A_v|$ ones with lowest reduced cost, where $K_2 \in (0,1)$ is a parameter. This sparsification method takes into account both the original cost of the arcs and the reduced prices to pay at the ports; we refer to this technique as (C+P).

The second strategy, instead, sorts the ports by their respective reduced prices π_i. Let $\bar{\pi}$ and $\underline{\pi}$ the highest and lowest, respetively, reduced prizes (the lowest one being the most desirable), and $\bar{i}$ and $\underline{i}$ the ports corresponding, respectively, to $\bar{\pi}$ and $\underline{\pi}$. Then each arc is removed with a probability directly proportional to the dual price of its target port, with arcs incoming to $\underline{i}$ having probability 0, and arcs incoming to $\bar{i}$ having probability K_3, where $K_3 \in (0,1]$ is a parameter. This sparsification method only takes into account the reduced prices, independently from the original cost of the arcs; we refer to this technique as (P).

4.5 Branch-and-price algorithm

As mentioned in Section 4.3, we solve the FNDP with a branch-and-price algorithm. We will explore the branch-and-bound tree according to a best-first lazy strategy: when a node is explored, the next node will be the one whose father's lower bound is the highest (with ties broken randomly).

4.5.1 Column generation

At each node of the tree we solve the linear relaxation of (RMP), which we call (RRMP), and use its dual values to find negative reduced cost columns, by solving k subproblems $(\text{SP'})_v$. We solve the subproblem using both the heuristics and the exact methods presented in Section 4.4. Heuristic pricing has been succesfully employed in order to speed up column generation in a variety of routing problems (see, e.g., Dumas et al. [18], Savelsbergh and Sol [34], and Desaulniers et al. [16]).

The following methods are used sequentially; if any of them finds a route of negative reduced cost, the column generation phase is halted and (RRMP) is re-solved.

1. The first heuristic is the randomised-greedy heuristic of Section 4.4.1, which is tried 100 times in each direction, with parameter $K_1 = 10$. This heuristic is very quick and has the advantage of producing feasible columns; however, it is usually able to produce negative-reduced-cost columns only at the beginning of the exploration of a branch-and-bound node.

2. Next, we solve the SPPRC by dynamic programming, employing the acceleration techniques presented in Section 4.4.4. We use (C+P) with parameter $K_2 = 0.25$ and (P) with parameter $K_3 = 1$.

3. Finally, we solve the SPPRC by dynamic programming, on the complete graph.

4.5.2 Column management

We solve the first iteration at the root node with a pool made of one single dummy column, corresponding to a visit to each port. We attribute a very high cost to this column and consider as infeasible any solution that includes it among the base columns. If unused vessels can be chartered out, and β_v is the revenue generated by chartering vessel $v \in V$, another possibility is to initialise the column pool with one column for each vessel, each with cost $-\beta_v$ and not visiting any port.

At each node we remove duplicate columns from the column pool. Notice that neither the order in which ports are visited, nor the visit times, nor the visits to the hub are encoded in the columns, so it is possible to have duplicate columns with different costs. In this case we only keep the column with the lowest cost.

4.5.3 Branching

In classic VRP-like problems, the integrality and feasibility of the solution is often imposed by considering a route corresponding to a fractional column, and a customer (say i) visited by that route, which is also covered by another fractional column, such that the the two customers (say j, k) preceding i in the two routes differ. Branching is performed by imposing in one child node that arc (j, i) is not used (e.g. by removing that arc from the graph), and in the other child that node i can only be reached via j (e.g. by removing all other arcs leading to i and all other arcs leaving j). In this latter node, because of the set covering constraints, nodes i and j need to be visited, and therefore the use of arc (j, i) is guaranteed.

Since in our model (4.2)–(4.4) we do not have such set covering constraints, this branching rule would not actually guarantee an integer solution in the second node. In our problem, in fact, more decisions need to be made: whether a port is visted or not; if so, by which vessel; once the vessel is fixed, by using which arc.

For this reason, we propose the branching rules described below. The application of these branching rule will also guarantee that unfeasible columns (i.e., those corresponding to routes that visit a port more than once) will not appear in the optimal solution. The order in which the following rules are applied is: (1) branch on port visit; (2) branch on vessel; (3) branch on successive visits; (4) branc on arc selection.

Branching on port visit

The first branching rule is used to determine whether or not a port is visited, by any vessel. This branching rule was also used by Boussier et al. [7] in their branch-and-price algorithm for the TOP.

Let, for a given solution $\hat{x}$ of (RRMP) and for a port $i \in P$, $x(i) = \sum_{r \in R'} \varepsilon_{ri} \hat{x}_r$ (R' is the set of routes active at the node). If not all values $x(i)$ are integer (i.e., either 0 or 1) then one port is visited with fractional flow. We then select the port i for which the quantity $x(i)$ is most fractional (i.e., closer to 0.5), and create two branches.

In the first branch i is visited and, in the master problem, constraint (4.3) is replaced by

$$\sum_{r \in R} \varepsilon_{ri} x_r = 1 \tag{4.8}$$

The subproblem remains unchanged, but notice that now the dual variable π_i associated with the port is unrestricted, meaning that a path visiting i will now potentially collect a prize $\pi_i \leq 0$ which will be deducted from the cost of the partial path.

In the second branch, i is not visited. In the master problem, we remove all columns corresponding to routes r for which $\varepsilon_{ri} = 1$. In the subproblem, we remove from all graphs the nodes of set N_i. Therefore, no column covering i will appear in the subtree of this branch.

Branching on vessel selection

If all ports have an integer value for $x(i)$, it can still happen that there is a port i which is visited by more than one vessel. Define, for a solution $\hat{x}$ of (RRMP), a port $i \in P$, and a vessel $v \in V$, the quantity $x_v(i) = \sum_{r \in R'_v} \varepsilon_{ri} \hat{x}_r$ (R'_v is the set of routes associated with vessel v, and active at the node). We then select the port i and the vessel v for which the quantity $x_v(i)$ is most fractional, and create two branches.

In the first branch, we want i to be visited by v (if it is visited at all). In the master problem, we remove all columns of $r \in R'_w$ with $\varepsilon_{ri} = 1$, for all vessels $w \neq v$. In the subproblem, we remove all nodes of N_i from all graphs associated with vessels $w \neq v$. Notice that if the inequality (4.3) corresponding to i was already transformed into an equality by the previous branching rule, than we can also remove all columns of $r \in R'_v$ with $\varepsilon_{ri} = 0$, as in this case we know that i will be visited, and will be visited by v.

In the second branch, i is not visited by v. Therefore, in the master problem, we remove all columns of R'_v which cover i. In the subproblem, we remove all nodes of N_i from the graph associated with v.

Branching on arc selection

Consider a solution which does not trigger any of the preceding branching rules, i.e. each port has integer flow, and is visited by only one vessel. Let $r \in R'$ be the route corresponding to the most fractional column in the base, and v the associated vessel.

A branching rule sufficient to ensure that the final solution be integer and without infeasible columns would consist in considering any arc a of r connecting two ports, say linking (i, t) to (j, t') in the time-expanded graph, and creating two branches.

In the first branch, we force v to use arc a if travelling from i to j. To do so, we remove from the master problem all columns corresponding to routes $r \in R'_v$ such that $\varepsilon_{ri} = \varepsilon_{rj} = 1$, but which do not use a. In the subproblem we remove from G_v all other arcs linking nodes of N_i with nodes of N_j.

In the second branch, we forbid the use of the arc; in the master problem we remove the routes of R'_v which use a, and in the subproblem we simply remove the arc from the graph G_v.

Notice that this rule can also exclude routes containing cycles, as in these routes there is at least one port which is visited twice, i.e. by using two incoming arcs, and one of the arcs can be forbidden by using this rule. In this case, the route need be applied to integer columns as well.

However, the rule produces two unbalanced subtrees, as one branch imposes a much stronger condition than the other. Furthermore, routes where v does not visit neither i nor j can appear in both branches. In order to improve the convergence of the algorithm, then, we also devise another branching rule, described in the next subsection.

Notice, finally, that this branching rule used in conjunction with the previous two forms a complete branching scheme. Indeed, in the branch forbidding the use of the arc, its flow will be obviously integer (in particular, it will be 0). When imposing the use of the arc, however, the optimal solution to (RRMP) could still give fractional flow. In this case, however, if the total flow on port j is fractional, we will perform branching on the port visit. In the branch that excludes j the flow on the arc will be again obviously integer (and equal to 0). In the other branch, we still have the possibility that the flow be fractional. But in this case, since the total flow on j is 1, there must be another route associated with a different vessel whose corresponding column is also in the base. In this case we will perform branching on vessel selection. In the branch that assigns j to v the flow on the arc is forced to be 1; in the other branch, it is forced to be 0.

Branching on successive visits

Let $r \in R'_v$ be the route corresponding to the most fractional column in the base. If there is another fractional column corresponding to a route $r' \in R'_v$ ($r' \neq r$), such that there is a port i visited by both r and r', and the ports preceding i in r and r' are different (say, j and j'), then we can perform binary branching by respectively forcing and forbidding the consecutive visit of ports j, i. This rule can again be applied to routes corresponding to integer columns, in order to remove cycles.

In the first branch, we remove from the master problem all columns of R'_v whose corresponding route does not visit the succession of ports j, i; in the subproblem, we can remove from G_v all arcs from nodes of N_j to nodes of N_k (for $k \neq i$), and from nodes of N_k to nodes of N_i (for $k \neq j$). In the second branch, we remove from the master problem all columns of R'_v whose corresponding route visits j, i in succession; in the subproblem we remove from G_v all arcs from nodes of N_j to nodes of N_i.

4.5.4 Upper bounding

In order to strengthen the upper bound, at the end of the exploration of the root node, problem (RMP) is solved as an integer problem with the feasible columns currently in the column pool, using a black-box commercial solver. This step is performed only if the column pool has fewer than 1500 columns.

4.6 Results

In this section we describe how the test instances were generated starting from instances already present in the literature, and we provide computational results that highlight both the performance of our algorithm and key features of optimal routes.

4.6.1 Instance generation

The instances used in this paper are derived from the library LinerLib, presented by Brouer et al. [8]. We considered the two LinerLib scenarios *Baltic* and *Western Africa* (*WAF*). Both scenarios include a single hub: Bremerhaven for the Baltic scenario and Algeciras for the Western African one. In the Baltic scenario, we have a 1-week planning horizon, modelled with a discretisation of 2 hours per time interval; in the WAF scenario, the planning horizon is of 4 weeks, and each time instant represents 8 hours. The Baltic scenario comprises 13 ports (see Figure 4.4a) and 6 vessels, while the Western African one has 20 ports (see Figure 4.4b) and 10 vessels. In both scenarios the bunker price is of \$375/tn, and all α_i have been set to 0, thereby considering the pure opportunity cost of each service (revenue minus cost of performing the service). Furthermore, each vessel can sail at a low, medium, or high speed (the minimum and maximum speeds are obtained from the LinerLib data).

We generated respectively 12 and 8 base instances for the Baltic and WAF scenarios. These instances share the same network topology of the LinerLib original instances, but have different values for time windows, transit times, and handling times:

- We considered instances both with and without closing time windows. When they are present, their centres are distributed evenly along the time horizon, to simulate ports closing at night, each day of the week. For the Baltic scenario, we have three options: no time window; all time windows have duration of 1 time interval; time windows have durations of either 1 or 3 time intervals (the actual value is chosen at random for each port). For the WAF scenario, we only have two options: no time window, or all time windows with duration 1.

- We also generated instances with and without maximum transit times. When the transit times were enabled, their value was comprised between 5 and 6.5 days for the Baltic scenario, and between 23 and 27 days for the WAF scenario. The actual transit time for each port was chosen according to a uniform random distribution over said intervals.

- The handling times were also distributed in intervals ($[2,4]$ or $[3,5]$ time units for the Baltic scenario; $[1,1]$ or $[1,2]$ time units for the WAF scenario). However, the actual value for each port has been chosen proportionally to the number of containers to be handled at the port.

We thus generated a total of 20 base instances, available at Santini [32], which we use in Section 4.6.2 to verify that the proposed branch-and-price algorithm attains state-of-the-art performances in terms of solution time and quality vs number of ports and vessels considered. We then generated further instances, which were used for the scenario analysis presented in

Section 4.6.3. The objective of the scenario analysis is to assess the impact of external conditions, designer's decisions, and modelling precision on the optimal solutions. These instances were based on the Baltic scenario. Typical questions that are investigated in the scenario analysis are, e.g.: under which conditions it is convenient to buy capacity on competitors? what happens when there is a surge in bunker prices? will considering more possible speeds in the model lead to significantly better solutions? will longer routes lead to better utilisation of ship capacities?

In order to perform this analysis, we varied further characteristics of the base instances:

Bunker price While the bunker price was fixed to an average value of \$375/tn in the base instances, in this scenario analysis we use the values $\{250, 300, 375, 450, 500\}$ to assess what changes in the solution under low, medium, and high bunker prices. This price is in practice subject to changes for two main reasons: the volatility of crude oil price, and the possibility that certain countries will pass legislation forcing the use of low-sulfur bunkers, which are less polluting but much more expensive.

Speeds In this analysis we use 1, 3 (base instances), and 5 possible speed values for each vessel, in order to check whether the increased complexity (especially in terms of arcs being created in the graphs G_v) given by adding more speed values leads to significantly better solutions.

Demand In this analysis we consider scenarios with demand multiplied by a factor of 1.0 (base instances), 0.8, 0.6, and 0.4. This analysis is particularly important, as one of the major consequences of a global financial and consumption crisis is a steep decline in the volumes of goods traded and shipped by sea.

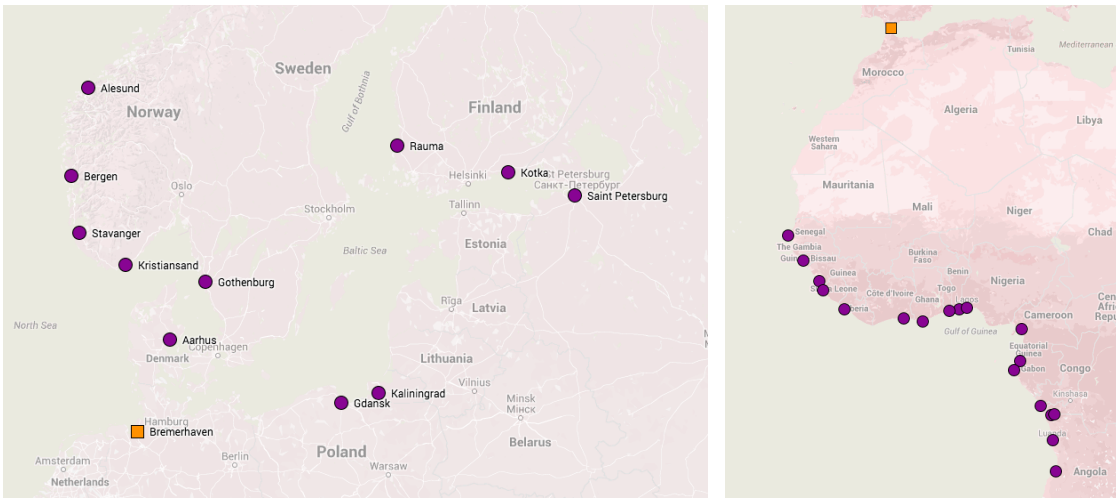
Penalties We use penalties to model the purchase of capacity on a competitor. In particular, while in the base instances we only had opportunity costs, in this scenario analysis we can outsource a service. In this case, we can still earn a small percentage of the revenue (1%, 5%, or 10%), while most of it is transferred to the competitor.

Time horizon We solved the Baltic instances with time horizons of 1 (base instances), 2, and 3 weeks. The number of hours per time interval were, respectively, 2, 4, and 6. The number of ships available is scaled according to the time horizon (1/2 and 1/3 of the original fleet size, respectively) to reflect the fact that multiple ships need to sail the same rotation at once.

In total, we generated additional 48 instances for the bunker price analysis, 24 for the speeds analysis, 36 for the demand analysis, 36 for the penalties analysis, and 24 for the time horizon analysis.

4.6.2 Computational results

Computational experiments were run on a dual-core 3.10GHz Xeon CPU with 4GB of RAM. The LPs were solved using CPLEX 12.6, limited to the use of one thread. Table 4.2 reports the results for the Baltic and Western African scenarios.



(a) The Baltic scenario. (b) The WAF scenario.

Figure 4.4: Ports in the considered scenarios. Map data: Google.

Instance	Time	SP Time %	Gap %	RGap %	Cols	NNodes	Depth
Baltic1	0.77	96.75	0.00	0.00	72	1	1
Baltic2	4.07	99.54	0.00	0.00	91	1	1
Baltic3	13.87	98.67	0.00	0.00	145	1	1
Baltic4	9.39	98.65	0.00	0.00	113	1	1
Baltic5	8.66	99.16	0.00	0.00	114	1	1
Baltic6	4.99	99.60	0.00	0.00	88	1	1
Baltic7	11.67	99.08	0.00	0.00	135	1	1
Baltic8	1.71	98.80	0.00	0.00	64	1	1
Baltic9	1.65	98.18	0.00	0.00	70	1	1
Baltic10	2.21	96.39	0.00	0.00	78	1	1
Baltic11	4.08	98.25	0.00	0.00	95	1	1
Baltic12	2.44	98.62	0.00	0.00	75	1	1
WAF1	3600.00	91.66	0.04	0.34	3402	321	151
WAF2	819.04	93.40	0.00	0.34	5132	45	13
WAF3	1382.66	93.93	0.00	0.71	5036	89	16
WAF4	1229.08	94.98	0.00	0.34	5396	47	12
WAF5	555.15	92.61	0.00	0.21	4255	25	9
WAF6	319.76	93.38	0.00	0.21	3474	17	8
WAF7	275.23	87.44	0.00	0.21	3101	11	5
WAF8	497.11	92.99	0.00	0.21	4754	35	13

Table 4.2: Computational results for the base instances.

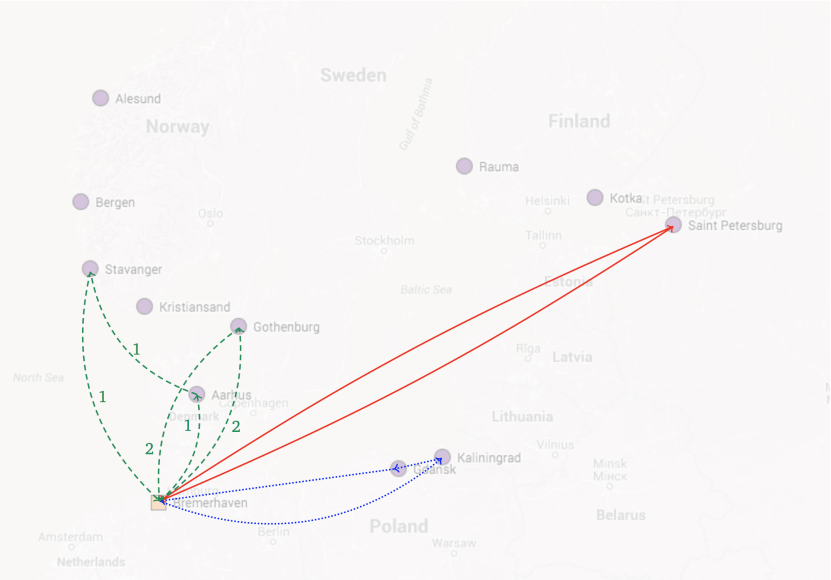


Figure 4.5: Optimal solution for instance Baltic1.

Column “Time” reports the execution time in seconds, while column “SPTime” tells the time, in percentage, spent solving the subproblem. Columns “RGap” and “Gap” report the optimality gap percentage, respectively at the root node and at the end of the solution process. The gap is calculated as $100(\text{UB} - \text{LB})/\text{UB}$. Finally, Column “Cols” indicates the size of the column pool, “NNodes” gives the number of branch-and-bound nodes explored, and “Depth” is the maximum depth reached in the branch-and-bound tree.

Table 4.2 shows that the proposed approach is able to solve to optimality (or almost to optimality) instances of realistic size. Our results are in line or better with recent work in maritime optimisation: Reinhardt and Pisinger [31], e.g., solve instances with up to 15 ports; Plum et al. [27] solve instances with up to 25 ports, but only consider one vessel.

The root node gaps show that the linear relaxation of (MP) is very strong. In particular, comparing with an earlier version of this paper, we noticed that relaxing the requirement that each route visits the hub exactly once provides both a stronger dual bound at the root node, and quicker convergence towards optimality. This comes at no cost in terms of solution quality, as the present version of the problem also produces routes which are closer to the real-life requirements of network planners.

All Baltic instances are solved at the root node, whereas branching is required for the WAF instances. Most of these instances are solved to optimality exploring fewer than 100 nodes; the only instance that requires exploring a large number of nodes and going deep in the branch-and-bound tree, is also the only open instance, WAF1. An analysis of the logs, furthermore, has shown that the branching rules described in Sections 4.5.3 to 4.5.3 were sufficient to solve all closed instances to optimality, while the branching rule described in Section 4.5.3 was never used.

Figures 4.5 and 4.6 show two optimal solutions, respectively to instances Baltic1 and WAF5.

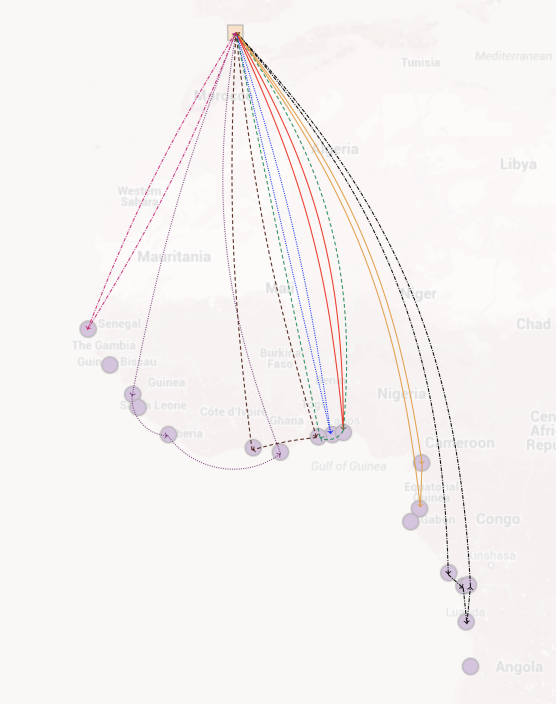


Figure 4.6: Optimal solution for instance WAF4.

In Figure 4.5, for example, a butterfly route is used to serve first Aarhus and Stavanger and, after returning to the hub, Gothenburg. A dedicated service is performed in Saint Petersburg, which has high demand; on the ohter hand, the port of Rauma (Finland) is distant and has very low demand, so its service would have a negative reduced cost, and the service is skipped. Some port might be unprofitable to serve, but an operator might still decide to serve it for prestige reasons or other commercial considerations. In this case, the network planner could direct the model in this sense, by applying an appropriate penalty α_i. This could happen, e.g., if the operator decided to serve more than just one port in Norway, by mantaing an additional presence in Bergen, Alesund, or Kristiansand.

By analysing these solutions, we noticed two recurring features. First of all, there are ports which require dedicated services because of their high volume. Second, routes tend to serve ports clustered together, as is clear in Figure 4.6. These observations are validated by real-life practice where indeed some port is served by dedicated vessels and services are often separated into short-sea clusters and deep-sea clusters, as this configuration gives ports called on each type of service a relatively good transit time for both import and export volumes. Notice, e.g., the short-sea cluster Conarky – Monrovia – Takoradi served by the dotted purple route in Figure 4.6 and the deep-sea cluster Pointe-Noire – Boma – Lobito – Matadi served by the black dash-dotted line.

An analysis of the solutions of the base instances has shown an average vessel speed of around 15kn in the Baltic scenario, and 13.5kn in the Western African one, and that the cost of bunker accounts for around 30% and 20% of the total costs, repsectively. These values are

Value	Rev	SU	Srv	HLE	CTD	Spd	BC
250	+5.80	0.00	+5.30	-2.02	-0.58	0.69	-28.36
300	+3.36	0.00	+5.30	-2.02	-0.58	0.63	-16.98
375	0.00	0.00	0.00	0.00	0.00	0.00	0.00
450	-3.92	0.00	0.00	0.00	0.24	-0.03	11.72
500	-6.19	0.00	0.00	0.00	0.24	-1.13	23.28

Table 4.3: Scenario analysis for the "Bunker Price" value.

Value	Rev	SU	Srv	HLE	CTD	Spd	BC
1	-24.37	-20.83	-9.09	-43.51	-17.63	-19.94	-38.51
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	+0.36	0.00	+4.55	0.00	-0.81	0.21	-0.37

Table 4.4: Scenario analysis for the "Number of speeds" value.

Value	Rev	SU	Srv	HLE	CTD	Spd	BC
0.4	-78.75	-62.50	-36.36	-44.85	-22.44	-11.08	-25.26
0.6	-55.12	-25.00	4.55	-56.42	-6.99	-8.17	-14.89
0.8	-8.52	0.00	13.64	-16.22	8.26	0.31	-3.25
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.5: Scenario analysis for the "Demand" value.

also in line with real-life data.

4.6.3 Scenario Analysis

Tables 4.3 to 4.7 present the summary of the scenario analyses. Column "Value" reports the variations on the relevant value that is being changed (bunker price, number of speeds, demands, share of revenue kept when outsourcing, time horizon length). Each row aggregates, by taking the average, all instances which share the same relevant value. The underlined value is the one used for the base instances. All other columns report the percentage variation of some metric compared to its base value. If a metric has value m_v for one group of instances and value m_b for the base instances, the column will show the value $100(\frac{m_v}{m_b}-1)$. The metrics considered are:

Rev The total revenue earned.

SU The total number of ships used. This value is adjusted appropriately when instances with different time horizons are considered.

Srv The total number of services performed.

HLE The highest load efficiency. For a given instance this value is the average of the highest load efficiency for each rotation. The highest load efficiency for a rotation is the highest quantity D/Q_v achieved at some point in the rotation, where D is the total quantity of cargo on board, and Q_v is the capacity of the considered vessel.

Value	Rev	SU	Srv	HLE	CTD	Spd	BC
Keep 10%	+9.24	0.00	+1.52	-4.04	+0.48	-2.25	-4.88
Keep 5%	+4.60	0.00	+0.76	-2.02	+0.24	-1.13	-2.44
Keep 1%	+0.98	0.00	0.00	0.00	0.00	-0.04	0.00
Keep 0%	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.6: Scenario analysis for the "Penalties" value.

Value	Rev	SU	Srv	HLE	CTD	Spd	BC
1wk	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2wk	+18.70	+100.00	+27.27	+4.10	+42.73	-27.39	-18.90
3wk	+1.19	+100.00	+20.00	+4.58	+80.85	-23.10	-11.60

Table 4.7: Scenario analysis for the "Time horizon" value.

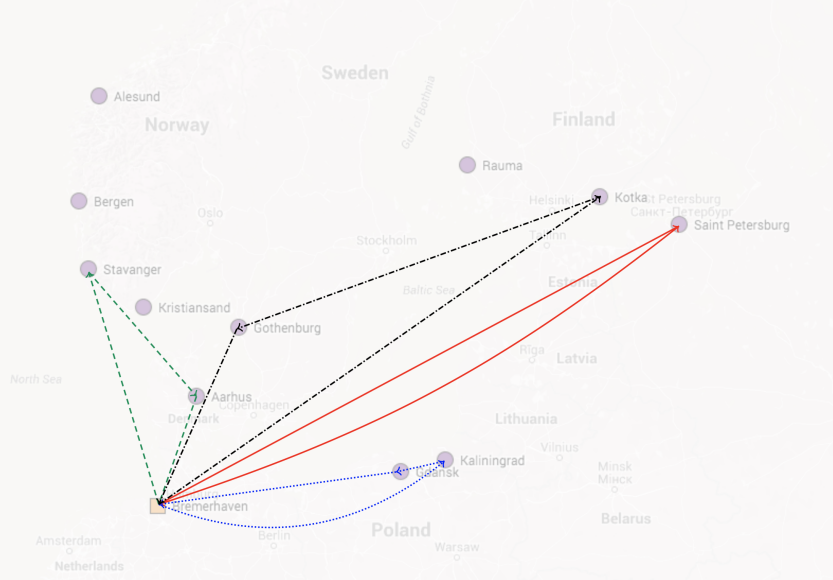


Figure 4.7: Optimal solution for the Baltic1 instance with demand coefficient of 0.8.

CTD The cargo travel distance. For a given instance this value is the average amount of nautical miles that each cargo has spent travelling on-board the ship.

Spd The average speed, considering in each instance all legs sailed by all vessels employed.

BC The percentage of the total costs incurred which is attributable to bunker costs.

All the instances in the scenario analysis were solved to optimality, but for those with 2- and 3-weeks time horizons, for which the scenarios refer to the best integer solution obtained within 1 hour. All optimality gaps were under 1% for the 2-weeks time horizon, and under 10% for the 3-weeks time horizon.

Table 4.3 shows the sensitivity of the solutions to variations in bunker price. As it can be expected, the share of costs attributable to bunker price is the most affected KPI (Key Performance Indicator). Revenue also benefit greatly from lower bunker costs, by exhibiting a wide gap between the lowest and highest values. It is interesting to notice that lower bunker prices allow for slightly faster speeds; this, in turn, could mean being able to serve more ports, thereby further increasing revenue and coverage. This can be seen for the $250 and $300 prices, where a +5.30% in the number of services means that one more service was performed.

Notice that the insertion of one port can alter the overall shape of the rotations considerably. Therefore, it can be difficult to add one additional service at a later time, after the network design has been decided, just because of a decrease in bunker prices which now makes that service profitable. For this reason, we advise network planners to first consider network layouts with more services, and to try increase the profitability of the routes by means of fuel hedging, contracting lower port fees, etc.

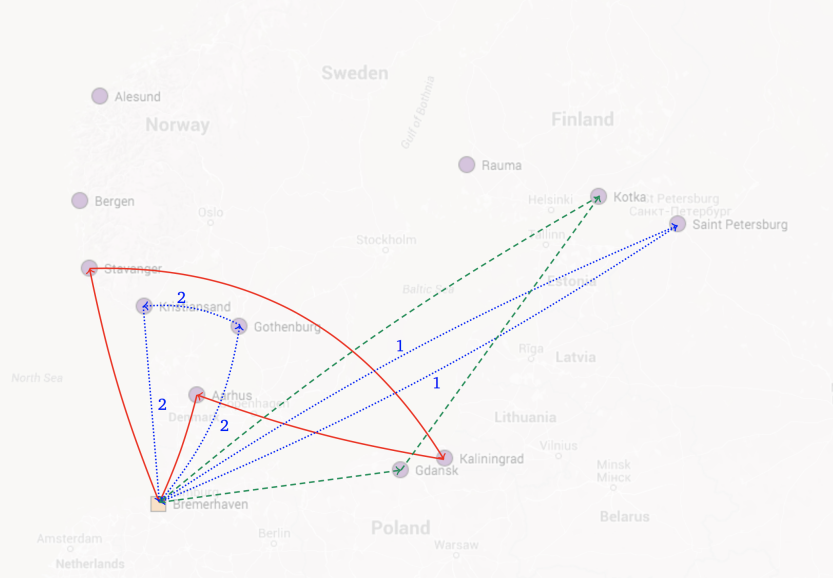


Figure 4.8: Optimal solution for instance Baltic1 with a 2-week time horizon.

Table 4.4 analyses the impact of speed optimisation in the network plan. Notice how considering three rather than just one fixed speed has a great impact on all the KPIs considered. Not including a speed decision means obtaining solutions with −24% revenue and fewer served ports. Notice that the average speed is lower when only considering 1 speed (in this case the speed considered was the average between the maximum and minimum speeds provided by the LinerLib for each vessel) showing that the advantages of speed optimisation cannot be simply attributed to low-speed steaming, but rather to a smart combination of low-speed and high-speed legs which, on average, brings the speed up and allows a vessel to serve more ports in a single route. In some instances, using only one speed meant the deployment of one fewer vessel, as a route became either unfeasible or unprofitable.

On the other hand, using five speed values has a positive effect on the revenue and the number of services performed. The difference is not as big as the one noted before (e.g. in terms of revenue generated) and the route shapes are very similar between the solutions with 3 and 5 speeds. Finally, the average speed is also very close between these two group of instances. In short, we see no downsides in using a higher number of speeds, as all instances could still be solved to optimality, but the planner should not expect to see big improvements just by increasing the granularity of speed discretisation beyond a certain point.

Other interesting observations can be made by considering Table 4.5, which shows the impact of demand. First of all notice that, as expected, less demand means less revenue for the oprerator. However, notice how revenue multiplier 0.8, corresponding to a reduction in demand of −20%, gives a reduction in revenue of just −8.52%. This is because, by using the same number of ships (column “SU”) the optimal route increases coverage (column “Srv”) as an effective countermeasure. A further reduction in demand, with multipliers 0.6 and 0.4 is, instead, eccessive and cannot be counteracted as effectively.

The increased coverage seen for demand multiplier 0.8 is due to the fact that routes that before were infeasible because of capacity restrictions, now become feasible. Compare, for example, the optimal solutions to instance Baltic1 with multipliers 1.0 (Figure 4.5) and 0.8 (Figure 4.7); in the second network plan, a vessel serves the port of Kotka which was previously unserved. The ports of Gothenburg and Kotka are served by a vessel with capacity 800TEU; the sum of their pickup demands is 822TEU, but it reduces to just 658TEU when the multiplier is 0.8, thereby making the route feasible.

In summary, we can notice that a decrease in the demand corresponds to a roughly proportional decrease in the total revenue, but the planner can respond with better fleet utilisation and wider coverage. This shows that restructuring the routes can be an effective countermeasure during extended periods of low demand.

In Table 4.6 we report the results of allowing to outsource some service. It can be noted that admitting this possibility only resulted in small changes in the generated routes. The major variation is recorded in the generated revenue, as ports that are not served earn nothing in the base case, while they earn a small fraction of their revenue in the other cases.

It is interesting to notice that, when keeping 10% of the revenue, the total number of services performed actually increased: the port of Saint Petersburg, which was served with a dedicated vessel, was instead outsourced and the vessel was used to increase coverage of other ports. This hints to the fact that the intuitive rule of thumb of focussing on high-demand (and, therefore, high-revenue) ports in the strategic phase, and delegating the decision of buying capacity on competitors to the tactical or operational stages, can lead to sub-optimal results.

Finally, Table 4.7 shows what happens when we allow longer routes. In the base case, the routes can last up to one week and there are 6 vessels available; in the 2-week case, we have 3 vessels; in the 3-week case, we only have 2 available vessels. It appears that the optimal route duration from the point of view of revenue should be of 2 weeks, and that further increasing the time horizon to three weeks actually gives worse solutions, even though still slightly better than the 1-week base case.

While in the 1-week scenario we were using on average 3 vessels (out of 6 available), in the 2-week case we produce 3 rotations (thereby deploying all 6 vessels), and in the 3-week case we produce 2 rotations (again deploying all 6 vessels). The number of services performed increases, as do the highest load efficiency and the cargo travel distance. At the same time, longer routes allow for lower speeds and, therefore, a lower share of costs attributable to the bunker. We can compare the optimal solutions of instance Baltic1 for a 1-week (Figure 4.5) and for a 2-week time horizon (Figure 4.8). Notice how the vessel that is serving Saint Petersburg can now be reused in the second part of the route, after being unloaded at the hub, and proceeds to serving Gothenburg and Christiansand. At the same time, the other two vessels also perform longer routes.

This is probably the most impactful design decision, as the variation in earned revenue gets up to +18.70%. Considering that the time horizon length is mostly an operator's decision which does not depend on external factors, this can surely be the most critical decision in the design of a feeder network.

4.7 Conclusions

In this paper we proposed an exact algorithm for the solution of the Feeder Network Design Problem. The algorithm can handle instances of realistic size and either solves them to optimality, or finds a solution close to the computed lower bound. The modelling framework is able to describe many real-world constraints and, as such, has been used to perform scenario analysis with the objective to derive general guidelines for network planners.

In particular, we assessed: (1) The impact of bunker price on the profitability of the services; we advise the planners to prioritise wider service coverage and fuel negotiations options. (2) The importance of leg-by-leg speed optimisation; to this end, while slow steaming is a consolidated practice for inter-continental services, we show that a combination of slower and faster sailing speeds is more apt for feeder networks. (3) The effect of demand fluctuations; we have showed that demand is a crucial factor in determining profitability, but the detrimental effects of a prolonged period of low demend can be reduced if the planner responds with a suitable network restructuring. (4) Outsourcing services by buying capacity on competitors can have deep effects on the network design; capacity availability on competitors, however, can be volatile and therefore we advise prudence when trying to incorporate this decision at the strategic level. (5) Designing longer rotations and deploying more vessels to each of them, can have a strong positive impact on profitability; however, the relationship is not linear, and the planner must perform an accurate analysis to determine the optimal rotation length.

As for future research avenues, we would like to retrieve realistic data for scenarios with more ports, in order to better asses which are the largest instances that the algorithm can solve to optimality. Furthermore, we would like to test the validity of the proposed approach for similar problems, such as the TOP with pickup and delivery for which, at the best of our knowledge, only one heuristic algorithm has been proposed [5].

Acknowledgments

We are grateful to the support from Optimization Manager Mikkel M. Sigurd from Mærsk Line, for fruitful discussions which made this work possible. Stefan Røpke was supported by The Danish Strategical Research Council and The Danish Energy Technology Development and Demonstration Program (EUDP) under the ENERPLAN and GREENSHIP project. We are also grateful to the anonymous referees for their comments, which greatly contributed to the improvement of the present work.

Bibliography

[1] Richa Agarwal and Özlem Ergun. Ship scheduling and network design for cargo routing in liner shipping. *Transportation Science*, 42(2):175–196, 2008.

[2] Martin Andersen. *Service Network Design and Management in Liner Container Shipping Applications*. PhD thesis, Danish Technical University, 2010.

[3] Claudia Archetti, Dominique Feillet, Alain Hertz, and Maria Grazia Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6):831–842, 2009.

[4] Claudia Archetti, M Grazia Speranza, and Daniele Vigo. Vehicle routing problems with profits. In Paolo Toth and Daniele Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[5] DG Baklagis, G Dikas, and I Minis. The team orienteering pick-up and delivery problem with time windows and its applications in fleet sizing. *RAIRO-Operations Research*, 50 (3):503–517, 2016.

[6] Maria Battarra, Artur Pessoa, Anand Subramanian, and Eduardo Uchoa. Exact algorithms for the traveling salesman problem with draft limits. *European Journal of Operational Research*, 235(1):115–128, 2014.

[7] Sylvain Boussier, Dominique Feillet, and Michel Gendreau. An exact algorithm for team orienteering problems. *4OR: A Quarterly Journal of Operations Research*, 5(3):211–230, 2007.

[8] Berit Brouer, Fernando Alvarez, Christian Plum, David Pisinger, and Mikkel Sigurd. A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2):281–312, 2013.

[9] Pierre Cariou. Is slow steaming a sustainable means of reducing CO2 emissions from container shipping? *Transportation Research Part D: Transport and Environment*, 16(3): 260–264, 2011.

[10] Ching-Chih Chang and Chih-Min Wang. Evaluating the effects of speed reduce for shipping costs and CO2 emission. *Transportation Research Part D: Transport and Environment*, 31:110–115, 2014.

[11] Marielle Christiansen, Kjetil Fagerholt, and David Ronen. Ship routing and scheduling: Status and perspectives. *Transportation science*, 38(1):1–18, 2004.

[12] Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Maritime transportation. *Transportation*, 14:189–284, 2006.

[13] Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228 (3):467–483, 2013.

[14] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164, 1981.

[15] Mauro Dell'Amico, Giovanni Righini, and Matteo Salani. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. *Transportation Science*, 40(2):235–247, 2006.

[16] Guy Desaulniers, François Lessard, and Ahmed Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008.

[17] Maritime Research Drewry. Seaborne trade annual report 2013. Technical report, Drewry, 2014.

[18] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[19] Jørgem Glomvik Rakke, Marielle Christiansen, Kjetil Fagerholt, and Gilbert Laporte. The traveling salesman problem with draft limits. *Computers & Operations Research*, 39 (9):2161–2167, 2012.

[20] Stefan Irnich, Guy Desaulniers, et al. Shortest path problems with resource constraints. *Column generation*, 6730:33–65, 2005.

[21] Christos Kontovas. The green ship routing and scheduling problem (gsrsp): A conceptual approach. *Transportation Research Part D: Transport and Environment*, 31:61–69, 2014.

[22] Marine Intelligence Unit Lloyd's. Measuring global seaborne trade. Technical report, Lloyd's, 2009.

[23] Berit Løfstedt, Fernando Alvarez, Christian Plum, David Pisinger, and Mikkel Sigurd. An integer programming model and benchmark suite for liner shipping network design. Technical Report 19, DTU, Technical University of Denmark, 2010.

[24] Enrico Malaguti, Silvano Martello, and Alberto Santini. The traveling salesman problem with pickups, deliveries, and draft limits. *Omega*, In press, 2017.

[25] Qiang Meng, Shuaian Wang, Henrik Andersson, and Kristian Thun. Containership routing and scheduling in liner shipping: overview and future research directions. *Transportation Science*, 48(2):265–280, 2013.

[26] Judith Mulder and Rommert Dekker. Methods for strategic liner shipping network design. *European Journal of Operational Research*, 235(2):367–377, 2014.

[27] Christian Plum, David Pisinger, Juan-José Salazar-González, and Mikkel Sigurd. Single liner shipping service design. *Computers & Operations Research*, 45:1–6, 2014.

[28] Christian Plum, David Pisinger, and Mikkel Sigurd. A service flow model for the liner shipping network design problem. *European Journal of Operational Research*, 235(2): 378–386, 2014.

[29] Harilaos Psaraftis and Christos Kontovas. Speed models for energy-efficient maritime transportation: A taxonomy and survey. *Transportation Research Part C: Emerging Technologies*, 26:331–351, 2013.

[30] Harilaos Psaraftis and Christos Kontovas. Ship speed optimization: Concepts, models and combined speed-routing scenarios. *Transportation Research Part C: Emerging Technologies*, 44:52–69, 2014.

[31] Line Blander Reinhardt and David Pisinger. A branch and cut algorithm for the container shipping network design problem. *Flexible Services and Manufacturing Journal*, 24(3): 349–374, 2012.

[32] Alberto Santini. Maritime-vrp: v1.1, May 2016. URL http://dx.doi.org/10.5281/zenodo.51312.

[33] Alberto Santini, Stefan Ropke, and Christian E.M. Plum. A branch-and-price approach to the Feeder Network Design Problem. *European Journal of Operational Research (under revision)*, pages 1–16, 2017.

[34] Martin Savelsbergh and Marc Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[35] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*, volume 18. Siam, 2014.

[36] Shuaian Wang and Qiang Meng. Sailing speed optimization for container ships in a liner shipping network. *Transportation Research Part E: Logistics and Transportation Review*, 48(3):701–714, 2012.

[37] Shuaian Wang and Qiang Meng. Liner shipping network design with deadlines. *Computers & Operations Research*, 41:140–149, 2014.

[38] The World Shipping Council. The liner shipping industry and carbon emission policies. Technical report, The World Shipping Council, 2009.

5 Maritime seaside logistics: the travelling salesman problem with pickup, delivery, and draft limits

Abstract We introduce a new generalization of the traveling salesman problem with pickup and delivery, that stems from applications in maritime logistics, in which each node represents a port and has a known draft limit. Each customer has a demand, characterized by a weight, and pickups and deliveries are performed by a single ship of given weight capacity. The ship is able to visit a port only if the amount of cargo it carries is compatible with the draft limit of the port. We present an integer linear programming formulation and we show how classical valid inequalities from the literature can be adapted to the considered problem. We introduce heuristic procedures and a branch-and-cut exact algorithm. We examine, through extensive computational experiments, the impact of the various cuts and the performance of the proposed algorithms.

5.1 Introduction

One of the most well known variants of the (asymmetric) *Traveling Salesman Problem* (TSP) is the *TSP with Pickup and Delivery* (TSPPD). The problem is defined on a directed graph $G = (N, A)$ with node set $N = \{0, 1, \ldots, n, n+1, \ldots 2n, 2n+1\}$ and arc set $A = \{(i, j) : i, j \in N\}$. Node 0 is the starting depot and node $2n+1$ is the ending depot (that can eventually coincide). Each arc $(i, j) \in A$ has a cost $c_{ij} \geq 0$, and we assume that the *triangle inequality* ($c_{ij} \leq c_{ik} + c_{kj}$ $\forall i, j, k \in N$) holds. One has to serve n customers, each of which is associated with a *pickup* node i and a *delivery* node j. We assume, without loss of generality that, for any customer i, the pickup node i is in $\{1, \ldots, n\}$, and the corresponding delivery node j coincides with $n+i$. The objective is to find a Hamiltonian path of minimum total cost that starts at node 0 and terminates at node $2n+1$, in which the pickup node of every customer is visited before the corresponding delivery node. Although a customer may be origin or

This chapter is based on the contents of: Enrico Malaguti, Silvano Martello, and Alberto Santini. The Travelling Salesman Problem with pickups, deliveries, and draft limits. *Omega (to appear)*, pages 1–17, 2017. doi: 10.1016/j.omega.2017.01.005.

destination of a number of different requests, we always associate two distinct nodes to each request.

In the *capacitated* TSPPD (sometimes referred to in the literature as *the* TSPPD),

(i) each customer has a *demand* d_i, defined by a positive value (*weight*) associated with his pickup node i. We conventionally associate $d_{n+i} = -d_i$ with the corresponding delivery node. (For the depot, we assume $d_0 = d_{2n+1} = 0$.);

(ii) pickups and deliveries are performed by a single vehicle of capacity Q;

(iii) at no time during the tour the total load of the vehicle can exceed Q;

(iv) the vehicle leaves and returns to the depot empty.

In the present work we consider a generalization of the capacitated TSPPD that stems from maritime applications, in which nodes represent ports. Each node $i \in \{1, \dots, 2n\}$ has a draft limit $l_i > 0$. In maritime terminology the draft is the distance between the waterline and the bottom of the hull of a ship, and it varies as a function of the cargo onboard the ship. If the draft of a ship is greater than the draft limit of a port, the ship is not able to enter and operate safely at that port (see Figure 5.1). A ship could then deliver part of its cargo at other ports, until its draft is small enough to allow a visit to the port. The relationship between the amount of cargo onboard and the draft of a ship is given, and therefore the draft limit l_i can be expressed with the same unit as the demands d_i and the capacity Q. In other words, for the *Traveling Salesman Problem with Pickups, Deliveries and Draft Limits* (TSPPDD) it must also hold that

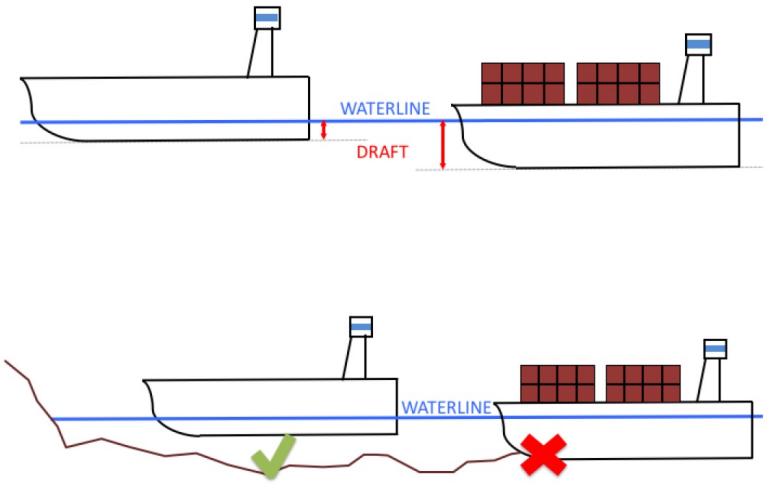


Figure 5.1: In the picture above, the draft of a ship as a function of the cargo on board. In the picture below, a ship able to enter a port (left) and one whose draft is too large to enter the same port (right).

(v) when traveling along arc (i, j), the total load of the ship cannot exceed $\min(l_i, l_j)$.

We assume in the following, without loss of generality, that demands, ship capacity, and drafts are positive integers.

The impact of drafts on maritime logistics is becoming more and more important, as the average size of the vessels is increasing. While draft was traditionally an issue related mostly with tankers and bulk vessels, it now involves container ships as well: the average size of a container ship has increased by 19% just in the four years between January 2007 and January 2011 (see Notteboom and Vernimmen [12]). Upgrading port infrastructure is, most of the time, too expensive and time consuming to be considered a feasible solution. Therefore, the burden of ensuring a proper balance between the economy of scale provided by the bigger vessels and the feasibility of the fleet composition and route planning, is left with the ship operator. As observed by Tirschwell [18],

> It's a lot easier for a carrier CEO to sign an order for a new ship than for a port to deepen its draft so that ships can enter or leave fully loaded. One takes 10 minutes, the other 10 years.

To the best of our knowledge, this is the first study on the TSPPDD, although the constraints we impose have been separately considered by other authors.

Dumitrescu, Ropke, Cordeau, and Laporte [6] studied the polytope of the TSPPD, derived facet-defining inequalities, and developed a branch-and-cut algorithm in which the inequalities are separated heuristically. They solved to optimality instances with up to 35 origin-destination pairs.

Ropke, Cordeau, and Laporte [16] and Ropke and Cordeau [15] studied the pickup and delivery problem with time windows, i.e., a multi-vehicle generalization of the TSPPD in which customers can only be visited within their opening time. The former paper presents a branch-and-cut algorithm, while the latter improves on it, by using a branch-and-cut-and-price approach. The traveling salesman problem with draft limits was introduced by Glomvik Rakke, Christiansen, Fagerholt, and Laporte [7]. In this problem, the ship starts from the depot completely loaded and the objective is to find the shortest Hamiltonian path to satisfy the demands of the customers without violating the drafts limits. They proposed two formulations, a branch-and-cut algorithm, and a method to strengthen the bounds through the solution of knapsack problems. The approach was tested on 240 instances with up to 48 nodes, derived from the TSP Library.

Battarra, Pessoa, Subramanian, and Uchoa [3] investigated the same problem, proposing mathematical formulations as well as a branch-and-cut and a branch-and-cut-and-price algorithm. The latter algorithm proved to be very effective and solved to optimality all the instances proposed in [7].

A constraint that can remind our draft constraint has been considered by Ma, Cheang, Lim, Zhang, and Zhu [10], who studied a vehicle routing problem with link capacity constraints, in which road links (i.e., arcs) have limitations on the tonnage of the vehicles allowed to travel along them.

Differently from other generalizations of the TSP (see, e.g., Cordeau, Nossack, and Pesch [5]), the TSPPDD does not have a natural decomposition into simpler problems. In the

next section we present a mathematical model for the TSPPDD. In Section 5.3 we obtain a number of valid inequalities that are used in Section 5.4 to obtain a branch-and-cut algorithm. In order to provide a good initial solution to the algorithm, a heuristic and a local search approach are proposed in Section 5.5. Computational experiments are presented in Section 5.6, and conclusions follow in Section 5.7.

5.2 Mathematical model

In this section we present an *Integer Linear Programming* (ILP) formulation of the TSPPDD, and we show how it can be simplified through arc removal.

5.2.1 Integer Linear Program

For each arc $(i,j) \in A$, let x_{ij} be a binary variable taking the value 1 if and only if arc (i,j) is part of the solution, and y_{ij} be an integer variable representing the quantity of cargo on board the ship when traveling along arc (i,j).

Let us define two parameters, λ_{ij} and υ_{ij}, to represent a lower and an upper bound, respectively, on y_{ij}. The former can be defined as

$$\lambda_{ij} = \begin{cases} d_i & \text{if } i \in \{1,\dots,n\} \text{ and } j \in \{1,\dots,n\} \cup \{n+i\}\ ; \quad (5.1) \\ -d_j & \text{if } i,j \in \{n+1,\dots,2n\}; \quad (5.2) \\ d_i - d_j & \text{if } i \in \{1,\dots,n\} \text{ and } j \in \{n+1,\dots,2n\} \setminus \{n+i\}; \quad (5.3) \\ 0 & \text{otherwise.} \quad (5.4) \end{cases}$$

In case (5.1) i is an origin and j is either another origin or the destination of i: a ship traveling along (i,j) must carry at least the cargo picked up at i. In case (5.2) both i and j are destinations: the cargo destined to j must be on board when traveling along (i,j). In case (5.3) i is an origin and j is a destination either than that of i: a ship traveling along (i,j) must carry both the cargo picked up at i and the one to be delivered at j. Finally, if i is a destination and j is an origin, the ship could possibly be empty.

An obvious upper bound on y_{ij} is $\min\{l_i, l_j, Q\}$. A tighter bound may be obtained by decreasing these three quantities as

$$\upsilon_{ij} = \min\{l_i + \min\{0, d_i\}, l_j - \max\{0, d_j\}, Q - \max\{0, -d_i, d_j\}\} \quad (5.5)$$

Indeed: (i) if i is a destination then the minimum between l_i and Q may be decreased by the amount of cargo delivered at i; (ii) if j is an origin then the minimum between l_j and Q may be decreased by the amount of cargo to be picked up at j.

The TSPPDD can then be formally defined through the following *Integer Linear Programming* (ILP) model:

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \quad (5.6)$$

$$\text{s.t.} \sum_{j \in N} x_{ij} = 1 \qquad (i = 0, \ldots, 2n) \tag{5.7}$$

$$\sum_{i \in N} x_{ij} = 1 \qquad (j = 1, \ldots, 2n+1) \tag{5.8}$$

$$\lambda_{ij} x_{ij} \leq y_{ij} \leq \upsilon_{ij} x_{ij} \qquad (i, j = 1, \ldots, 2n) \tag{5.9}$$

$$\sum_{j \in N} y_{ij} - \sum_{j \in N} y_{ji} = d_i \qquad (i = 1, \ldots, 2n) \tag{5.10}$$

$$\sum_{j \in N} y_{0j} = 0 \tag{5.11}$$

$$\sum_{j \in S} x_{ij} \geq 1 \qquad (i = 1, \ldots, n; S \subset N : i \notin S \text{ and } n+i \in S) \tag{5.12}$$

$$\sum_{j \in S} x_{ij} \geq 1 \qquad (i = n+1, \ldots, 2n; S \subset N : i \notin S \text{ and } 2n+1 \in S) \tag{5.13}$$

$$x_{ij} \in \{0, 1\}, y_{ij} \in \mathbb{N} \qquad (i, j = 0, \ldots, 2n+1). \tag{5.14}$$

The objective function (5.6) minimizes the total cost of the route. Constraints (5.7) and (5.8) ensure that the ship starts from depot 0 and ends at depot $2n+1$ after having visited every port exactly once. Constraints (5.9) guarantee the feasibility of the quantity of cargo onboard at any time. Constraints (5.10) impose that all pickups and deliveries be fulfilled. Constraint (5.11) ensures that the ship starts its route with no load. The precedence constraints (5.12) enforce each origin to be visited before the corresponding destination, while constraints (5.13) impose that depot $2n+1$ be visited after all destinations. Note that constraints (5.7), (5.8), (5.12) and (5.13) together ensure that the classical subtour elimination constraints be satisfied.

5.2.2 Arc removal due to precedence, capacity and draft constraints

The ILP model can be enhanced by removing arcs from set A according to the following considerations:

- self-loop arcs (i, i) $(i \in N)$ are not considered;
- arcs of the form $(0, n+i)$ or $(i, 2n+1)$ $(i \in \{1, \ldots, n\})$ cannot be part of a feasible solution, as they would violate precedence constraints;
- arcs of the form $(n+i, i)$ $(i \in \{1, \ldots, n\})$ would make no sense in a solution;
- arcs of the form (i, j) $(i, j \in \{1, \ldots, n\})$ such that $d_i + d_j > \min\{l_j, Q\}$ would violate either the draft limit at j or the ship capacity;
- arcs of the form $(n+i, n+j)$ $(i, j \in \{1, \ldots, n\})$ such that $d_i + d_j > \min\{l_{n+i}, Q\}$ would violate either the draft limit at $n+i$ or the ship capacity;
- arcs of the form $(i, n+j)$ $(i, j \in \{1, \ldots, n\}, j \neq i)$ such that $d_i + d_j > \min\{l_i, l_{n+j}, Q\}$ would violate either the draft of i, or the draft of $n+j$, or the ship capacity.

5.3 Valid inequalities

The TSPPDD is as a generalization of the TSPPD which, in turn, is a special case of the *Precedence Constrained TSP* (PCTSP) in which the solution must satisfy precedence relations $i \prec j$ imposed to a set of node pairs. A number of valid TSPPD or PCTSP inequalities are either valid or can be adapted to the TSPPDD, as well as to other related problems (see, e.g., Xue, Luo, and Lim [19]). We considered in particular subtour-elimination, generalized order, capacity and fork cuts.

5.3.1 Subtour elimination cuts

Given a set $S \subset N$, let $A(S) = \{(i,j) : i,j \in S\}$ and $\bar{S} = N \setminus S$. The classical TSP facet-defining subtour-elimination cut is

$$\sum_{(i,j)\in A(S)} x_{ij} \leq |S| - 1 \quad \forall S \subset N. \tag{5.15}$$

We will adopt the notation of Cordeau [4], namely:

$$\sigma(S) = \{i \in N \ : \ n+1 \leq i \leq 2n \ \text{ and } \ i-n \in S\} \ \ (\textit{successor nodes});$$
$$\pi(S) = \{i \in N \ : \ 1 \leq i \leq n \ \text{ and } \ n+i \in S\} \qquad (\textit{predecessor nodes}).$$

Balas, Fischetti, and Pulleyblank [2] have lifted (5.15) for the PCTSP through the precedence constraints. As each node (but the depots) is the predecessor or successor of exactly one other node, (5.15) can be lifted in two ways. Let $\delta(S,T) = \{(i,j) \in A : i \in S, j \in T\}$. For predecessors, we have:

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\delta(S\cap\pi(S),\bar{S}\setminus\pi(S))} x_{ij} + \sum_{(i,j)\in\delta(S,\bar{S}\cap\pi(S))} x_{ij} \leq |S| - 1 \quad \forall S \subset N, \tag{5.16}$$

while for successors we have

$$\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\delta(\bar{S}\setminus\sigma(S),S\cap\sigma(S))} x_{ij} + \sum_{(i,j)\in\delta(\bar{S}\cap\sigma(S),S)} x_{ij} \leq |S| - 1 \quad \forall S \subset N. \tag{5.17}$$

Consider the relaxation of the TSPPDD obtained by eliminating the constraints on draft limits and ship capacity. The resulting problem is a special case of the PCTSP, and hence inequalities (5.16) and (5.17) are valid for the TSPPDD as well.

Another TSP facet-defining cut can be found by a different lifting of (5.15). Given a set $S \subset N$ with $h = |S| \geq 3$, and any ordering of its nodes $S = \{i_1, \ldots, i_h\}$, Grötschel and Padberg [8] proved that the following inequalities are valid for the TSP:

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=2}^{h-1} x_{i_k,i_1} + \sum_{k=3}^{h-1}\sum_{l=2}^{k-1} x_{i_k,i_l} \leq |S| - 1. \tag{5.18}$$

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=3}^{h} x_{i_1,i_k} + \sum_{k=4}^{h}\sum_{l=3}^{k-1} x_{i_k,i_l} \leq |S| - 1. \tag{5.19}$$

The dial-a-ride problem is a routing problem in which one has to design vehicle routes and schedules for a set of requests which specify pickup and delivery between origins and destinations. Cordeau [4] proved that, for such problem, the above cuts can be further strengthened by adding a term that takes into account the resulting precedence constraints, obtaining:

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=2}^{h-1} x_{i_k,i_1} + \sum_{k=3}^{h-1}\sum_{l=2}^{k-1} x_{i_k,i_l} + \sum_{j\in\bar{S}\cap\sigma(S)} x_{j,i_1} \leq |S|-1, \tag{5.20}$$

$$\sum_{k=1}^{h-1} x_{i_k,i_{k+1}} + x_{i_h,i_1} + 2\sum_{k=3}^{h} x_{i_1,i_k} + \sum_{k=4}^{h}\sum_{l=3}^{k-1} x_{i_k,i_l} + \sum_{j\in\bar{S}\cap\pi(S)} x_{i_1,j} \leq |S|-1. \tag{5.21}$$

Since the precedence constraints of the dial-a-ride problem are the same as those of the TSPPDD, these cuts are also valid for our problem.

5.3.2 Generalized order cuts

Another family of valid inequalities, called *generalized m-order constraints*, was introduced by Ruland and Rodin [17] for the TSPPD. Given m disjoint subsets $S_1,\dots,S_m \subset N$ such that none of them contains 0 or $2n+1$, if it is possible to find a sequence of nodes $i_1,\dots,i_m \in \{1,\dots,n\}$ such that:

$$\begin{aligned} i_k &\in S_k \quad (k=1,\dots,m),\\ n+i_{k+1} &\in S_k \quad (k=1,\dots,m-1),\\ n+i_1 &\in S_m, \end{aligned}$$

then the following inequality is valid:

$$\sum_{l=1}^{m}\sum_{(i,j)\in A(S_l)} x_{ij} \leq \sum_{l=1}^{m}|S_l| - m - 1. \tag{5.22}$$

It has been proved in [4] that, by taking into account the precedences induced by pickup and delivery, these cuts can be lifted in two ways:

$$\sum_{l=1}^{m}\sum_{(i,j)\in A(S_l)} x_{ij} + \sum_{l=2}^{m-1} x_{i_1,i_l} + \sum_{l=3}^{m} x_{i_1,n+i_l} \leq \sum_{l=1}^{m}|S_l| - m - 1; \tag{5.23}$$

$$\sum_{l=1}^{m}\sum_{(i,j)\in A(S_l)} x_{ij} + \sum_{l=2}^{m-2} x_{n+i_1,i_l} + \sum_{l=2}^{m-1} x_{n+i_1,n+i_l} \leq \sum_{l=1}^{m}|S_l| - m - 1. \tag{5.24}$$

Again, the validity for the TSPPDD comes from the consideration that the precedence constraints of the two problems coincide.

5.3.3 Capacity-draft cuts

Given a subset $S \subset N$, let $d(S) = \sum_{i \in S} d_i$. Consider a set S such that $d(S) > 0$, and define the *reduced capacity* with respect to S as $Q(S) = \min(Q, \max_{i \in S}\{l_i\})$ (upper bound on the load when visiting a node of S). An immediate lower bound on the number of times a vehicle must visit S is then

$$\sum_{(i,j) \in \delta(S,\bar{S})} x_{ij} = \sum_{(i,j) \in \delta(\bar{S},S)} x_{ij} \geq \lceil d(S)/Q(S) \rceil. \tag{5.25}$$

Following Ropke, Cordeau, and Laporte [16], cut (5.25) can be strengthened by considering two sets $S, T \subset N$ with $q(S) > 0$, and defining $U = \pi(T) \setminus (S \cup T)$. We obtain

$$\sum_{(i,j) \in A(S)} x_{ij} + \sum_{(i,j) \in A(T)} x_{ij} + \sum_{(i,j) \in \delta(S,T)} x_{ij} \leq |S| + |T| - \left\lceil \frac{d(S) + d(U)}{Q(S \cup T)} \right\rceil, \tag{5.26}$$

which coincides with the cut obtained by [16], with the only difference that $Q(S \cup T)$ replaces Q.

5.3.4 Fork cuts

Consider any routing problem in which a feasible path $P = (k_1, \ldots, k_r)$ becomes infeasible if two nodes $i \in S$ and $j \in T$ $(S, T \subset N)$, are added at the beginning and at the end of P. Then the *fork inequality*

$$\sum_{i \in S} x_{i,k_1} + \sum_{h=1}^{r-1} x_{k_h,k_{h+1}} + \sum_{j \in T} x_{k,j} \leq r \tag{5.27}$$

obviously holds. It has been shown in [16] that (5.27) can be strengthened through sets of nodes that produce intermediate infeasible paths. Specifically we consider subsets S, $T_1, \ldots, T_r \subset N$ such that $k_h \notin T_{h-1}$ for $h = 2, \ldots r$. If the path $(i, k_1, \ldots, k_h, j)$ is infeasible for any $h \leq r$ and any pair $(i \in S,\ j \in T_h)$, then the *outfork inequality*

$$\sum_{i \in S} x_{i,k_1} + \sum_{h=1}^{r-1} x_{k_h,k_{h+1}} + \sum_{h=1}^{r} \sum_{j \in T_h} x_{k_h,j} \leq r \tag{5.28}$$

prohibits infeasible paths obtained by prematurely leaving P. Exactly in the same way one can derive *infork* inequalities by prohibiting infeasible paths obtained by entering P at an intermediate node. As these cuts are valid for any routing problem in which one can decide whether a certain path is infeasible, they hold for the TSPPDD as well.

5.4 Branch-and-cut algorithm

We implemented a branch-and-cut algorithm based on the root-node formulation (5.6)-(5.14). At the root node we relax constraints (5.12)-(5.13), which impose precedence and subtour-elimination. At each decision node, we separate those inequalities that are violated by the current (fractional) solution. In addition to these two families of constraints, which ensure

feasibility, we generate the cuts described in Section 5.3. The branch-decision tree exploration is managed by a general purpose software (e.g, CPLEX). In this section we describe how the model was strengthened and how the cuts were separated.

5.4.1 Strengthened model

In order to strengthen the root-node formulation, we added two sets of constraints to the relaxed model.

Classical *2-cycle elimination* constraints

$$x_{ij} + x_{ji} \leq 1 \quad \forall\, (i,j) \in A : j > i \;\text{ and }\; (j,i) \in A. \tag{5.29}$$

Property 5.4.1. *In spite of their simplicity, constraints* (5.29) *are not implied by the relaxed model* (5.6)-(5.11), (5.14). *Indeed*

Proof. It is enough to consider the case $i \leq n$, $j > n$, $j \neq n+i$, $d_j = -d_i$. Solution $x_{ij} = x_{ji} = 1$, $y_{ij} = d_i$, $y_{ji} = 0$ does not violate (5.10), but it violates (5.29). □

There are $O(n^2)$ potential 2-cycle elimination constraints, hence their addition to the model is not computationally heavy. The experiments showed however that they have limited impact on the solution quality, so we developed the following specialized constraints, that gave much better results.

Draft oriented *2-path elimination* constraints

$$x_{ij} + x_{jk} \leq 1 \quad \forall\, i,j,k \in \{1,\dots,2n\} : \text{certain conditions (see below) hold.} \tag{5.30}$$

Property 5.4.2. *Inequalities* (5.30) *are valid for the following cases (corresponding to the enumeration of all possible characterizations of i,j,k), in which a path (i,j,k) would violate either a draft (cases 1-6) or a precedence (cases 7 and 8) constraint (see Figure 5.2, where pickup nodes are drawn bigger than delivery nodes, and the value on an arc gives the minimum load the ship would have when traveling along it):*

1. $i \leq n, j \leq n, k \leq n$ *and* $d_i + d_j + d_k > \min(Q, l_k)$.
2. $i \leq n, j \leq n, k > n, k \neq n+i, k \neq n+j$ *and either* $d_i + d_j - d_k > \min(Q, l_j, l_k)$ *or* $d_i - d_k > \min(Q, l_i, l_j)$;
3. $i \leq n, j > n, k \leq n, j \neq n+i$ *and* $d_i + d_k > \min(Q, l_k)$;
4. $i \leq n, j > n, k > n, j \neq n+i, k \neq n+i$ *and either* $d_i - d_j - d_k > \min(Q, l_i, l_j)$ *or* $d_i - d_k > \min(Q, l_j, l_k)$;
5. $i > n, j \leq n, k > n, k \neq n+j$ *and* $-d_i - d_k > \min(Q, l_i)$;
6. $i > n, j > n, k > n$ *and* $-d_i - d_j - d_k > \min(Q, l_i)$;

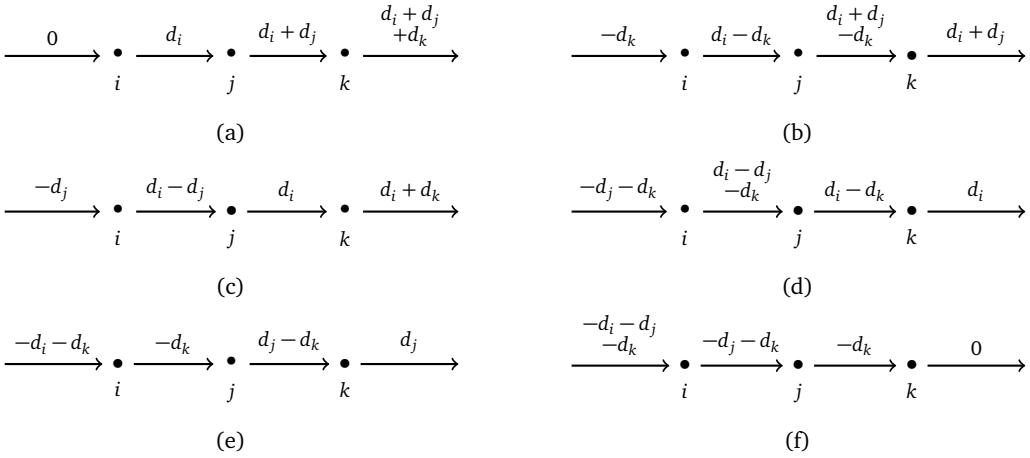


Figure 5.2: Minimum load on board a ship traveling along arcs (i, j) and (j, k).

7. $i > n, j \leq n, k \leq n$ *and* $i = n + k$;

8. $i > n, j > n, k \leq n$ *and* $i = n + k$;

Proof. Consider Case 1: the load on the arc leaving k would be at least $d_i + d_j + d_k$ (Figure 5.2a). Very similar reasonings, immediately emerging from the figures, prove: Case 2 (Figure 5.2b), and note that the last condition is equivalent to $d_i - d_k > l_i$); Case 3 (Figure 5.2c); Case 4 (Figure 5.2d, and note that the last condition is equivalent to as $d_i - d_k > l_k$); Case 5 (Figure 5.2e); Case 6 (Figure 5.2f). In cases 7 and 8 no draft violation occurs, but the precedence condition between i and k would be violated. □

The number of potential 2-path constraints is $O(n^3)$ but their inclusion into the model proved to be effective. Note in addition that, as these constraints represent incompatibilities between pairs of arcs, it would be possible to aggregate some of them into stronger clique inequalities, representing incompatibilities between subsets of arcs. This is however automatically done by the solver we used (CPLEX), so there would be no advantage in doing it explicitly.

5.4.2 Cut separation

The *precedence inequalities* (5.12) and (5.13) can both be separated exactly in polynomial time through series of max-flow problems. Violated inequalities (5.12) can be found by solving n max-flow problems from i to $n + i$ $(i = 1, \ldots, n)$, where the arc capacities are the values of variables x_{ij}. Violated inequalities (5.13) can be found by solving, in an analogous way, n max-flow problems from $n + i$ to $2n + 1$ $(i = 1, \ldots, n)$. Details on these separation methods can be found, e.g., in Padberg and Hong [13].

All the cuts discussed in Section 5.3 were instead separated in a heuristic way. A heuristic separation method for *subtour elimination cuts* (5.16) and (5.17) was given by [4]. Observe that, for any set $S \neq \emptyset$, the arcs incident with all nodes of S can either belong to $\delta^+(S)$, or to

$\delta^-(S)$, or to $A(S)$ (in which case they appear twice), and hence

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} + 2\sum_{(i,j)\in A(S)} x_{ij} = 2|S|. \tag{5.31}$$

By combining (5.31) with twice (5.16) in one case, and twice (5.17) in the other, one obtains

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} - 2\sum_{(i,j)\in\delta(S\cap\pi(S),\bar{S}\setminus\pi(S))} x_{ij} - 2\sum_{(i,j)\in\delta(S,\bar{S}\cap\pi(S))} x_{ij} \geq 2 \tag{5.32}$$

$$\sum_{(i,j)\in\delta^+(S)\cup\delta^-(S)} x_{ij} - 2\sum_{(i,j)\in\delta(\bar{S}\setminus\sigma(S),S\cap\sigma(S))} x_{ij} - 2\sum_{(i,j)\in\delta(\bar{S}\cap\sigma(S),S)} x_{ij} \geq 2 \tag{5.33}$$

We therefore heuristically search for subsets S violating (5.32) or (5.33), using the simple Tabu search scheme proposed by Augerat [1] for the capacitated vehicle routing problem. Consider the separation of (5.16) through (5.32). The search starts from an empty set S and iteratively adds or removes elements from S, trying to minimize the left hand side of (5.32). When a node is removed from S, its insertion is marked as tabu for a certain number of iterations. In addition, at each iteration, if $|S| \geq 3$, the current set S is also used to check whether (5.20) is violated: in fact, we can choose i_1 of (5.20) as the node with the largest outflow and compute the left-hand side of (5.20) by numbering all other nodes at random. A similar procedure is used for separating (5.17) through (5.33) as well as, if $|S| \geq 3$, for checking whether (5.21) is violated.

We separate *generalized order cuts* (5.23) and (5.24) only for $m = 3$ and $|S_l| = 2$ ($l = 1, 2, 3$) as, for larger values, they become computationally very expensive. Notice that in this case sets S_l can be written as:

$$S_1 = \{i_1, n+i_2\}, \quad S_2 = \{i_2, n+i_3\}, \quad S_3 = \{i_3, n+i_1\}$$

and equation (5.23) becomes:

$$x_{i_1,n+i_2} + x_{n+i_2,i_1} + x_{i_2,n+i_3} + x_{n+i_3,i_2} + x_{i_3,n+i_1} + x_{n+i_1,i_3} + x_{i_1,i_2} + x_{i_1,n+i_3} \leq 2. \tag{5.34}$$

For every possible choice of $i_1 \in \{1,\dots,n\}$, we find the node $i_2 \in \{1,\dots,n\}$ such that the three terms containing only indices $i_1, i_2, n+i_2$ in the lhs of (5.34) are maximized. Then, we find the node $i_3 \in \{1,\dots,n\}$ that maximizes the other five terms. In other words, for (5.23)

$$i_2 = \arg\max_{1\leq j\leq n} \{x_{i_1,n+j} + x_{n+j,i_1} + x_{i_1,j}\}; \tag{5.35}$$

$$i_3 = \arg\max_{1\leq j\leq n} \{x_{i_2,n+j} + x_{n+j,i_2} + x_{j,n+i_1} + x_{n+i_1,j} + x_{i_1,n+j}\}, \tag{5.36}$$

and analogously, for (5.24):

$$i_2 = \arg\max_{1\leq j\leq n} \{x_{i_1,n+j} + x_{n+j,i_1} + x_{n+i_1,n+j}\}; \tag{5.37}$$

$$i_3 = \arg\max_{1\leq j\leq n} \{x_{i_2,n+j} + x_{n+j,i_2} + x_{j,n+i_1} + x_{n+i_1,j}\}. \tag{5.38}$$

We separate *capacity-draft cuts* (5.26) using the procedure detailed in [16] which starts with sets $S = \{i\}$ and $T = \{n + j\}$ for all possible $i, j \in \{1, \dots, n\}$ and tries to augment these sets at each iteration.

Finally, *fork cuts* are separated in both their basic version (5.27), and in the strenghtened infork and outfork versions (see (5.28)). The path $P = (k_1, \dots, k_r)$ that forms the backbone for the cut is constructed as follows. We fix a node $k_0 \in \{1, \dots, 2n\}$ and we consider all paths $(k_0, k_1, \dots, k_r)$ for $r \geq 2$, that can be constructed by adding arcs corresponding to base columns of the linear relaxation of the problem. In other words, arc (i, j) is used to extend the path only if $x_{ij} > 0$. For each such path, set T is constructed as

$$T = \{j : j \notin P \text{ and } (k_0, k_1, \dots, k_r, j) \text{ is infeasible}\},$$

and the corresponding set S is

$$S = \{i : i \notin P \text{ and } (i, k_1, \dots, k_r, j) \text{ is infeasible for all } j \in T\}.$$

Notice that, by construction, $k_0 \in S$. An inequality (5.27) is added whenever it is violated by the current choice of P, S, and T. For non-violated inequalities, we attempt lifting into outfork and infork inequalities. For example, we attempt to find a violated outfork inequality (5.28) by adding, in a greedy way, as many nodes as possible to sets $T_1, \dots, T_r$. Attempting this procedure for all r values would clearly be computationally too expensive, and hence, on the basis of preliminary experiments, we only considered paths with $r \leq 6$. In addition, whenever we check a sub-path for feasibility, we store the result in a hash table from which it can be retrieved at a later time. The feasibility check ensures that no precedence constraint is violated and that the draft limits are respected, by assuming that the ship is has the minimum possible load when it enters the sub-path.

5.5 Heuristic algorithms

In this section we present the heuristics used to obtain feasible initial solutions to the TSPPDD. We will call an origin-destination pair $(i, n+i)$ a *request*. We will call an *insertion* of a request in a partial path a couple $(p_{\text{orig}}, p_{\text{dest}})$ that indicates the positions in the partial path where, respectively, the origin and the destination of the request are inserted. Our approach consists of two constructive heuristics, followed by a refinement procedure.

5.5.1 Constructive heuristics

Our constructive heuristics start with an empty path and proceed by inserting one request at a time, until no requests are left (and hence an initial feasible solution has been obtained). We considered two approaches, denoted as *Sorted Insert* and *Best Insert*. In the former approach, the requests are preliminarily ordered according to some score that only depends on the requests themselves, and then are inserted one by one in such order: the current request is inserted in a position chosen according to a heuristic criterion. In the latter approach, at each iteration, each non-inserted request is assigned a score and a possible insertion, and the request with the highest score is correspondingly inserted.

The heuristics build a solution by using two kinds of scores, one related to the requests, and one related to their insertion. The *request scores* are

R1 the cost $c_{i,n+i}$ of the origin-destination arc;

R2 the value $\min(l_i, l_{n+i}) - d_i$ of the additional load the ship can carry when entering the two ports.

In order to introduce the insertion scores, let us define, for a path P:

- $c_P = \sum_{(i,j)\in P} c_{ij}$, the cost of the path;
- $d_P = \sum_{(i,j)\in P:1\le i\le n} d_i$, the total load picked up along the path;
- $w_P = \sum_{(i,j)\in P}(\min\{Q, l_i, l_j\} - y_{ij})$, where y_{ij} is the load of the ship when traveling along arc (i, j): w_P represents the waste of capacity along the path.

The *insertion score* is assigned to a possible insertion $(p_{\text{orig}}, p_{\text{dest}})$ by considering the extended path P given by the insertion. Four scores (the lower, the better) were evaluated:

I1 c_P, the cost of the new path;

I2 $c_P d_P$, a measure that favors paths with low cost, while giving priority to requests with low demand;

I3 $c_P + \rho d_P$, where $\rho > 0$ is a prefixed parameter, a measure similar to the previous measure, but with lesser impact of d_P. (We adopted, on the basis of preliminary computational experiments, the value $\rho = 1$);

I4 $c_P w_P$, a measure that favors paths with low cost and high capacity utilization.

Four *Sorted Insert* procedures were obtained by sorting the requests according to decreasing or increasing request score R1 or R2. For each of them, the insertion was decided using, as insertion score, either I1 or I4 (note that I2 and I3 need not be considered, since once the current request has been fixed, d_P is constant for all insertions). In total this results in eight different implementations.

Four *Best Insert* procedures were obtained by respectively evaluating, for each non-inserted request, insertion scores I1-I4. For each of them, two implementations were obtained by selecting the next request and position either as the one providing the smallest insertion score, or the one providing the largest *regret*, i.e., the largest difference between the second minimum and the minimum insertion score (or the insertion score, when only one insertion is feasible). In this case too we thus obtained eight different implementations.

For the values of n we used in our computational experiments, the CPU time taken by these procedures is negligible, hence all of them were executed (and refined, as shown in the next section). Other scores were attempted too, but the sixteen implementations we described were the only non dominated ones.

5.5.2 Refinement

The feasible solutions produced by the constructive heuristics were improved through a very simple Tabu search, defined by the following ingredients:

- *move*: three-opt (see Lin [9]) with check on the feasibility of the resulting solution. Notice that, for an oriented graph, every triplet of arcs has just one possible recombination;
- *Tabu list*: for each move, the cheapest removed arc is stored;
- *Tabu tenure*: a prefixed parameter (having value 30 in our implementation);
- *halting criteria*: a prefixed maximum number of iterations, or of iterations with no improvement. (We used values 50000 and 500, respectively, in our experiments).

5.6 Computational experiments

The exact and heuristic approaches of the previous sections were implemented in C++ and run on an Intel Xeon 3.10 GHz with 8 GB RAM, equipped with four cores. In order to allow future fair comparisons, all the experiments were performed by setting to one the number of threads.

We used IBM ILOG CPLEX 12.6 as ILP solver for the branch-and-cut algorithm of Section 5.4. Remind that we relax the precedence and subtour-elimination inequalities (5.12)-(5.13): at each decision node, the inequalities that are violated by the current solution are separated and added via a CPLEX callback. The additional valid inequalities of Section 5.3 were not generated at each decision node: the decision about separation is taken according to different probabilistic distributions, depending on the number of explored nodes and on the specific cut. Namely, the probability of separation linearly decreases from 1 to α for nodes 1–100, from β to γ for nodes 101–20 000, while it is set to γ for all subsequent nodes. Good values of α, β and γ were determined, through preliminary computational experiments, as

- subtour elimination cuts: $\alpha = 0.9$, $\beta = 0.5$, $\gamma = 0.05$;
- generalized order cuts: $\alpha = 1$ (always separated), $\beta = 1$, $\gamma = 0.1$;
- capacity-draft cuts: $\alpha = 0.75$, $\beta = 0.125$, $\gamma = 0.0125$;
- fork cuts: $\alpha = 0.75$, $\beta = 0.0625$, $\gamma = 0.00625$.

We randomly generated our benchmark starting from the eight instances of the TSPLIB [14] that have been used in [7] and in [3] to generate benchmarks for the TSP with draft limits: bayg29, burma14, fri26, gr17, gr21, gr48, ulysses16, and ulysses22. From each TSP instance we obtained TSPPDD instances having $2n + 2$ nodes, with $n \in \{10, 14, 18, 22\}$, as follows. For each value of n,

$\|N\|$	C	Basic model		2-cycle		2-path		Subt. elim.		Gen. order		Cap.-draft		Fork		B&C	
		Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final	Root	Final
22	0.1	4.06	0.00	3.88	0.00	4.02	0.00	4.01	0.00	4.01	0.00	3.78	0.00	2.65	0.00	2.47	0.00
22	0.3	16.84	2.30	16.69	2.33	18.03	2.22	16.27	1.87	16.42	1.78	16.31	1.76	14.28	0.33	14.20	0.32
22	0.5	20.35	2.98	20.69	3.21	21.06	2.84	19.96	2.10	20.12	3.05	20.09	2.95	19.67	1.92	19.23	1.52
22	2.0	9.99	0.00	10.05	0.00	9.71	0.00	8.74	0.00	9.90	0.00	9.53	0.00	9.77	0.00	8.72	0.00
30	0.1	15.95	6.51	15.56	6.46	15.78	6.63	15.83	6.19	15.87	6.07	15.22	5.65	12.37	2.67	12.35	2.41
30	0.3	27.47	19.35	27.42	19.09	27.45	19.25	27.00	18.23	27.43	18.95	26.97	18.63	26.52	17.11	25.93	16.02
30	0.5	24.34	15.96	24.29	16.20	24.34	16.16	23.71	14.54	24.30	16.03	24.30	16.01	23.96	15.51	23.44	14.01
30	2.0	10.27	0.83	10.28	0.77	10.27	0.74	9.89	0.14	10.20	0.79	10.28	0.82	10.27	0.83	9.89	0.09
38	0.1	19.74	15.07	19.47	15.06	19.74	15.02	19.68	14.88	19.69	14.82	18.92	13.79	16.96	9.45	16.00	8.88
38	0.3	28.74	24.74	28.51	24.74	28.61	24.66	28.55	24.49	28.54	24.78	28.45	24.58	28.20	23.93	27.78	22.39
38	0.5	23.53	19.53	23.63	19.50	23.53	19.54	23.43	19.03	23.15	19.01	23.53	19.37	23.32	18.96	23.15	18.45
38	2.0	10.47	4.16	10.48	4.20	10.47	4.15	10.43	3.71	10.44	3.98	10.43	4.49	10.47	4.62	10.43	4.56
46	0.1	24.67	21.59	24.57	21.51	24.67	21.56	24.61	21.50	24.58	21.61	23.71	20.54	21.14	15.85	19.94	15.17
46	0.3	36.79	34.64	36.74	34.59	36.74	34.49	36.70	34.40	36.79	34.67	35.93	33.54	36.40	34.14	35.25	31.35
46	0.5	29.68	27.40	29.40	27.40	29.56	27.42	29.62	27.07	29.71	27.42	29.57	27.23	29.59	27.37	29.23	26.12
46	2.0	15.27	12.82	15.32	12.79	15.24	12.43	15.23	11.94	15.26	12.66	15.27	12.96	15.27	12.80	15.23	11.94
Average		21.82	14.96	21.72	14.96	21.62	14.69	21.57	14.48	21.70	14.81	21.40	14.51	20.50	13.22	18.33	10.83

Table 5.1: Effect of elimination constraints and cuts on the percentage gaps between upper and lower bound.

C	$\mathscr{P}$	$\|N\| = 22$				$\|N\| = 30$				$\|N\| = 38$				$\|N\| = 46$			
		CH	TS	B&C	OPT	CH	TS	B&C	OPT	CH	TS	B&C	OPT	CH	TS	B&C	OPT
0.1	0	0.14	0.00	0.00	8	1.79	1.03	1.03	7	8.86	6.20	6.20	5	12.34	8.05	7.97	0
0.1	0.33	0.37	0.00	0.00	8	1.53	0.16	0.16	7	11.38	8.75	8.75	2	16.39	12.49	12.49	0
0.1	0.67	0.38	0.00	0.00	8	4.44	3.39	3.39	6	13.26	9.71	9.71	3	23.96	19.29	19.29	0
0.1	1	0.10	0.00	0.00	8	5.94	5.07	5.07	4	14.01	10.85	10.85	1	26.00	20.96	20.94	0
0.3	0	0.29	0.00	0.00	8	18.18	16.26	16.19	1	28.04	22.99	22.99	0	37.58	32.36	32.36	0
0.3	0.33	0.91	0.00	0.00	8	20.46	17.19	17.19	0	31.81	26.92	26.92	0	41.52	35.48	35.48	0
0.3	0.67	0.72	0.00	0.00	8	19.38	16.37	16.37	0	29.10	22.50	22.50	0	40.44	31.37	31.37	0
0.3	1	1.74	1.27	1.27	7	18.58	14.35	14.35	0	26.29	17.16	17.16	0	34.27	26.17	26.17	0
0.5	0	4.37	2.14	2.14	5	23.98	21.53	21.53	0	31.98	25.77	25.77	0	39.18	32.94	32.94	0
0.5	0.33	4.37	2.67	2.67	6	20.66	17.60	17.60	0	27.08	24.59	24.59	0	35.84	32.74	32.74	0
0.5	0.67	2.81	0.74	0.73	7	16.32	13.76	13.76	1	19.18	16.61	16.61	0	33.14	23.88	23.88	0
0.5	1	1.80	0.68	0.53	7	4.39	3.14	3.14	4	8.95	6.81	6.81	2	19.47	14.90	14.90	0
2.0	0	0.00	0.00	0.00	8	0.28	0.09	0.09	7	5.58	4.56	4.56	3	13.64	11.94	11.94	1
Average		1.38	0.58	0.56	7.38	11.99	10.00	9.99	2.85	19.66	15.65	15.65	1.23	28.75	23.27	23.27	0.08
CPU secs		0.01	2.23	431		0.02	10.32	2398		0.06	32.56	3175		0.14	94.01	3549	

Table 5.2: Percentage gaps of the upper bounds produced by the constructive heuristic, the Tabu refinement, and the branch-and-cut algorithm with respect to the best lower bound.

- a TSP node was randomly selected as the starting and ending depot (TSPPDD nodes 0 and $2n+1$). Then n origin-destination pairs were randomly selected from the remaining TSP nodes, together with the corresponding costs. A TSP node was allowed to be selected more than once, but not for the same pair;
- the n demands d_j were randomly generated in the interval $[1, 100]$;
- four sets of instances were obtained by setting the ship capacity to $Q = 50\,n\,C$, with $C \in \left\{\frac{1}{10}, \frac{3}{10}, \frac{1}{2}, 2\right\}$, as follows:
  - for each $C \in \left\{\frac{1}{10}, \frac{3}{10}, \frac{1}{2}\right\}$, four instances were produced by: (i) randomly selecting, with probability $\mathscr{P} \in \left\{0, \frac{1}{3}, \frac{2}{3}, 1\right\}$, nodes j $(1 \leq j \leq 2n)$ that will have a binding draft; (ii) randomly generating the draft l_j of each selected node in the interval $[|d_j|, Q-1]$; (iii) setting the draft of the non-selected nodes to Q. Note that, for $\mathscr{P} = 0$, no node has a binding draft, so we can evaluate our methods also on the special case given by a capacitated TSPPD;
  - for the same reason, for $C = 2$, we only generated a single instance with all nodes having draft $Q = 100\,n$, i.e., we obtained an uncapacitated TSPPD instance.

In total, we obtained 13 instances for each value of n, i.e., 52 TSPPDD instances for each TSP instance, and hence an overall benchmark of 416 instances. The computer code and the instances are available at https://github.com/alberto-santini/tsppddl. The results of the computational experiments are reported in Tables 5.1 and 5.2.

Table 5.1 examines the impact of strengthening constraints (Section 5.4.1) and valid inequalities (Section 5.3). The table considers the separate inclusion of each constraint or cut and reports, for each of them, the percentage gaps (at the root node and final, i.e. after 1 hour CPU time) with respect to the best known upper bound. For different values of n and C, the first two columns give the percentage gaps for the basic model (5.6)-(5.14), the last two columns give the percentage gaps for the branch-and-cut algorithm (Sections 5.4 and 5.5) while the other pairs of columns refer to the separate addition of constraints and cuts. An additional row gives the average gaps over the 416 instances.

The results after 1 hour CPU time (columns 'Final') show that fork cuts are the most powerful inequalities for smaller capacity values, while subtour elimination cuts frequently obtain better results for larger capacities. In a single case ($|N| = 46$, $C = 0.3$) capacity-draft cuts prevail: disaggregated results show that they produce the best gap for 14 instances out of 32. In many cases subtour elimination, generalized order, and capacity-draft cuts produce similar gaps. The results at the root node (columns 'Root') exhibit a similar behavior. The last two columns show that an effective combination of the various cuts within the branch-and-cut algorithm produce by far the best results. There is a single exception for $|N| = 38$ and $C = 2.0$, where subtour elimination beats branch and cut: it must be noted, however, that, as previously described, such capacity value produces uncapacitated TSPPD instances.

Table 5.2 provides the percentage gaps of the upper bounds with respect to the best lower bound. For different values of C and $\mathscr{P}$, the table contains four groups of four columns (one group for each number of nodes). In each group, the first three columns provide the percentage gaps between the upper bounds produced by the constructive heuristic of Section

5.5.1 (column CH), the tabu refinement of Section 5.5.2 (column TS), and the branch-and-cut algorithm (column B&C) with respect to the final lower bound value obtained by the branch-and-cut algorithm of Section 5.4. The fourth column of each group gives the number of instances (out of 8) solved to proven optimality by the branch-and-cut algorithm. Two additional rows give the average values over the 104 instances generated for each number of nodes, and the average CPU times (in seconds) required by the three algorithms.

The results show that the branch-and-cut algorithm is very effective for the instances with 22 nodes (92% of instances solved), while, as it could be expected, its behavior worsens for larger instances with 30, 38, and 46 nodes (36%, 15%, and 0.01% of instances solved, respectively). The same consideration holds for the B&C optimality gaps. The heuristic algorithms exhibit a satisfactory behavior: within very short CPU times (below 2 minutes, on average), the constructive heuristic and its simple Tabu search refinement give feasible solutions not much worse than those produced by the branch-and-cut algorithm (starting from such solutions) after one hour. By restricting the analysis to the 150 instances for which a provably optimal solution has been obtained, one can observe that the optimality gap of the constructive heuristic was 0.987% and that of the Tabu search refinement was 0.013%. Note however that the CPU time requested by branch-and-cut is not uselessly spent, as it allows to certify optimality or to evaluate the actual optimality gap.

Overall, the outcome of our computational experiments proves that taking into account realistic constraints like ship capacities and draft limits considerably increases the difficulty of finding optimal TSP solutions. Consider for example the line of Table 5.2 corresponding to $C = 2.0$, i.e., to uncapacitated TSP instances with pickup and delivery, and observe that both the gaps and the numbers of optimally solved instances are considerably better than the average values in the subsequent line. This is also confirmed by the fact that the algorithms in [6] for the TSPPD, as well as those in [3] for the TSPDL were able to solve larger instances of the respective problems. On the other hand, the good performance of the constructive heuristic and of its Tabu search refinement indicate that such algorithms can be profitably used for practical purposes.

5.7 Conclusion

We have studied for the first time a realistic variant of the classical traveling salesman problem with pickups and deliveries, that arises in maritime transportation. Considering the ship capacities and the draft limits of the ports to be visited is a crucial addition for realistically modeling problems in which one has to schedule the sequence of ports to be visited by a container ship. We have defined an integer linear programming model and we have shown how valid inequalities developed for the traveling salesman and the vehicle routing problem can be adapted to our problem. We have developed heuristic approaches and an exact branch-and-cut algorithm. Extensive computational experiments on instances of realistic size have shown that exactly solving this problem variant is extremely challenging. However, we have seen that approximate solutions of good quality (and hence particularly useful to practitioners) can be obtained within short computing times. Future developments could extend the study to the multi-vehicle case. Indeed, while the tramp shipping business is

usually interested in scheduling one ship at a time, liner shipping operators are faced with the problem of planning the routes of a whole fleet.

Acknowledgements

Research supported by Air Force Office of Scientific Research (Grants FA9550-17-1-0025 and FA9550-17-1-0067) and by MIUR-Italy (Grant PRIN 2015).

Bibliography

[1] P. Augerat, J.M. Belenguer, E. Benavent, A. Corberán, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106(2-3):546–557, 1998.

[2] E. Balas, M. Fischetti, and W.R. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical Programming*, 68(1-3):241–265, 1995.

[3] M. Battarra, A.A. Pessoa, A. Subramanian, and E. Uchoa. Exact algorithms for the traveling salesman problem with draft limits. *European Journal of Operational Research*, 235(1):115–128, 2014.

[4] J-F Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[5] M. Cordeau, J. Nossack, and E. Pesch. Mathematical formulations for a 1-full-truckload pickup-and-delivery problem. *European Journal of Operational Research*, 242:1008–1016, 2015.

[6] I. Dumitrescu, S. Ropke, J.-F. Cordeau, and G. Laporte. The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming*, 121(2):269–305, 2010.

[7] J. Glomvik Rakke, M. Christiansen, K. Fagerholt, and G. Laporte. The traveling salesman problem with draft limits. *Computers & Operations Research*, 39(9):2161–2167, 2012.

[8] M. Grötschel and M.W. Padberg. Lineare charakterisierungen von travelling salesman problemen. *Zeitschrift für Operations Research*, 21(1):33–64, 1977.

[9] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[10] H. Ma, B. Cheang, A. Lim, L. Zhang, and Y. Zhu. An investigation into the vehicle routing problem with time windows and link capacity constraints. *Omega*, 40(3): 336–347, 2012.

[11] Enrico Malaguti, Silvano Martello, and Alberto Santini. The Travelling Salesman Problem with pickups, deliveries, and draft limits. *Omega (to appear)*, pages 1–17, 2017. doi: 10.1016/j.omega.2017.01.005.

[12] T.E Notteboom and B. Vernimmen. The effect of high fuel costs on liner service configuration in container shipping. *Journal of Transport Geography*, 17(5):325–337, 2009.

[13] M. Padberg and S. Hong. On the symmetric travelling salesman problem: A computational study. *Mathematical Programming Study*, 12:78–107, 1980.

[14] G. Reinelt. Tsplib–a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[15] S. Ropke and J.-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[16] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[17] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1–13, 1997.

[18] P. Tirschwell. Berth productivity: The trends, outlook and market forces impacting ship turnaround times. Port Productivity (White paper), pages 1–24. Journal of Commerce, July 2014.

[19] L. Xue, Z. Luo, and A. Lim. Exact approaches for the pickup and delivery problem with loading cost. *Omega*, 59:131–145, 2016.

6 Railway logistics: the train rescheduling problem

Abstract We consider the real-time resolution of conflicts arising in real-world train management applications. In particular, given a nominal timetable for a set of trains and a set of modifications due to delays or other resources unavailability, we are aiming at defining a set of actions which must be implemented to grant safety, e.g., to avoid potential conflicts such as train collisions or headway violations, and restore quality by reducing the delays. To be compatible with real-time management, the required actions must be determined in a few seconds, hence specialized fast heuristics must be used. We propose a fast and effective parallel algorithm that is based on an iterated greedy scheduling of trains on a time-space network. The algorithm uses several sortings to define the initial train dispatching rule and different shaking methods between iterations. The performance is further enhanced by using various sparsification methods for the time-space network. The best algorithm configuration is determined through extensive experiments, conducted on a set of instances derived from real-world networks and benchmark instances. The resulting heuristic proved able to consistently resolve the existing conflicts and obtaining excellent solution quality within just two seconds of computing time on a standard personal computer, for instances involving up to 151 trains and two hours of planning time horizon.

6.1 Introduction

Modern railways represent a major form of transport with an ever-growing user base, as trains are flexible in terms of travelling distance (they can be used for local, regional and long-distance services) and capacity (as they are modular by nature). Furthermore, train transportation is usually the greenest transportation options for both goods and people.

Despite this, railways are confronted with the increase of operational costs and a fierce competition from other modes of transport. Many users demand more reliability in train

This chapter is based on the contents of: Andrea Bettinelli, Alberto Santini, and Daniele Vigo. A real-time conflict solution algorithm for the Train Rescheduling Problem. *Transportation Research, Part B (under revision)*, pages 1–28, 2017.

operations: a long delay, a cancelled train, a missed connection can easily decrease the perceived quality of service and turn away potential customers.

Most of the events that negatively affect train operations (broadly called *conflicts*) happen when, for some reason, there is a difference between the nominal and the actual service. The causes of such events are usually divided into *disturbances* and *disruptions* (Cacchiani et al. [6]). The former are small perturbations of the system that are handled by network operators by momentarily changing the timetable. The results of disturbances are usually minor, such as one or more delayed trains, or a platform change at a station. Disruptions, on the other hand, are major incidents that not only alter the nominal timetable, but also require changes in rolling stock and crews. The outcome of a disruption could include major delays, train cancellations, and long reroutings. Disturbances clearly happen much more often than disruptions and their impact is not to be underestimated: a train that is delayed just a few minutes can make a user miss an important connection and increase their travel time by hours. In this paper we consider both disturbances and disruptions in a unified way, by defining an algorithmics approach to handle the conflicts they cause.

Increasing systemwide reliability is crucial at every phase of the planning process. It starts at the strategic and tactical levels (budget allocation for maintenance, timetable robustness, etc.), but once at the operational level, it is almost impossible to avoid that day-to-day activities be disturbed by many kinds of unforeseen events.

When such an event occurs, it is the job of the *dispatcher* to restore the system in a working state. The job of dispatchers has been traditionally done by hand, based exclusively on their experience and practice. It was not until recent years that computer algorithms were developed with the aim of aiding the dispatchers in making the best decision that resolves the critical situation and minimises deviances from the nominal timetable.

In this paper we present such an algorithm, developed to solve the Train Rescheduling Problem (TRP): given a nominal timetable which has become infeasible because of one or more *conflicts* that have arisen, we are asked to produce a new conflict-free timetable that is as close as possible to the nominal one. Or, in case it is not possible to produce a conflict-free timetable, we need to warn the dispatcher about this and provide a timetable with the least possible number of conflicts.

Conflicts are all those situations that either can't physically happen (e.g., two trains occupying the same segment of track at the same time) or that can potentially compromise the safety of operations in the network (e.g., two trains running too close to each other in the same direction).

The algorithm presented in this paper is the result of a long lasting collaboration with Alstom, initiated by the company in 2012 with the aim of redesigning the optimisation algorithms incorporated in its Train Management System ICONIS. To this end, Alstom involved three important Italian research groups in specific research projects investigating various optimisation problems arising in the real time conflict resolution. As a result of such initial wide research effort, the team formed by Optit, an accredited spinoff of the University of Bologna, and the Department of Electrical, Electronic and Information Engineering of the University of Bologna, was selected to produce an innovative real-time conflict solution algorithm capable of taking into account the characteristics and constraints of practical applications which has been developed and industrialised during 2013, and extensively tested by Alstom in real-

world contexts. Recently, the new algorithm has been fully integrated in ICONIS and will be deployed at various international Alstom customers.

The paper is structured as follows. In the next section we give an overview of how a railway system works, how it can be affected by disturbances and what it means to reschedule a train. In Section 6.3 we review the existing literature on the TRP, based on the classification schema given by Cacchiani et al. [6]. In Section 6.4 we give a mathematical description of a railway network, of train timetables and of the relationship between them. We present an heuristic algorithm for the solution of the TRP in Section 6.5. We then describe the instances used and provide computational results in Section 6.6. Finally, we draw conclusions and propose further research paths in Section 6.7.

6.2 Timetables and conflicts

Nominal timetables are the crucial part of any railway systems. They describe in detail the trip of each train, from its departure to its arrival station, including all the intermediate stations where the train stops or passes by. This includes not only those parts of the trip where the train operates passenger service, but also all the movements necessary to perform service and maintenance, e.g., rolling stock relocation, cleaning, technical service.

Every arrival and departure is scheduled at specific time slots, which are calculated in advance by taking into account physical properties (e.g., track curvature and gradient, maximum allowed speed, train length) and interaction among trains. Clearly two trains can't occupy the same portion of tracks at the same time, but other constraints usually have to be respected. For example trains have to respect *headway times*, i.e., a minimum amount of time must be left as a buffer between trains travelling in the same direction. Another example are *dwell times* at platforms, which are needed to board and alight passengers.

Timetables can be *periodic* or *aperiodic*. Periodic timetables repeat themselves at certain time intervals (e.g., every second hour and every hour during peak times). Although such timetables are usually appreciated by customers, as they are easy to memorise and use, they are difficult to implement in a competitive market where many train operators are likely to request access to the same resources at the same time. For this reason, trains are often scheduled in aperiodic timetables. The name *aperiodic* is slightly misleading, since these timetables are repeated day after day so, strictly speaking, they have a period of one day.

Timetables are implemented by assigning *tasks* to *rolling stock* and *crews*. When it comes to passenger transportation, rolling stock are usually composed of one or more locomotives and many passenger cars; or, in case of multiple unit (MU) trains (MU trains are those composed by one or more similar self-propelled train cars), by one or more MUs. A crew includes a train driver and one or more train guards. Finally, a task represents a complete trip of the rolling stock and the crew from the train origin to its destination. The set of tasks carried out by rolling stock and crews in a day is called a *shift*, or duty.

Since in most countries the railway infrastructure is operated by a different actor than the trains, the timetables are usually created and managed by an *infrastructure manager*, who tries to accommodate the requests of train operators as much as possible, while abiding to safety rules and other operational constraints. Once the timetables are set up, train operators

will assign rolling stock and crews to the corresponding tasks.

During real-life operations a train can easily deviate from its nominal timetable: extra time might be needed at a station to board and alight passengers, weather conditions might force the driver to slow down in certain parts of the route, etc. These are examples of *primary* delays. A delayed train, in fact, could interfere with the operations of other trains, in turn delaying them (*secondary* delays) and many delays can end up knocking on from one train to another.

As already mentioned in Section 6.1, in this work we consider disturbances and disruptions (introduced in Section 6.1) under a unified umbrella. A detailed list of the conflicts we consider is given in Section 6.4.3. The corrective actions that our algorithm will suggest are limited to *retiming*, *respeeding*, and *rerouting* trains, collectively named *rescheduling*. Retiming consists in changing the durations of train stops at stations. Respeeding changes the times trains enter and leave different parts of the network (i.e., changing their speed). Finally, rerouting consists in assigning a train a new path in the network.

Several criteria can be considered when rescheduling a set of trains. For example, we may want to minimise the deviance from the nominal timetable, or the total delay, or the number of broken connections, etc. In our work, we present a general way of modelling events in the network, that is able to take into account all of these criteria (and many more).

6.3 Literature Review

Conflict resolution in train applications, often known as the *train dispatching problem* (see, e.g., Meng and Zhou [33]), is widely studied in the literature, and research contributions can be classified in several ways.

A first possible subdivision may take into account the level of detail used in modelling the physical resources composing the train network. In this respect, the main distinction was usuallly between *microscopic* and *macroscopic* modelling approaches. A microscopic approach would represent every element of the rail infrastructure in detail (individual tracks, platforms, etc.). In such a model every network element can be assigned to only one train at a time, thus leading to *explicit* capacity requirements on the resources. A typical macroscopic model, on the other hand, would disregard any fine-grained segmentation of the tracks, thus leading to *cumulative* capacity requirements, since each network element could represent several physical resources. In the literature, such models are also known, respectively, as *single-track* and *N-track* models (see, e.g., Törnquist and Persson [45]). This distinction, however, is often blurry, and several authors adopted a mixed approach, by considering so-called *mesoscopic* models, in which the modelling detail is not specified a priori. Here, network elements can represent either low level infrastructure, such as specific tracks or platforms, or aggregate one, such as entire stations or N-track segments.

Another widely used subdivision takes into account the type of conflict resolution actions available to the decision makers. These include the application of *retiming*, *reordering*, *re-tracking*, and *rerouting* of trains (Meng and Zhou [33]). Such actions involve, respectively: the adjustment of speeds and stopping times; modifying the order in which trains occupy platforms or track segments; small and large changes in the path followed by trains in the

network.

In their recent survey, Cacchiani et al. [6] also adopted a classification scheme which mainly takes into account the type of conflict to be managed by the model. More precisely, the authors distinguished between *disturbances* and *disruptions* and analysed the literature classifying models and solution approaches based on this viewpoint. The reader is also referred to Törnquist and Persson [45], Meng and Zhou [33] for additional literature analyses and classification. Other classification schemes proposed in recent surveys mainly focus on the solution methodology adopted (see Fang et al. [21]) or on dynamic and stochastic components related to on-line rescheduling (see Corman and Meng [10]). Finally, we direct the interested reader to the recent book of Hansen and Pachl [22] for a comprehensive analysis of many aspects of railway timetabling and operations, including train rescheduling.

Many works which employ a more microscopic approach revolve around the concept of *alternative graph*, introduced by Mascis and Pacciarelli [31] for the no-wait job shop scheduling. The problem of assigning a train to a track segment for a certain period of time, in fact, can be seen as a job shop scheduling problem where track segment are machines and the assignment of a train to a segment is an operation. Additional constraints, such as set-up times and no-wait constraints, are used to model specific characteristics of the problem. The alternative graph formulation was widely used to develop solution approaches to various rescheduling problems (see, e.g., D'Ariano et al. [17]) such as the ROMA tool (see, e.g., D'Ariano et al. [18, 19], Corman et al. [11, 12, 13, 14, 15], D'Ariano and Pranzo [16]).

Other approaches, which use alternative solution paradigms, have also been explored. Rodriguez [36] solved conflicts using constraint programming techniques and using the job shop model with additional constraints. Meng and Zhou [32] propose a stochastic programming model is used to reschedule trains on a single-track line, so that the new schedule is robust. Pellegrini et al. [35] solve a real-time traffic management problem using a pure Mixed-Integer Programming (MIP) model which represents a small section of a railway network with fine granularity. Samà et al. [39] use an ant-colony optimisation metaheuristic to select the best routing alternative for each train in a real-time setting. A simulation-based approach for train dispatching was proposed by Li et al. [30]. Finally, Mu and Dessouky [34] employs fuzzy optimisation techniques to reschedule trains after a low-probability disruption occurs.

Several authors tried to bridge the gap between fine-grained and more aggregate representations by using different techniques. For example, Lamorgese and Mannino [28, 29] propose an iterative macro- and microscopic approach, in which the *line traffic control* problem takes care of the macroscopic constraints (trains meeting at stations, stations' capacities respected) and acts as a master problem. The *station traffic control* considers instead detailed constraints at the station level and acts as a subproblem to generate cuts for the master problem, in a way analogous to Benders decomposition. Other mesoscopic approaches have been based on MIP formulations: Törnquist and Persson [46] used an exact model for rescheduling on N-track networks; Törnquist [44] used a MIP-based greedy heuristic starting from the same model; this model was further extended by Acuna-Agost et al. [1], who also consider intermediate stops and bidirectional tracks.

While minimising the total delay is a sensible choice, in recent years the focus of rescheduling techniques has been shifting towards a more passenger-oriented point of view, which aims to minimise the travellers' delay. In this spirit, Schöbel [42] solved the delay manage-

ment problem, consisting in deciding which connections between trains should be maintained, even when this would mean to introduce some delay on certain trains that would have to wait for others. The work has been expanded in Schöbel [43], Schachtebeck and Schöbel [41], Dollevoet et al. [20], while Kanai et al. [27] propose a combined optimisation/simulation algorithm that allows to track additional performance indicators other than total passenger delay. On the other hand, concering the scheduling of freight trains, Mu and Dessouky [34] recently proposed effective heuristic approaches based on decomposition.

6.4 Problem description

Given a description of the current state of the network, the goal of our algorithm is to produce a new timetable for the trains, keeping in mind what was the original, nominal timetable published to the users. There are, therefore, three main objects that we need to model to provide input data to the algorithm: the first is a description of the *physical network*; the second is the *nominal timetable*, the third is the current status of the trains in the network (called the *forecast timetable*).

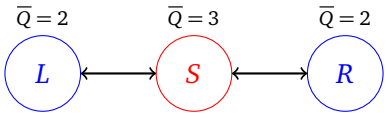
6.4.1 Network and timetables

The main tool we use to represent the train network is the *network (di)graph* $G_N = (V,A)$. Nodes in V represent resources. What a resource is can vary greatly and depends mostly on the level of detail we want to achieve when modelling the train network. At a microscopic level, a resource could be a single section of track between two signals, a platform at a station, a junction between tracks, etc. On a macroscopic level, it could be a whole station, or a set of parallel tracks between two stations, etc.

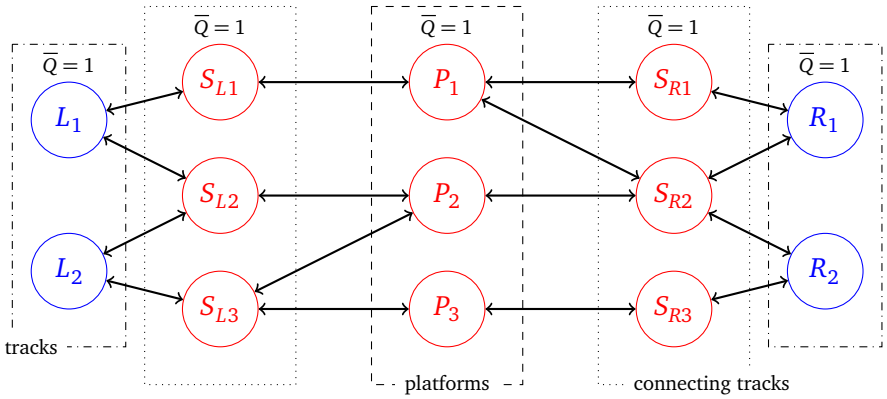
There are, of course, trade-offs between macroscopic and microscopic representations. While the latter will produce a larger graph, using the former will lead us to lose some information. For example, when we represent a set of parallel tracks as a single resource, we can't guarantee that a feasible assignment of trains to the tracks always exists.

Talking about resources rather than more specific railway elements, however, allows us to generalise many aspects of railway networks and even to mix micro- and macroscopic representations in the same graph. This is useful, for example, when the central part of the network is particularly congested and needs a higher level of detail, while peripheral parts are less loaded and can be modelled at a lower resolution.

For example, Figure 6.1a shows a macroscopic modelling of a station S and two set of tracks L and R. Figure 6.1b shows the same station and tracks at a microscopic level: the station has been substituted by three platforms and the generic set of tracks have been replaced by a node for each physical track. Furthermore, connecting tracks have been introduced, to model the connections between the platforms and the tracks. Notice that a solution that was feasible in Figure 6.1a might not be feasible in Figure 6.1b. For example, a train leaving P_1 to reach R_2 and a train leaving P_2 to reach R_1 can't depart at the same time, as they would violate the capacity of S_{R2} (which is 1), but we wouldn't have been able to rule out this solution just by looking at Figure 6.1a and the capacities of the aggregate nodes.



(a) Macroscopic representation of station S with tracks L on its left and R on its right.



(b) Microscopic representation of the same station as in Figure 6.1a, with all platforms and physical tracks modelled explicitly.

Figure 6.1: Differences between the micro- and macroscopic representations of a station. $\overline{Q}$ represents the capacity of a resource.

We identified certain properties that apply to all resources, no matter what parts of the physical network they represent:

- Every node $v \in V$ has an ideal (or *soft*) capacity $Q_v \in \mathbb{N}$ and a *hard* capacity $\overline{Q}_v \in \mathbb{N}$. The capacity of a resource indicates the number of trains that can occupy it at the same time. While the soft capacity can be violated (by possibly paying a certain penalty) the hard capacity cannot be violated under any circumstances. The relation $Q_v \leq \overline{Q}_v$ holds.

- Every node $v \in V$ also has an associated boolean parameter, $\omega_v \in \{0, 1\}$, that indicates whether overtaking and crossing between trains can happen at the node.

The arcs in set A represent the possibility for trains to move from one node to another. Arcs also have capacities, indicating the number of trains that can simultaneously transit from the source to the destination node of the arc: we indicate the capacity of $a \in A$ with $\overline{Q}_a \in \mathbb{N}$. This quantity is considered as a hard capacity.

The other main actors of a train network are, naturally, trains. Let I be the set of trains and consider the following properties that link together resources and trains:

- Given a train $i \in I$ and a resource $v \in V$, we give the minimum and maximum *travel times*, i.e., the minimum and maximum times that i is allowed to occupy v. We denote these values with $m_{i,v}$ and $M_{i,v}$ respectively. The physical meaning of these quantities can vary depending on what the resource models. In case of a section of track, $m_{i,v}$ is given by the length of the track and the maximum speed that the train can achieve on that track. On the other hand, in case of a platform, $m_{i,v}$ is the dwelling time.

- Given a resource $v \in V$, we denote with h_v the minimum *headway* at v, i.e., the time that must elapse between two trains occupying the resource.

The nominal timetable describes the ideal operational status of the network. Each train $i \in I$ has a predefined path in the network, denoted as $p_i = (v_1, v_2, \ldots, v_{k_i})$, which is simply a sequence of resources to be visited: $v_1, \ldots, v_{k_i} \in V$.

For each node in the path of train i, the nominal timetable also provides the times at which the train is supposed to enter and leave the node. These times are denoted as $\theta^{\text{in}}_{i,v_j}$ and $\theta^{\text{out}}_{i,v_j}$ respectively.

The current train plan describes the network as it is at the present moment — and as it is forecast to be in the future, given the information available. For this reason, such a plan is also called the *forecast timetable*. In an ideal scenario, the forecast timetable is always equal to the nominal one. In practice, when a disturbance or a disruption occurs, the forecast diverges from the nominal timetable.

The forecast timetable has a formal structure which is similar to that of the nominal timetable: it gives a sequence of nodes that each train must visit, together with the expected in- and out-times. Since both timetables describe the (ideal and real, respectively) situation of the network before the dispatcher takes any decision regarding rerouting, the train paths must be the same in both.

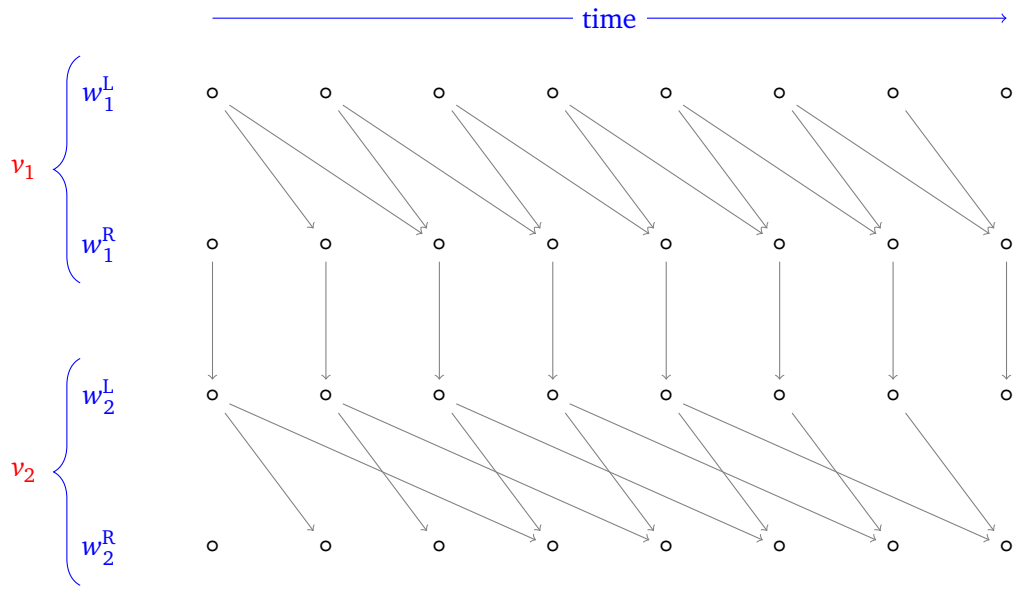


Figure 6.2: Example of a portion of time expanded graph.

The only new parameters associated with the forecast are, therefore, the in- and out-times. To account for the uncertainty that comes with the real-time situation of the network, we actually give pairs of minimum and maximum possible in- and out-times. These values should be considered as hard values, i.e., the train cannot possibly enter a node before the minimum in-time or after the maximum in-time.

The minimum in- and out-times are denoted, respectively, as t^{in}_{i,v_j} and t^{out}_{i,v_j}, for $j = 1,\dots,k$. The maximum in- and out-times are T^{in}_{i,v_j} and T^{out}_{i,v_j}.

As we mentioned in Section 6.2, rescheduling a train can involve rerouting it. This means that the dispatcher is allowed to change the path of the train in the network. In our model, we assume that it is only possible to choose detours from a predefined set available for each train. The set of detours associated with train i is D_i.

A detour is nothing more than a path in the network, so an element $d \in D_i$ contains a sequence of nodes: $d = (v_1, v_2, \dots, v_k)$. We only require that both the first and the last node of the detour are also part of the original train path p_i. Furthermore, similarly to what we have seen for the current train plan, maximum and minimum in- and out-times are given for each node v_j of the detour. These are denoted as t^{in}_{i,d,v_j} (minimum in-time), T^{in}_{i,d,v_j} (maximum in-time), t^{out}_{i,d,v_j} (minimum out-time), and T^{out}_{i,d,v_j} (maximum out-time).

6.4.2 Time-space graph

The network graph $G_N = (V,A)$ introduced in Section 6.4.1 does not explicitly model the time component. In this subsection, we present a time-space graph and we construct it starting from G_N and augmenting the number of nodes to take into account time and entry/exit points

of nodes of V. Time-expanded graphs have already been used to model railway networks, e.g., in Caprara et al. [8] and Cacchiani et al. [5]. The *time-space (di)graph* of a train $i \in I$ is denoted as $G^i_{\text{TS}} = (V^i_{\text{TS}}, A^i_{\text{TS}})$ and is obtained in the following way.

For every entry and exit point of every node $v \in V$, a node is added to V^i_{TS}. The definition of entry and exit point is strongly dependent on the physical resource modelled by v. For example, if v represents a set of parallel tracks, there will be one entry and one exit point for each track; if v models a station, there would be an entry and one exit point for every track running through the station. In general, the number of entry and exit point does not need to match (e.g., a station could have more tracks one side than the other). The names *entry* and *exit* are only used to distinguish two physical locations on the resource, but since trains can generally run on a resource in both directions, a specific train could actually enter the resource from one of its exit points and leave it from one of the entry points.

We then need to model time into the graph. In order to do this, we first have to decide a reasonable time horizon and a time discretisation. In practical applications, these values could be provided to the model by the upstream conflict detection system. Notice, though, that the flexibility bundled with our model allows us to use different time discretisations in different parts of the time-space graph: some resources or some time intervals can be modelled with a more precise time discretisation than others. For example, it is possible to have a denser time discretisation for peak times and a sparser one for low-congestion times (e.g., at night). A denser discretisation might also be necessary for short tracks, where the travelling time could be shorter than the standard time interval. Once the time discretisation and the time horizon have been fixed, each node gets one further copy per time instant.

Finally, two dummy nodes σ^i_{src} and σ^i_{snk} are added to each graph G^i_{TS}. They represent, respectively, a source and sink node used as the start and end point of the train's path in the graph.

Arcs are created between pairs of nodes $(w_1, w_2) \in V^i_{\text{TS}}$ and they are divided in three types. The first type links nodes which represent entry and exit point relative to the same resource $v \in V$. Such an arc would model the travelling of a train along the resource modelled by v, when the difference in time instants represents a feasible travelling time for train i.

The second type links nodes which represent entry and exit points of adjacent resources, that is of nodes $v_1, v_2 \in V$ such that $(v_1, v_2) \in A$. Such an arc would model a train that leaves a resource and (instantaneously) reaches a new one.

Finally, the third type links the source and the sink to the other nodes. Let w^i_{s} and w^i_{e} be the entry points that train i has to use to access and leave, respectively, its start and end resources. We then add to the arc set A^i_{TS} a: (a) arcs from σ^i_{src} to nodes of the form (w^i_{s}, t), where t is a time instant; (b) arcs from nodes of the form (w^i_{e}, t) to σ^i_{snk}, where t is a time instant; (c) arcs from nodes of the form (w, T) to σ^i_{snk}, where w is any entry or exit point, and T is the last time instant of the time horizon, used to represent a train that could not reach its destination within the time horizon considered.

We list the three type of arcs separately and let $A^i_{\text{TS}} = A^{i,1}_{\text{TS}} \cup A^{i,2}_{\text{TS}} \cup A^{i,3}_{\text{TS}}$, where the three sets contain, respectively, arcs of the three types listed above.

Figure 6.2 shows a portion of a time-expanded graph. Nodes w^{L}_1 and w^{R}_1 are the left and right extreme points of $v_1 \in V$, while nodes w^{L}_2 and w^{R}_2 are the left and right extreme points of

$v_2 \in V$. The arcs between w_1^{L} and w_1^{R} represent the traversal of resource v_1 and, analogously, the arcs between w_2^{L} and w_2^{R} represent the traversal of resource v_2. Different arcs having the same source node model the different travelling times associated to different speeds. The vertical arcs between w_1^{R} and w_2^{L} represent the possibility of moving from v_1 to v_2.

The arcs in $A_{\mathrm{TS}}^{i,1}$ can be mapped back to the resources and the time intervals they represent in the following way. For each arc $a \in A_{\mathrm{TS}}^{i,1}$, let $\rho(a) \in V$ be the underlying resource modelled by the arc; analogously, let $\tilde{\rho}(a) \in V \times \{-1, +1\}$ be the directed underlying resource, used to distinguish the direction in which the resource is being traversed; let $l(a)$ be the length of the associated resource $\rho(a)$. Let also $\lambda_{\mathrm{s}}(a)$ and $\lambda_{\mathrm{e}}(a)$ be the start and end time of arc a, i.e. the times when the train (respectively) occupies and frees the resource.

6.4.3 Constraints

As defined in Section 6.1, conflicts are those situations that either can't physically happen or that would compromise the safety of operations, and their resolution plays the same role as satisfying a constraint in a Mixed Integer Programme (MIP). In order to give a more precise description of the contraints presented in the rest of this section, we give some mathematical formulation in which we use the notation $x_a^i \in \{0, 1\}$ as a variable in a Mixed Integer Programme, having value 1 iff the arc $a \in A_{\mathrm{TS}}^i$ is part of the path of train i.

Preliminary experiments with solving a compact MIP formulation of real-life instances with a commercial solver have shown that model generation alone can take several minutes, and solving the root node requires more than a day. For this reason, we do not include a complete MIP model for the problem we are presenting. Rather, the notation x_a^i should be seen as a way to describe precisely the constraints taken into account by our algorithm, and how they are reflected on the time-space graph.

Notice, first of all, that a train schedule can be modelled as a path in G_{TS}^i, starting in σ_{src}^i and ending in σ_{snk}^i and abiding to the usual flow conservation constraint. Formally, this means that:

$$\sum_{a \in A_{\mathrm{TS}}^{i,+}(\sigma_{\mathrm{src}}^i)} x_a^i = 1 \tag{6.1}$$

$$\sum_{a \in A_{\mathrm{TS}}^{i,-}(\sigma_{\mathrm{snk}}^i)} x_a^i = 1 \tag{6.2}$$

$$\sum_{a \in A_{\mathrm{TS}}^{i,-}(w)} x_a^i = \sum_{a \in A_{\mathrm{TS}}^{i,+}(w)} x_a^i \qquad \forall w \in V_{\mathrm{TS}}^i \setminus \{\sigma_{\mathrm{src}}^i, \sigma_{\mathrm{snk}}^i\} \tag{6.3}$$

where $A_{\mathrm{TS}}^{i,+}(w)$ (resp. $A_{\mathrm{TS}}^{i,-}(w)$) is the set of all arcs outgoing from (resp. incoming to) node $w \in V_{\mathrm{TS}}^i$.

In this work we deal with the rescheduling of the trains once the conflicts have already been detected and reported, so we will assume that the complete list of conflicts is available together with the current train plan. In other words, the conflict detection system works upstream of our system. This assumption is not restrictive, as a simple linear-time algorithm over the current train plan is able to produce the complete list of conflicts.

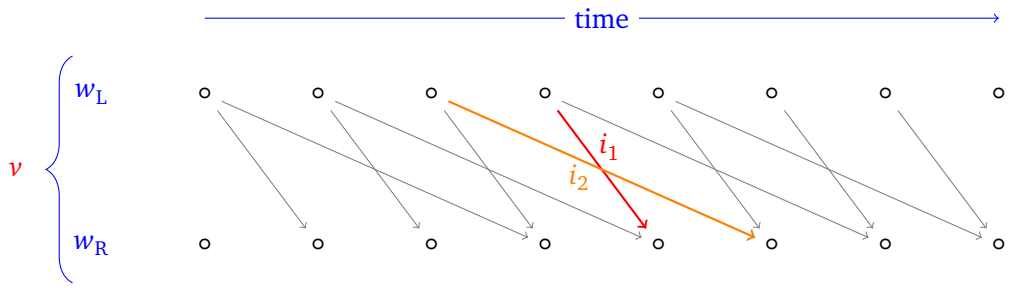


Figure 6.3: Train i_1 overtaking train i_2 on resource v. Notice how this situation corresponds to crossing arcs in the time-space graph.

The presence of conflicts could be formally detected by checking for violations in (hard) constraints involving the variables x_a^i. As we will see in Section 6.4.4, we want to penalise the violation of certain soft constraints. In a MIP model, for example, a hard capacity limit can be enforced via a constraint, whereas the extent of the violation of a soft capacity limit can be penalised, by introducing an auxiliary variable that plays the role of the slack variable relative to the constraint.

Illegal crossing and overtake

An illegal crossing (overtake) describes a situation when a train would cross (overtake) with another one, on a resource v where this is illegal, i.e. $\omega_v = 0$. An example of overtaking is described in Figure 6.3.

An overtake corresponds to the violation of the following constraints: for each train i, each resource v where overtaking is forbidden, and each arc $a \in A_{\text{TS}}^{i,1}$ such that $\rho(a) = v$:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A_{\text{TS}}^{j,1} \\ \tilde{\rho}(a') = \tilde{\rho}(a), \\ \lambda_s(a') > \lambda_s(a), \\ \lambda_e(a') \leq \lambda_e(a)}} x_{a'}^j + x_a^i \leq 1 \tag{6.4}$$

Equation (6.4) states that train j overtakes train i on resource v if j arrives in v after i, but leaves v before i, and both trains travel in the same direction.

Analogously, corresponds to the violation of the following contraints: for each train i, each resource v on which crossing is forbidden, and each arc $a \in A_{\text{TS}}^{i,1}$ such that $\rho(a) = v$:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A_{\text{TS}}^{j,1} \\ \rho(a') = \rho(a), \\ \tilde{\rho}(a') \neq \tilde{\rho}(a), \\ \lambda_e(a) \geq \lambda_s(a')}} x_{a'}^j + x_a^i \leq 1 \tag{6.5}$$

Equation (6.5) states that train j crosses train i on resource v if j arrives in v before i has left, and the two trains travel in opposite directions.

Capacity violation

A capacity violation occurs when the number of trains simultaneously occupying a resource is greater of the hard capacity of the resource. Such a conflict corresponds to a violated inequality of the following type:

$$\sum_{i \in I} \sum_{\substack{a \in A^i_{\mathrm{TS}} \\ \rho(a)=v, \\ \lambda_{\mathrm{s}}(a) \leq t, \\ \lambda_{\mathrm{e}}(a) \geq t}} x^i_a \leq \overline{Q}_v \tag{6.6}$$

for each resource v and each time instant t, where we remind that $\overline{Q}_v$ is the hard capacity of resource v.

Headway violation

Such a conflict occurs when a train occupies a resource that has been occupied by another train, and not enough time has elapsed between the first train leaving the resource and the second one entering it. For each train $i \in I$ and each arc $a \in A^{i,1}_{\mathrm{TS}}$ corresponding to a resource $\rho(a)$ on which crossing and overtaking is forbidden ($\omega_{\rho(a)} = 0$, as otherwise the headway must not be respected), a headway conflict corresponds to a violated constraint:

$$\sum_{\substack{j \in I \\ j \neq i}} \sum_{\substack{a' \in A^{j,1}_{\mathrm{TS}} \\ \rho(a')=\rho(a) \\ \lambda_{\mathrm{e}}(a') \geq \lambda_{\mathrm{s}}(a) - h_{r(a)} \\ \lambda_{\mathrm{s}}(a') \leq \lambda_{\mathrm{e}}(a) + h_{r(a)}}} x^j_{a'} + x^i_a \leq 1 \tag{6.7}$$

Time dependencies

Avoiding the conflicts described in the previous subsection is usually enough to come up with a new plan that allows safe operations and limits the deviations from the nominal timetable. Unfortunately, this is not always enough to provide a holistic, good solution.

Consider, for example, a passenger on a delayed train that risks missing his connection. From his point of view, a solution that also delays his next train (to "wait for him") is preferable to a solution that does not. But from a train operator's point of view, a solution that does not delay the second train may be considered better, since no delay is better than some delay.

Then, if the train operator's *service intention* is, for example, "moving passengers from A to B" (even at the cost of increasing the overall delay, to some extent), this should be taken into account by the rescheduling algorithm.

In order to take into account service intentions, we introduce the concept of *time dependencies* (Caimi et al. [7]). A time dependency is a relationship of precedence between two events happening in the network. For example, a time dependency could mandate that the event "train i_1 leaves node v" can only happen a certain time after the event "train i_2 arrives at node v"; intuitively, we would say that train i_1 needs to wait for train i_2 at node v.

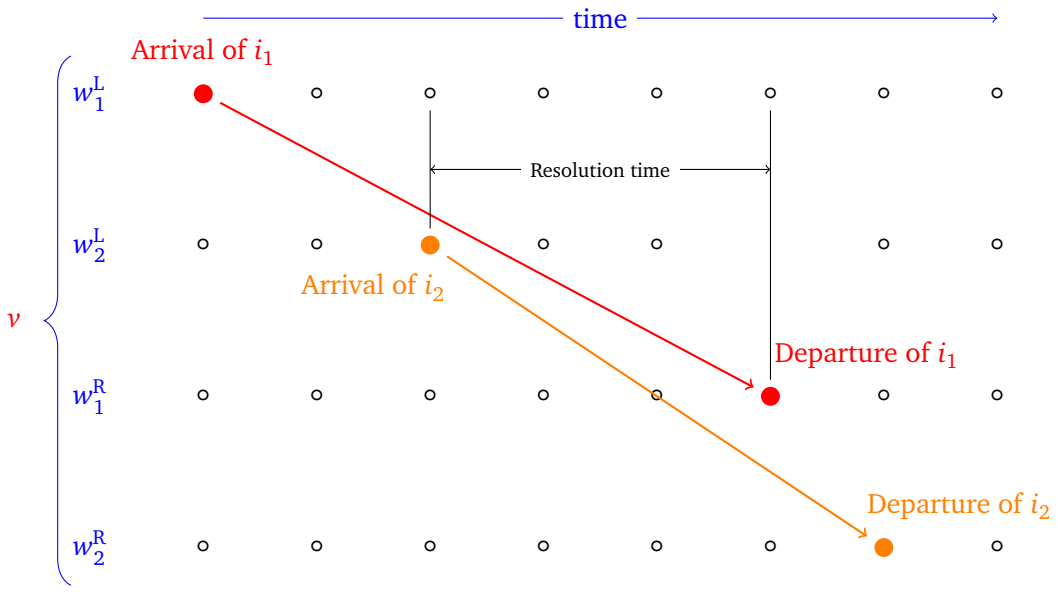


Figure 6.4: Resolution time of a time dependency between the arrival of train i_2 and the departure of train i_1 at a station.

A service intention can merely suggest a precedence between events, as in the case of passenger connections: it is preferable that the connection is kept, but this "promise" can be broken in order to improve the overall quality of service. On the other hand, the precedence can also be mandatory, as in the case of two trains that share crew or rolling stock.

Let F be the set of time dependencies. We associate to each element $f \in F$ the following values:

- The two trains $i_{f,1}, i_{f,2}$ involved in the dependency.
- The two resources $v_{f,1}, v_{f,2}$ at which the linked events need to take place, respectively.
- Two parameters $\varepsilon_{f,1}, \varepsilon_{f,2} \in \{0,1\}$ that take value 0 if the corresponding event is an arrival or value 1 if it is a departure.
- A parameter $\eta_f \in \{0,1\}$ that takes value 1 iff the dependency is mandatory.
- The minimum and maximum resolution time, φ_f and Φ_f. The dependency is considered satisfied if the two events take place at least φ_f and at most Φ_f time units apart.
- For non-mandatory (also called *logical*) time dependencies, we give a maximum waiting time w_f. The dependency must be satisfied if it is possible to do so by introducing at most w_f time units of delay. If this cannot be done, the dependency can either be satisfied or not. For example, in the case of a connection between two trains, we might require that a train waits for the other at most w_f time units. If the required wait is greater, the train can decide to break the connection.

Figure 6.4 shows an example in which train i_1 stays at station v for three time intervals after the arrival of train i_2, presumably to *wait* for passengers travelling on i_2. The resources $w_k^{\mathrm{L}}, w_k^{\mathrm{R}}$ represent entry and exit points of two platform at the station ($k = 1, 2$).

The violation of a mandatory time dependency $f \in F$ can be detected as follows. Assume wlog that the event relative to train $i_{f,1}$ needs to take place before the event relative to train $i_{f,2}$. For $k = 1, 2$ and $a \in A_{\mathrm{TS}}^{i_{f,k},1}$ such that $\rho(a) = v_{f,k}$, consider the parameter:

$$\delta_{f,k}(a) = \begin{cases} \lambda_{\mathrm{s}}(a) & \text{if } \varepsilon_{f,k} = 1 \\ \lambda_{\mathrm{e}}(a) & \text{if } \varepsilon_{f,k} = 0 \end{cases}$$

We can then formulate the corresponding constraint:

$$\varphi_f \leq \sum_{\substack{a \in A_{\mathrm{TS}}^{i_{f,2},1} \\ \rho(a) = v_{f,2}}} x_a^i \delta_{f,2}(a) - \sum_{\substack{a \in A_{\mathrm{TS}}^{i_{f,1},1} \\ \rho(a) = v_{f,1}}} x_a^i \delta_{f,1}(a) \leq \Phi_f \tag{6.8}$$

Split and merge

Similar to time dependencies are split and merge operations. These are events in which the trains that enter a node are not the same that leave it. They model real-life operations such as decoupling some cars from a train, so that they can get a new locomotive and proceed to a different destination.

In the most general version, any number of trains can enter a certain node and any number of trains can leave it, so a split/merge is identified by a node $v \in V$ and two sets of trains $I_1, I_2 \subset I$ that represent in-trains and out-trains. The out-trains can leave the node only after a certain amount of time has passed since the last in-train reached it. This time accounts, in practice, for the time necessary to perform any physical coupling and decoupling, or to change crew or rolling stock. Special cases of split/merge operations are:

- *Split*, when one train enters the node, and two or more exit it.
- *Merge*, when more than one train enter the node, and only one exits it.
- *Rename*, when one train enters the node and one train exits it.

In case of split/merge events, the node's capacity is not considered, as it is assumed that it is always feasible for the event to take place in the node specified. Finally, note that these events are mandatory: for example, it is not possible that a train will be detoured around a node in which it has to undergo a split.

In the following we show how a split or merge operation can be modelled in terms of mandatory time dependencies; in this way, the conflict resulting from a missed split or merge can be detected by checking for the violation of the corresponding time dependency constraint. For example, if train i needs to be split into trains i', i'' at resource v, then said resource will be set as the destination of i and the origin of i' and i''. Furthermore, two dependencies will be created:

- $f' \in F$ links trains i and i' and has: $i_{f',1} = i, i_{f',2} = i'$; $v_{f',1} = v_{f',2} = v$; $\varepsilon_{f',1} = 0, \varepsilon_{f',2} = 1$; $\eta_{f'} = 1$; $\varphi_{f'}$ will be the time needed to perform the split operation; $\Phi_{f'}$ will be the maximum time allowed for the split to take place, if any.
- $f'' \in F$, analogously links trains i and i''.

Maximum and minimum entry, exit, and travel times

Maximum and minimum entry, exit, and travel times can be enforced by removing from the graphs G^i_{TS} those nodes that would correspond to an infeasible (resource, time) couple. For example, if a train $i \in I$ cannot enter resource v before time t, all nodes of V^i_{TS} corresponding to resource v at a time $t' < t$ can be removed from the graph. Analogously, if the minimum travel time along a resource v is t, all arcs $a \in A^{i,1}_{\text{TS}}$ such that $\lambda_{\text{e}}(a) - \lambda_{\text{s}}(a) < t$ can be removed.

6.4.4 Objective function

The objective value can be written as a function of the paths in the time-space graphs, and therefore of $\vec{x} = (x^i_a)$, in the following way:

$$f(\vec{x}) = f_1(\vec{x}) + f_2(\vec{x}) + f_3(\vec{x}) + f_4(\vec{x}) \tag{6.9}$$

The four components correspond to delays, logical dependency breaking, soft capacity violations, and the use of detours. Each of these components represents a sum of penalties that quantify how undesirable it is to incur in the corresponding violations. The penalty, therefore, is not only limited to represent the economical disadvantage of taking a particular decision (e.g., increased energy consumption) but can also represent intangible values, such as customer satisfaction. In the following, we analyse these four components.

Delays

In the nominal timetable, we associated to each train i and each resource v_j in the train's path, an ideal in-time $\theta^{\text{in}}_{i,v_j}$ and an ideal out-time $\theta^{\text{out}}_{i,v_j}$. Any deviation from these times can be penalised, by considering two piecewise-linear functions that respectively assign a cost to delays in arriving at and departing from the resource. These penalty functions are denoted as $\pi^{\text{ind}}_{i,v_j}(\cdot)$ and $\pi^{\text{outd}}_{i,v_j}(\cdot)$.

Notice that this general definition allows us to assign different penalty profiles to different resources: for example, if some resource is considered critical for a train, we can assign a higher penalty to delays at that resource. The function can also operate on negative delays, allowing us to penalise trains that arrive at a node with excessive advance. Finally, further flexibility is bundled in the piecewise-linear nature of the function: for example, we might want to have a penalty that grows linearly with the delay up to a certain point, after which a big flat penalty is assigned, as any further delay does not worsen the situation any more. This could be achieved with a penalty profile such as that in Figure 6.5.

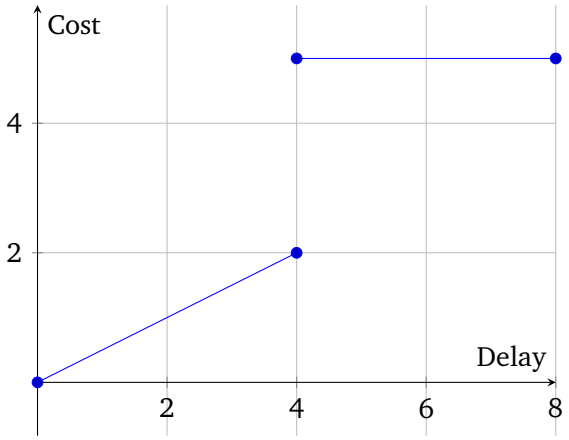


Figure 6.5: Example of a penalty profile for the arrival of a train at a certain resource, with a "jump" in cost if the delay is greater than 4 time units.

Dependency breaking

As we mentioned in Section 6.4.3, logical dependencies are not mandatory and therefore we can decide to break them. In our implementation, when such a dependency $f \in F$ is broken, we pay a penalty π_f^{dep}. (Notice that, in principle, a piecewise linear function could be used, as done for delays.)

Capacity violations

When we violate the soft capacity Q_v of a resource v, we pay a penalty π_v^{cap}. In our implementation, this penalty remains the same no matter how big the capacity violation is. (Notice that, in principle, a piecewise linear function could be used in this case too.)

Detours

Finally, a fixed penalty π_d^{det} is paid when a train i is re-routed along a detour $d \in D_i$. This penalty takes into account all the costs (economical or otherwise) incurred because of the rerouting. Dependency breaking and capacity violations are calculated separately when a detour is taken. If some node of the detour can be naturally mapped to nodes of the original path, the delay penalty can also be calculated. For example, if the detour consists of a platform change at a station, we can naturally assign to the new platform the in- and out-times at the old platform.

The penalty π_d^{det} should be considered as a fixed cost incurred by the mere fact of having rerouted the train. It can include both *real* and *virtual* costs. For example, if the detour consist of a path longer than the original one, there would be increased energy costs. But if the detour also excludes a station, there would be a much increased passenger dissatisfaction. Therefore, the magnitude of the penalty depends on the type of detour: a platform change

will have a small associated penalty, while a major change in the train's path will be associated with a bigger penalty.

Other terms

Even though in the present work we only consider the four components discussed above, further terms can be easily introduced in the objective function to take into account other indicators. For example:

- Number of modifications with respect to the nominal timetable, eventually weighted differently depending on the nodes or trains involved. This could be done to make sure that the new timetable does not disrupt too much the current operations (i.e., changes too many train paths) just to save a few seconds of overall delay.
- Increased travel time on resources, eventually involving a piecewise-linear penalty profile. This term would help avoiding unnecessary stops or excessive brake-accelerate cycles, for increased passenger comfort and reduced energy consumption.

6.5 Solution Algorithm

A solution to the TRP is a collection of paths, one for each train, in the time-space graph G_{TS}. The solution algorithm must be able to modify the forecast timetable in order to solve the conflicts, and to compute the new objective function. Key requirements for this algorithm, deriving from its real-time nature, are:

- It must produce solutions of high quality in very short computational times (a few seconds). This is due to the fact that the algorithm is used on-line by dispatcher, who needs reasonable advice in few seconds, to guarantee the safety of operations in the network. For this reason, we focussed on a heuristic approach.
- If the algorithm is not able to find a conflict-free schedule, it has to give the dispatcher a schedule with the smallest possible number of remaining conflicts. This means that solving more conflicts needs to always have priority over other factors. In order to accomplish this, we established an implicit hierarchy through our objective function. A very high penalty is assigned to each unresolved conflict, so that a solution with fewer conflicts will always prevail on one which has more, while the "standard" objective function is used to decide the best one between two solutions with the same number of remaining conflicts. Therefore, the objective function presented in (6.9) is used as a part of the overall objective function: $\tilde{f}(\vec{x}) = P \cdot N_{\mathrm{RC}}(\vec{x}) + f(\vec{x})$, where P is the large penalty to pay for each conflict, and N_{RC} gives the number of remaining conflicts in the solution.
- The algorithm should allow for a high degree of parallelisation, allowing to concurrently produce multiple rescheduled timetables that will be stored in a solution pool, from which the best one will be selected. This allows the algorithm to employ and evaluate

different strategies in situations where none of them is clearly superior to the others, thereby focussing on different key aspects of the problem.

With these requirements in mind, we now give a general description of the algorithm which boradly falls into the category of iterated greedy algorithms (see, e.g., Ruiz and Stützle [38]). After an initialisation phase in which trains are ranked according to some criterion, the algorithm iterates among two main phases, until a termination condition is met. The two phases are:

1. **Construction**: a new timetable is obtained by rescheduling the trains one by one, according to their ranking.
2. **Shaking**: the train ranking is perturbed following a set of rules.

In our case the termination criterion is a hard time limit, with early termination if the solutions didn't improve over a certain number of iterations. The next subsections will describe each phase in detail. Furthermore, in order to speed up the computational time of the construction phase, we employed a sparsification of the graph G_{TS}. This is a technique used to remove some edges and vertices from the time-space graph. Its use is justified by the fact that, by choosing a fine time discretisation, we might create a great number of edges, many of which can be removed without strongly impacting the quality of the train plans.

Notice that, for the algorithm initialisation, for the shaking phase, and for sparsification we propose several possible alternatives. Therefore, an instance of our algorithm is completely defined once we specify which initial sorting, shaking policy, and sparsification method is used. An extensive experimental testing of the proposed alternatives, described in Section 6.6.1, will help determining well-performing combinations of the algorithm's components.

6.5.1 Initial sorting

Since the algorithm constructs a schedule for one train at a time (Step 1), the order in which the trains are considered is clearly important, as trains scheduled later will be constrained by those scheduled earlier. Since there is no "natural" order of trains, we used various sorting criteria:

- **Random**: the trains are randomly sorted.
- **Congestion**: the trains are sorted according to the number of conflicts in their forecast timetable, putting trains with more conflicts first. The rationale behind this choice is that a train that generates a lot of conflicts is harder to schedule, and therefore should be scheduled earler. Using the notation introduced earlier, if $\vec{x}^*$ is the assignment of variables corresponding with the forecast timetable, then we sort the trains by decreasing value of $N^i_{\text{RC}}(\vec{x}^*)$, where $N^i_{\text{RC}}(\vec{x}^*)$ is the number of conflicts in train i's schedule, and $N_{\text{RC}}(\vec{x}^*) = \sum_{i \in I} N^i_{\text{RC}}(\vec{x}^*)$.
- **Length**: the trains are sorted according to the number of nodes in their original path in decreasing order. The longer the path, in fact, the higher the chances that a conflict will be present at some node. The trains are therefore sorted in decreasing order of the size of their set $\{a \in A^i \ : \ \vec{x}^* = 1\}$.

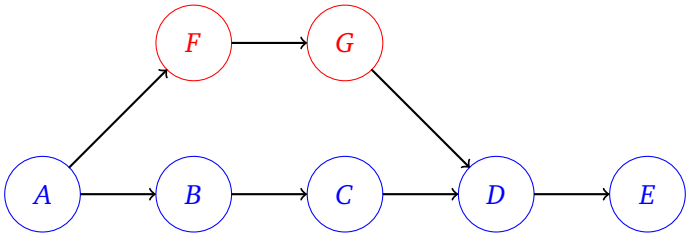


Figure 6.6: Topological order between nodes (resources) of a simple network.

- **Conflict time**: the trains are sorted according to the time instant of the earliest conflict in their forecast timetable. This is because an early conflict can impact the overall network status at a much later time. In other words, there is more freedom when fixing conflicts happening earlier, and we want to fully use this freedom.
- **Speed**: the trains with highest average speed are scheduled earlier. This strategy is based on the observation that faster trains have schedules that are more sensitive to variations. The average speed of a train i is given by:

$$\frac{\sum_{a \in A^i_{\mathrm{TS}}, \vec{x}^*_a = 1} \frac{l(a)}{\lambda_{\mathrm{e}}(a) - \lambda_{\mathrm{s}}(a)}}{\left| \left\{ a \in A^i_{\mathrm{TS}} \ : \ \lambda_{\mathrm{e}}(a) > \lambda_{\mathrm{s}}(a) \text{ and } \vec{x}^*_a = 1 \right\} \right|}$$

During a preliminary experimental phase, we noticed that using the sorting criteria in reverse order can sometimes lead to better results. For this reason we also considered the criteria **Reverse congestion** and **Reverse speed**.

6.5.2 Construction

Each train schedule is constructed by solving a shortest-path problem on the time-expanded graph G_{TS}, where the starting node correspons to the current position of the train and the ending node corresponds to the train's desired position at the end of the time horizon.

Since trains run along fixed routes, with only a few possible detours, and since they have hard constraints on the time at which they can reach and leave certain resources, the graph G_{TS} can be pruned accordingly for each train. Once this is done, the shortest path is constructed with a custom label-setting algorithm. Given a partial path to a certain node $(w, t) \in V_{\mathrm{TS}}$, the corresponding label will be $L = ((w', t'), c)$ where the node $(w', t') \in V_{\mathrm{TS}}$ is the predecessor of (w, t) in the partial path and $c \in \mathbb{R}$ is the cost of the partial path up to the current node.

Notice that, since trains are scheduled sequentially and the algorith is ran on a different time-space graph for each train, we cannot run into deadlocks. A deadlock will indeed correspond to an unresolved conflict (usually a capacity violation) and be accordingly heavily penalised in the objective function.

We discuss now two main aspects of this algorithm: the order in which we extend the labels and how we update the cost component. Labels are extended greedily, i.e., starting

from the one with the lowest cost component, but with one exception: a label related to a resource w will be extended only after all the labels related to resources $w' \prec w$ have already been extended (independently from the time interval), even if they have a higher cost. The relation $\prec$ is the topological order relation between nodes in the subgraph of G_N induced by the union of the path of the train p_i and all the detours in D_i. To better understand this rule, consider Figure 6.6, which describes the topological setup of a network with a main corridor and a possible detour (via F and G). Labels related to resource D, for example, will be extended only after all labels related to resources A, B, C, F and G have been extended, since $A \prec B \prec C \prec D$ and $A \prec F \prec G \prec D$.

The cost component is updated taking into account the various penalties included in the objective function. Some of these penalties, however, depend on the interaction between different trains. As an example, consider a connection between trains i_1 and i_2. If i_1 is scheduled before i_2, when we schedule i_1 will just assume that the connection will be satisfied. If, when scheduling i_2, we realise that the connection is broken, we will add the penalty on the objective function of i_2.

Finally, we need to take special care in case of split/merge operations, as these require the presence of multiple trains on the same resource at the same time. When the partial path of an input train reaches one of these resources, we fix the schedule of the train up to that point, by choosing the lowest cost partial path (we first explore all non-dominated partial paths up to the synchronisation point). We proceed with the construction of the schedule for the output trains, only once all the schedules of the input trains have been fixed up to the considered resource.

6.5.3 Shaking

In the shaking phase we perturb the ordering of the trains (the *dispatching sequence*) with the aim of finding an ordering which leads, through a new construction phase, to an improving solution. We present two alternative policies, inspired to two well-known metaheuristic algorithms: Reduced Variable Neighbourhood Search (RVNS) and Tabu Search (TS). As mentioned in the beginning of this section, these two alternative policies are experimentally evaluated in Section 6.6.

The RVNS (see, for example, Hansen et al. [23]) explores the solution space of the problem by employing a sequence of neighbourhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_K$. A neighbourhood structure defines a way to describe the neighbourhood of any solution x in the solution space.

Starting from a solution x, RVNS generates a new random solution $x' \in \mathcal{N}_1(x)$. If the new solution is not better than the previous one, it goes on to the second neighbourhood structure and generates a random $x' \in \mathcal{N}_2(x)$. The procedure continues until either we run out of neighbourhood structures or the new solution x' is better than the current one x. In the first case, the algorithm terminates; in the second case, the algorithm is restarted using x' as initial solution and going back to using $\mathcal{N}_1$.

The neighbourhood structures are typically such that

$$\mathcal{N}_1(x) \subseteq \mathcal{N}_2(x) \subseteq \ldots \subseteq \mathcal{N}_K(x) \tag{6.10}$$

for all solutions x. This means that while no improving solution is found, the search space around x is enlarged.

In our case, the neighbourhood structure $\mathcal{N}_k(x)$ consist in considering all dispatching orders that can be obtained from x by performing at most k swaps. From this definition, it follows immediately that (6.10) is satisfied. The trains to be moved in the new dispatching order are selected at random with a roulette wheel selection procedure where the probability associated to each train is proportional to its contribution to the objective value. The number of positions each train is moved up is again chosen at random, according to a uniform distribution in $[\mu_{\min}, \mu_{\max}]$.

The second policy is inspired to Tabu Search. Starting from a dispatching sequence, we produce a new one analogously to what done with the RVNS policy. We will place in our tabu list the precedence relations between the moved trains. For example, if we transform the sequence $x = (A, B, C, D)$ into $x' = (A, D, B, C)$, we will store the precedence relations (D, B) and (D, C). While these are in the tabu list, the relative order of trains D, B and D, C will not be inverted. If, at the next iteration, train B will be selected to be moved up, then train D will have to move together with B, so not to invert the relation (D, B); the new dispatching sequence will then be (D, B, A, C).

The number of iterations each precedence move is stored in the tabu list depends on three factors:

1. The change in the part of the objective function relative to the moved train;
2. The change in the overall objective function;
3. Whether the new solution improved the incumbent solution.

6.5.4 Sparsification

As previously mentioned, the sparsification of the graph G_{TS} is used to remove some edges and vertices from the time-space graph, to speed up the computation of shortest paths by the labelling algorithm. Its use is justified by the fact that, by choosing a fine time discretisation, we might create a great number of edges, many of which can be removed without strongly impacting the quality of the train plans.

As an example, consider a segment of track 5km long and a train that, at full speed, would travel on this segment at 100km/h. The running time of this train will be of 3 minutes. If we choose time intervals to represent 1 second, that would be 180 time instants. If the entry point is w^{L} and the exit point is w^{R}, when we consider an entry time of t, we would create all the arcs

$$\big((w^{\text{L}}, t), (w^{\text{R}}, t + 180)\big), \big((w^{\text{L}}, t), (w^{\text{R}}, t + 181)\big), \big((w^{\text{L}}, t), (w^{\text{R}}, t + 182)\big), \dots$$

up to the end of the time horizon. This level of accuracy is clearly not needed in this situation: a train that took 181 time instants rather than 180 to travel along that segment, would go at a speed of 99.45km/h which is indistinguishable from 100km/h for any practical purpose. So, even if removing some edge from the graph would — in principle — cause the algorithm

to miss some feasible train plan, if these edges are properly selected, we can easily reduce the probability to miss a train plan that would produce a considerable improvement.

Here we propose four main strategies for graph sparsification. Let v be a node in the network, i the train under consideration and $m_{i,v}$ the minimum travel time of i along v. Furthermore, let t be the entry time of the train at the node and t' the first feasible exit time, taking into account both the minimum travel time and the other constraints such as the minimum and maximum out-times $t^{\text{out}}_{i,v}$ and $T^{\text{out}}_{i,v}$. The strategies we used are the following:

Fixed step We only consider departure times starting at t' and then keeping a time instant in every s. The set of possible departure times will be

$$\left\{ t' + k \cdot s \;\middle|\; k = 0, 1, \ldots \right\}$$

Fixed step with threshold The previous strategy can be improved by specifying a threshold τ and keeping all departure times between t' and $t' + \tau$. In this way we keep those arcs that are close to t' and therefore correspond to a minimal delay of the train.

Linear This case is similar to the fixed step sparsification, but the step s is adjusted to be inversely proportional to $m_{i,v}$. The set of possible departure times is

$$\left\{ t' + k \cdot \max\left(1, \frac{m_{i,v}}{s}\right) \;\middle|\; k = 0, 1, \ldots \right\}$$

Progressive With this strategy we allow a higher density for time instants close to t', while we retain fewer arcs as long as we move away from that time instant. The idea behind this criterion is that good train plans are characterised by train schedules that are delayed as little as possible. The set of possible departure times is $\left\{t'_k\right\}$ where $t'_0 = t'$ and

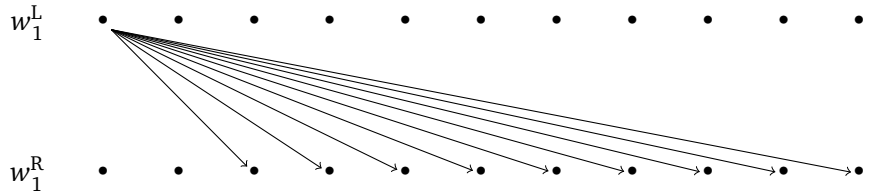
$$t'_k = t'_{k-1} + 1 + \left\lfloor \frac{t'_{k-1} - t}{s} \right\rfloor$$

In our testing we used $s \in \{2, 3, 5\}$ and $\tau \in \{5, 10, 15\}$, leading to a total of 19 combinations, including the case when sparsification is disabled. Figures 6.7a to 6.7d give a graphical representation of the sparsification methods for a segment v with endpoints w^{L}_1 and w^{R}_2 and a minimum travel time $m_{i,v} = 2$.

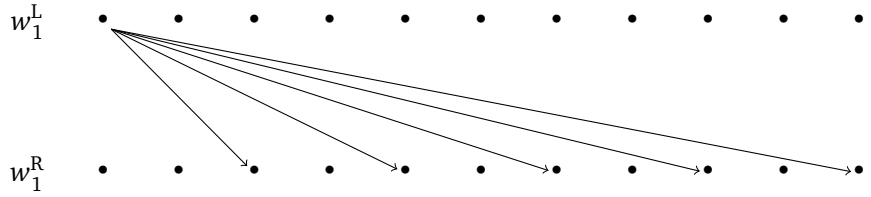
6.6 Computational Results

The aim of computational results presented in this section is to verify that the proposed approach is valid for complex instances coming from real-life applications, that describe an extended network with many trains, and an extended time horizon with a dense discretisation. We also want to validate that our algorithm achieves good results in real-time settings, where running times are limited to few seconds.

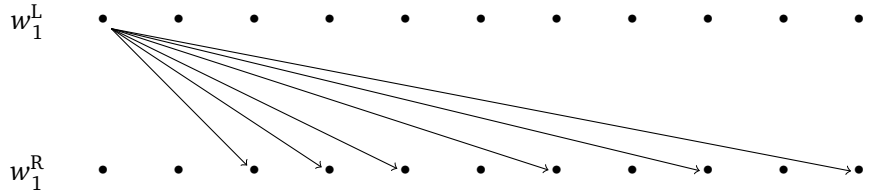
To this end, we have initially considered 23 instances, generated from three different real-world railway networks, provided to us by Alstom Transport. The first set of instances



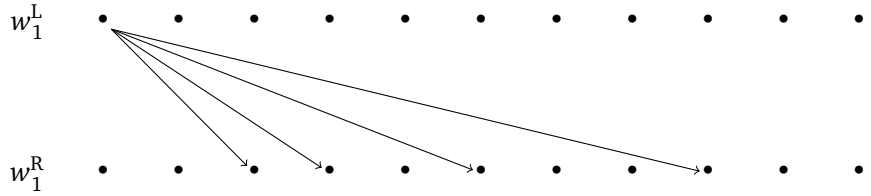
(a) No sparsification.



(b) Fixed step sparsification with $s = 2$.



(c) Fixed step sparsification with $s = 2$ and threshold $\tau = 3$.



(d) Progressive sparsification.

Figure 6.7: Graphical representation of various graph sparsification techniques on the time-space graph.

Name	# Trains	# Nodes	Time horizon (h)	Discretisation (s)	# Conflicts
N01	28	108	2	15	24
N02	16	167	2	15	15
N03	28	172	2	15	36
N04	17	112	2	15	4
N05	18	112	2	15	12
N06	17	112	2	15	2
N07	28	126	2	15	12
N08	30	132	2	15	5
N09	28	130	2	15	4
N10	30	135	2	15	3
N11	15	137	1	15	1
N12	20	153	1	15	2
N13	33	135	4	15	37
L01	139	664	1	15	20
L02	103	631	0.75	15	54
L03	131	666	1	15	62
L04	132	675	1	15	25
L05	151	673	1.25	15	97
L06	133	671	1	15	38
P01	55	859	1	10	22
P02	55	814	1	10	22
P03	61	742	1	10	72
P04	71	731	1	10	70

Table 6.1: Main characteristics of the instances provided by Alstom.

(N01-N13) describes a relatively small network chatacterised by the presence of single track lines used in both directions. The second set of six instances (L01-L06) refers to a busy regional network with a large main station and several smaller stations. Finally, the third set includes four instances (P01-P04) describing a high-speed network with frequent long-distance trains.

Table 6.1 outlines the main characteristics of the instances considered. Column "# Trains" lists the number of trains present in the instance; "# Nodes" is the number of resources, i.e., the cardinality of the set V in the network graph G_N (before time expansion); "Time horizon" is the span, in hours, of the planning horizon; "Discretisation" is the time discretisation step used, expressed in seconds; "# Conflicts" is the sum of the number of conflicts (as described in Section 6.4.3: illegal overtaking, hard capacity violation, headway time violation) plus the number of violated mandatory time dependencies, given as input in the current train plan.

In Section 6.6.1 we perform parameter tuning, to determine which graph sparsification methods and initial sortings are more likely to produce good solutions when used together with each policy (RVNS or Tabu) and time limit (2s or 10s). In Section 6.6.2 we run a simple parallel version of the algorithm on the 23 instances, using the tuned parameters.

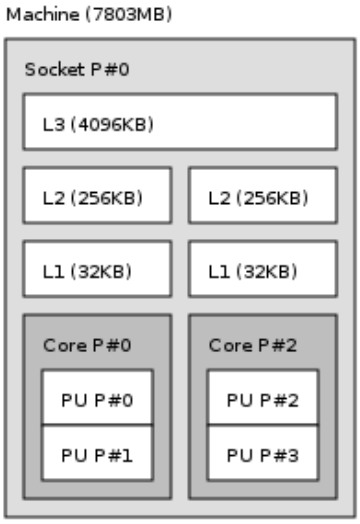


Figure 6.8: Schematic hardware configuration of the test machine.

In order to validate and benchmark our approach, we also generated new instances, which we are making publicly available. We used the network topology of the 2012 RAS Competition instances (see INFORMS [24]), in two configurations: in the first (letter "N"), each segment is modelled as a separate resource, giving an N-track scenario; in the second (letter "S") parallel tracks are not modelled separately, but as a single resource with the appropriate capacity, giving a single-track scenario. We generated 15 instances of each type, divided in groups of 3. The nominal timetable is the same for each group, while the disturbances change, so to have 5 different forecast timetables for each nominal one. Because the RAS network is smaller than the networks used in the instances of Table 6.1, in order to obtain feasible nominal timetables we had to either use fewer trains (instances of the first group), or a longer time horizon with a more coarse discretisation (instances of the second and third groups). These instances are available at [40].

To ease the comparison with other algorithms that might be developped in the future, we did not consider problem-specific features such as time dependencies, and splits and merges. The conflicts that can arise, therefore, are limited to headway, hard and soft capacity, crossing, and overtake violations. Soft violations are penalised with a simple linear function. Table 6.2 describes the features of the generated instances; columns "H", "O", and "C" give a detailed breakdown of the type of conflicts: headways, overtake, and hard capacity, respectively. In Section 6.6.2 we apply the tuned parallel algorithm to the new instances, similarly to what we do for the proprietary instances.

The tests presented in this section have been run on a dual-core 3.2GHz Intel i5 machine, with 7803MB of RAM. The CPU configuration and the L1, L2 and L3 cache sizes are detailed in Figure 6.8, as produced by the software Hwloc (Broquedis et al. [4]).

				# Conflicts		
Name	**# Trains**	**Time horizon (h)**	**Discretisation (s)**	**H**	**O**	**C**
N-1-1	7	4	15	0	0	6
N-1-2	7	4	15	63	0	43
N-1-3	7	4	15	35	2	41
N-1-4	7	4	15	35	1	42
N-1-5	7	4	15	0	0	32
N-2-1	12	12	60	2	0	3
N-2-2	12	12	60	4	0	6
N-2-3	12	12	60	2	0	4
N-2-4	12	12	60	0	0	7
N-2-5	12	12	60	6	0	13
N-3-1	24	12	60	4	0	25
N-3-2	24	12	60	105	0	53
N-3-3	24	12	60	0	0	6
N-3-4	24	12	60	103	0	89
N-3-5	24	12	60	0	0	10
S-1-1	7	4	15	0	0	3
S-1-2	7	4	15	32	0	22
S-1-3	7	4	15	4	1	28
S-1-4	7	4	15	30	1	24
S-1-5	7	4	15	2	0	19
S-2-1	12	12	60	0	0	2
S-2-2	12	12	60	2	0	2
S-2-3	12	12	60	6	0	5
S-2-4	12	12	60	0	0	4
S-2-5	12	12	60	2	0	4
S-3-1	24	12	60	0	0	21
S-3-2	24	12	60	59	0	25
S-3-3	24	12	60	0	0	4
S-3-4	24	12	60	60	0	60
S-3-5	24	12	60	2	0	12

Table 6.2: Main characteristics of the instances generated starting from the 2012 RAS Competition instances.

6.6.1 Parameter tuning

The objective of parameter tuning is to measure the impact of the sparsification methods and the sorting criteria introduced in Section 6.5, on real-time applications of the algorithm.

We ran six sets of experiments overall, in order to determine which combinations of sparsification and sorting are particularly effective with the RVNS and Tabu policies. For each of these two policies, the three sets of experiments only vary in the hard time limit given to the algorithm. The first two time limits are of 2 and 10 seconds (real-time); the third time limit is of 60 seconds, and is used to provide better solutions that can be used as a baseline for comparisons.

For each set of experiments, the tests were run on all the 23 Alstom instances. For each instance, we tried all combinations of sparsification methods and initial sortings, using the parameters described in Section 6.5. In total, we had 19 possible settings for the sparsification methods and 7 possible sortings, giving 133 tests for each policy, time limit and instance, giving a grand total of $133 \cdot 2 \cdot 3 \cdot 23 = 18354$ runs.

Table 6.3 and Table 6.4 show the results we obtained during parameter tuning for the two solvers, with time limits 2 and 10 seconds. In the first table the results have been aggregated by sparsification method, while in the second, they have been aggregated by sorting criterion.

Columns "Sparsification" (in Table 6.3) or "Sorting" (in Table 6.4) tell, respectively, for which sparsification method or sorting criterion the data is being aggregated. For each line the data are grouped in four blocks, corresponding to the four combinations of policy (Tabu or RVNS) and time limit (2s or 10s). Column "CF" gives the fraction of tests for which the algorithm was able to find a conflict-free schedule. Column "Dev" is the average deviation, calculated as $(z - z^*)/z^*$ where z is the solution value obtained by the algorithm with the specified configuration, and z^* is the best known solution value. This best known value comes from the 60 seconds runs that we use as baseline, and is the best value across all possible parameter combinations. Since, as explained in Section 6.5, the objective function has a hierarchical structure and unresolved conflicts take a very large penalty, the values in this column tend to be quite large, as one single instance for which a method was not able to produce a conflict-free schedule can increase the average considerably. This is, however, a good metric of the desirability of a method, because solving conflicts always has priority on any other measure of solution quality. In column "CF Dev", we similarly report the average deviation, but this time we only consider the conflict-free solutions in the average.

We want to select the best sparsification method for each policy (RVNS or Tabu) and each time limit (2s or 10s). In order to do this, it is not sufficient to take the policy with the lowest deviation, but one has to ensure that the differences in deviation are statistically relevant.

For this reason, we ran a Wilcoxon signed-rank test on each pair of sparsification methods, to measure whether their deviations across the various instances come from the same distribution or not (in this latter case, a difference in the average deviation is statistically relevant).

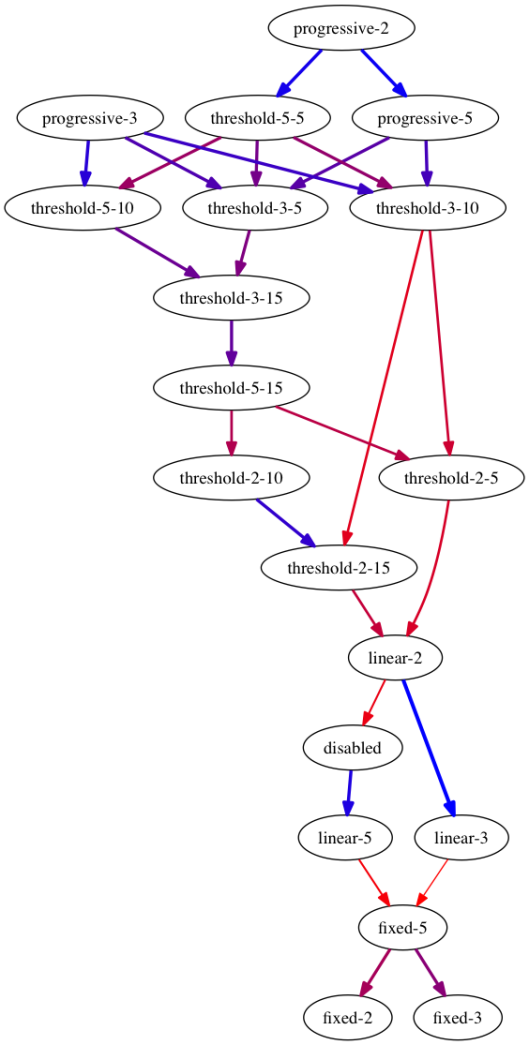
Figure 6.9 and Figure 6.10 give a graphical representation of the outcomes of the Wilcoxon test. The sparsification methods are represented by nodes of an oriented graph. For each pair of methods, if the p-value of the Wilcoxon test was < 0.05, an arc is created between the respective nodes. The arc goes from the node with the better average deviation, to that

	Tabu 2s			Tabu 10s			RVNS 2s			RVNS 10s		
Sparsification	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**
disabled	0.64	10912.64	1.88	0.78	593.90	1.33	0.79	488.43	10.32	0.80	370.22	1.43
fixed-2	0.70	16252.12	1.68	0.77	15374.42	1.35	0.75	15811.52	1.73	0.75	15558.96	1.38
fixed-3	0.71	16161.95	1.66	0.78	15312.86	1.34	0.75	15686.94	1.62	0.76	15528.27	1.42
fixed-5	0.74	15875.32	1.62	0.78	15321.86	1.32	0.74	28820.40	1.59	0.75	26406.56	1.40
linear-2	0.65	6391.54	1.85	0.80	339.41	1.36	0.80	377.08	10.06	0.80	370.23	1.45
linear-3	0.64	8725.04	2.06	0.78	418.67	1.39	0.79	599.52	10.20	0.80	370.24	1.46
linear-5	0.62	11325.47	1.95	0.77	739.40	1.43	0.78	4981.04	8.29	0.80	370.25	1.47
progressive-2	0.76	834.95	1.49	0.82	31.93	1.24	0.78	696.63	1.51	0.80	308.66	1.34
progressive-3	0.73	1348.76	1.41	0.81	62.66	1.23	0.80	493.27	1.55	0.80	247.19	1.34
progressive-5	0.73	1439.36	1.50	0.82	64.21	1.23	0.78	663.47	1.57	0.78	379.93	1.32
threshold-2-5	0.66	3977.49	4.29	0.80	217.94	1.26	0.80	406.13	7.95	0.80	370.16	1.38
threshold-2-10	0.67	3851.51	4.25	0.79	247.15	1.27	0.78	445.13	8.05	0.79	339.43	1.38
threshold-2-15	0.66	4157.31	4.34	0.79	310.16	1.27	0.80	406.14	7.95	0.80	370.17	1.38
threshold-3-5	0.70	1482.97	1.62	0.80	247.14	1.28	0.79	568.01	7.93	0.80	370.15	1.36
threshold-3-10	0.69	1627.55	1.63	0.80	247.15	1.29	0.78	475.82	7.99	0.79	400.89	1.36
threshold-3-15	0.68	1742.63	4.20	0.80	187.18	1.24	0.79	436.84	7.95	0.79	370.16	1.36
threshold-5-5	0.73	1333.08	1.49	0.81	93.41	1.22	0.79	556.38	3.63	0.79	308.66	1.34
threshold-5-10	0.70	1708.34	1.56	0.81	93.42	1.24	0.79	525.67	3.67	0.79	370.14	1.34
threshold-5-15	0.69	1750.12	1.58	0.81	62.70	1.27	0.79	525.69	3.69	0.79	400.89	1.36

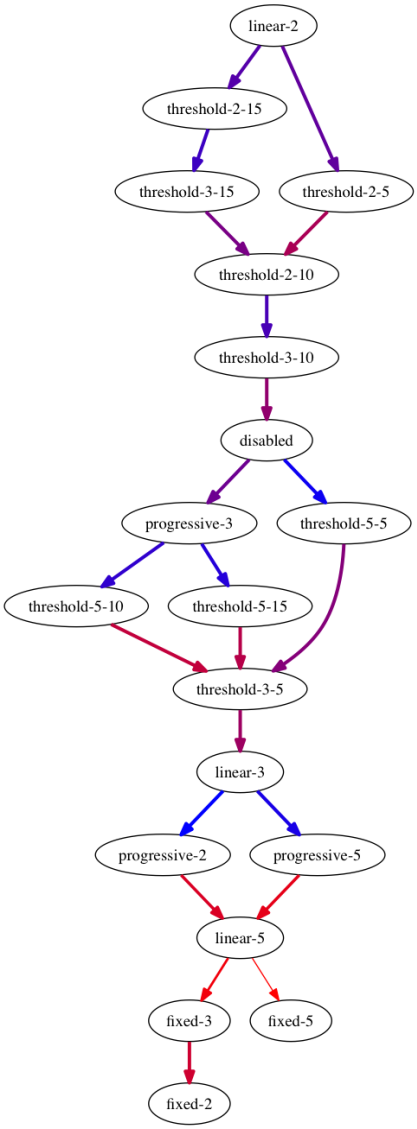
Table 6.3: Parameter tuning results aggregated by sparsification method.

	Tabu 2s			Tabu 10s			RVNS 2s			RVNS 10s		
Sorting	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**	**CF**	**Dev**	**CF Dev**
Congestion	0.73	3197.61	1.58	0.80	2509.98	1.29	0.79	3535.41	11.38	0.80	3349.47	1.42
Conflict time	0.73	7938.66	4.92	0.80	2589.95	1.28	0.78	4012.83	9.14	0.78	3700.91	1.31
Length	0.73	3302.75	1.64	0.80	2635.49	1.27	0.78	3882.08	1.75	0.78	3791.12	1.44
Random	0.59	13031.87	2.01	0.81	2637.88	1.34	0.76	5482.95	9.27	0.78	2623.51	1.45
Reverse congestion	0.74	5170.65	1.72	0.78	2589.07	1.29	0.78	3848.59	1.77	0.78	3756.94	1.37
Reverse speed	0.65	4110.60	1.61	0.79	2584.59	1.26	0.77	2890.27	1.75	0.77	2841.63	1.35
Speed	0.68	4105.06	1.70	0.79	2861.43	1.31	0.82	3229.39	4.74	0.82	3224.74	1.33

Table 6.4: Parameter tuning results aggregated by initial sorting.

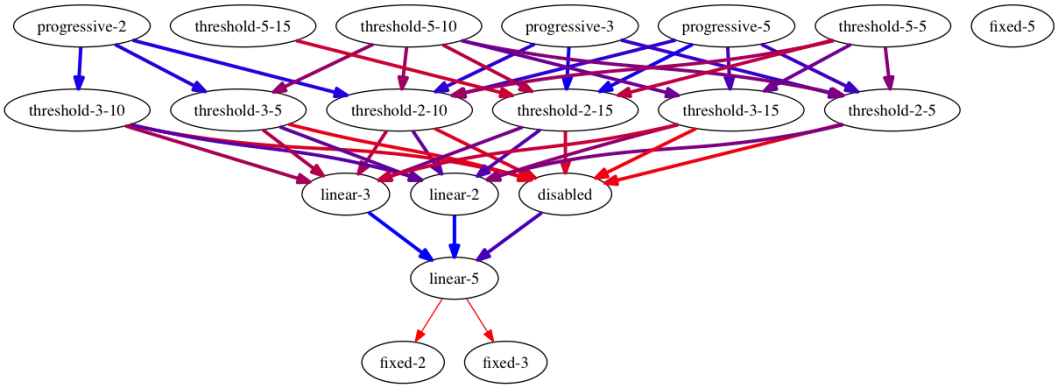


(a) Policy: Tabu, time limit: 2s.

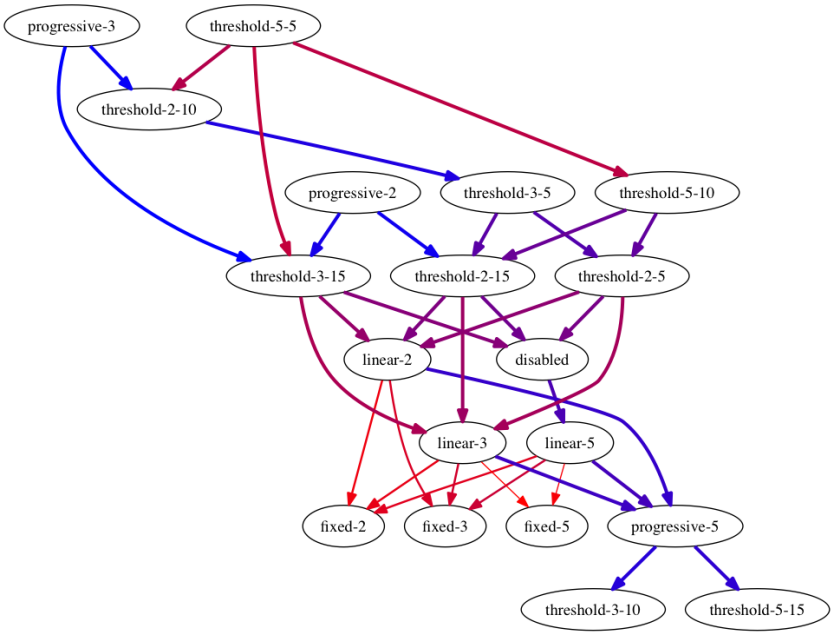


(b) Policy: RVNS, time limit: 2s.

Figure 6.9: Visual representation of the results of the Wilcoxon test (time limit: 2s).



(a) Policy: Tabu, time limit: 10s.



(b) Policy: RVNS, time limit: 10s.

Figure 6.10: Visual representation of the results of the Wilcoxon test (time limit: 10s).

	Tabu 2s progressive-2			Tabu 10s progressive-2			RVNS 2s linear-2			RVNS 10s progressive-3		
	Deviation			Deviation			Deviation			Deviation		
Sorting	Avg	Std	Best	Avg	Std	Best	Avg	Std	Best	Avg	Std	Best
Congestion	441.45	2115.65	8	0.16	0.22	12	11.83	53.76	6	0.36	0.83	8
Conflict time	430.66	2063.77	2	0.22	0.30	2	442.05	2062.05	6	430.57	2063.81	5
Length	1076.20	5159.38	4	0.24	0.38	2	861.38	4127.40	2	215.53	1031.94	1
Random	2461.98	9855.31	3	0.22	0.31	2	227.90	1030.85	1	430.60	2063.80	3
Rev. Congestion	226.44	1083.85	3	0.14	0.16	2	646.11	3095.51	3	430.59	2063.73	1
Rev. Speed	554.88	1887.01	1	0.19	0.33	2	431.11	2063.69	3	215.47	1032.00	1
Speed	646.03	3095.86	2	215.37	1031.95	3	12.15	53.70	2	0.23	0.42	4

Table 6.5: Average deviations of different sortings, for the chosen sparsification methods.

with the worse one. To simplify the graph, whenever there are arc $(M_1, M_2), (M_2, M_3)$ and (M_1, M_3), this latter arc is removed.

The colour and the thickness of the arc depend on the difference of the deviations: the greater the difference, the thicker and more blue the arc; on the other hand, small differences are represented by thin red-ish arcs. Arcs are drawn from top to bottom, so that the best sparsifications are in the top part of the graph.

The sparisification methods were chosen as follows:

- In the case of policy Tabu and time limit 2s (Figure 6.9a) the only two undominated methods are "progressive-2" and "progressive-3"; since the average deviation of the former is 38.1% smaller than that of the latter (see Table 6.3), we decided to take "progressive-2" as the chosen sparsification method.
- For policy RVNS and time limit 2s, the only undominated method is "linear-2", which also has a considerably smaller deviation compared to the other methods.
- An interesting case is that of policy Tabu and time limit 10s (Figure 6.10a), as this is the case where it is most unclear which sparsification emerges as a winner. However, since the deviation of method "progressive-2" is the smallest, and it is 49.04% smaller than the second-smallest one ("progressive-3"), we chose this method.
- Finally, for policy RVNS and time limit 10s, the three undominated methods were "progressive-2", "progressive-3", and "threshold-5-5". Again, since the average deviation for "progressive-3" is 19.92% smaller than that for the other two (which have the same deviation), we chose that method.

Table 6.5 shows the chosen sparsification methods and gives the average deviations (columns "Avg"), together with their standard deviation (column "Std"), obtained by employing the different initial sortings with each sparsification. Column "Best" tells the number of instances (out of 23) for which each sorting criterion provided the best result, compared to the other criteria in the same column.

Tabu 2s progressive-2	**Tabu 10s progressive-2**	**RVNS 2s linear-2**	**RVNS 10s progressive-3**
Congestion	Congestion	Congestion	Congestion
Length	Conflict Time	Conflict Time	Conflict Time
Reverse Congestion	Reverse Congestion	Reverse Congestion	Speed
Random	Reverse Speed	Reverse Speed	Random

Table 6.6: List of sorting criteria chosen for each policy and time limit, to be used in the parallel algorithm.

6.6.2 Parallel algorithm

In this section we provide computational results for a very simple parallel implementation of the algorithm. We ran four sets of experiments, namely one for each combination of policy and time limit, together with the respective sparsification method chosen during parameter tuning, as described in Section 6.6.1. For each set, the parallel implementation simply consists of launching four parallel instances of the algorithm, each using one of four sorting criteria. When the time limit hits, the four solutions provided by the parallel instances are examined and the best one is returned as the overall solution.

The usage of parallel algorithms in operational research is well-established. We refer the reader to, e.g., Clausen and Perregaard [9] for parallel strategies for branch-and-bound exact algorithms, or to Ropke and Santini [37] for a systematic analysis of the speed-ups obtained by parallelising the Adaptive Large Neighbourhood Search metaheuristic. With respect to the train rescheduling problem, Iqbal et al. [25, 26] proposed parallel algorithms for rescheduling under disturbances.

In our case, the implementation of a parallel algorithm is motivated by the high dispersion of the solution values with respect to the average one, obtained by the different sorting methods, as witnessed by the high values of Standard Deviation reported in Table 6.5 (this effect is more evident for the 2s time limit compared to the 10s one, and for the RVNS policy compared to the Tabu one). This means that, in practice, even the best sorting method was not able to resolve some solvable conflict, thus resulting in high solution values, due to the hierarchical nature of our objective function. Furthermore, we observed a certain complementarity in the capability of resolving conflicts across different sortings, which we see as a hint towards the parallel use of different sorting criteria.

More formally, we investigated the dependance of sorting criteria to instance characteristics via simple *one-vs-all* and *one-vs-one* multiclass classification algorithms provided by the library `scikit` (see, e.g., Aly [2]). These algorithms were based on the binary classifier that, for a fixed time limit, assigns an instance to an intial sorting criterion (class) if there is at least one policy (either Tabu or RVNS) for which that sorting provided the best result for that time limit. The instance features considered were the one listed in Tables 6.1 and 6.2. Neither the *one-vs-all* nor the *one-vs-one* algorithm found statistically significant relationships of the sorting criteria to the instance features.

Once established the need for an algorithm that employs more than one sorting criteria at once, it is clearly important to perform a good choice of the criteria. The simplest approach

Instance group	Algorithm	Sorting	Dev	Conflicts	Infeasible	Modified	Delay
L	Tabu 2s	Random, Length, Reverse congestion	0.17	0.50	0.00	104.50	28.64
L	Tabu 10s	Reverse congestion	0.11	0.50	0.00	104.67	29.00
L	RVNS 2s	Reverse speed	0.17	0.50	0.00	105.67	33.34
L	RVNS 10s	Conflict time	0.05	0.50	0.00	100.83	10.64
N	Tabu 2s	Congestion	0.16	0.38	0.08	7.85	79.43
N	Tabu 10s	Congestion	0.08	0.38	0.08	8.23	70.12
N	RVNS 2s	Congestion	0.45	0.38	0.08	7.54	99.30
N	RVNS 10s	Congestion	0.26	0.38	0.08	8.00	67.69
P	Tabu 2s	Reverse congestion	0.50	0.00	0.00	59.25	247.17
P	Tabu 10s	Reverse speed	0.03	0.00	0.00	59.50	220.52
P	RVNS 2s	Conflict time	0.30	0.00	0.00	59.50	226.83
P	RVNS 10s	Congestion	0.20	0.00	0.00	59.50	221.58
Overall 2s		**Congestion**	**0.29**	**0.35**	**0.04**	**42.09**	**99.81**
Overall 10s		**Congestion**	**0.14**	**0.35**	**0.04**	**41.74**	**82.56**
Overall Tabu		**Congestion**	**0.15**	**0.35**	**0.04**	**42.15**	**90.45**
Overall RVNS		**Congestion**	**0.27**	**0.35**	**0.04**	**41.67**	**91.92**
Overall		**Congestion**	**0.21**	**0.35**	**0.04**	**41.91**	**91.19**

Table 6.7: Parallel algorithm results on the Alstom instances.

would be to choose the four criteria that produce the four lowest deviations for a given policy and time limit (see Table 6.5). This choice, however, has not proven particularly good especially for the lowest time limit, and in one case we even had one instance (instance "P4" for the Tabu policy at 2s) for which not all solvable conflicts were actually resolved.

What we aim for is a choice of methods out of which, given any instance, there are high chances to find one that works reasonably well with that instance, eliminating all solvable conflicts, and therefore exploting the aforementioned complementarity. For this reason, we decided to choose the methods in a way to maximise the number of instances for which at least one of the methods chosen was the best. Table 6.6 lists the chosen sorting criterian for each policy and time limit.

Table 6.7 reports aggregate results for the parallel algorithm on the Alstom instances. Column "Sorting" reports which initial sorting produced the optimal solution most often, for a fixed choice of instance group and algorithm. Column "Dev" gives the average deviation, calculated as $(z-z^*)/z^*$ where z is the solution value obtained by the algorithm, and z^* is the best known solution value. Column "Conflicts" lists the average number of conflicts remaining in the output solution, while columns "Infeasible" and "Modified" report, respectively, the number of trains that are infeasible and whose schedule has been modified in the output solution. Finally, "Delay" lists the average delay (or advance, in which case the figure is < 0) reported by trains at their final destination.

By observing the tables, it is clear that the benefit of the parallel algorithm is considerable when compared to fixed choices of sorting criteria. This is particularly evident for the 2s time limit, where the best average deviations achieved by using only one sorting (see Table 6.5) were of 226.44 for Tabu and 11.83 for RVNS, while the parallel algorithms gives — respectively — 0.22 and 0.35 (see the detailed results in Tables 6.9 and 6.10). In addition, by looking at the detailed results, we can notice that all four sortings selected for each set

Instance group	Algorithm	Sorting	Dev	Conflicts	Infeasible	Modified	Delay
S-1	Tabu 2s	Reverse congestion	$2.26 \cdot 10^{-2}$	0	0	4.60	132.67
S-1	Tabu 10s	Reverse congestion	$1.99 \cdot 10^{-2}$	0	0	4.60	131.72
S-1	RVNS 2s	Reverse speed	$3.58 \cdot 10^{-2}$	0	0	4.60	134.77
S-1	RVNS 10s	Length, Speed	$1.64 \cdot 10^{-2}$	0	0	4.60	129.30
S-2	Tabu 2s	Reverse congestion	$3.97 \cdot 10^{-3}$	0	0	3.80	63.77
S-2	Tabu 10s	Congestion	$1.83 \cdot 10^{-3}$	0	0	3.80	60.61
S-2	RVNS 2s	Conflict time	$6.75 \cdot 10^{-4}$	0	0	3.80	60.04
S-2	RVNS 10s	Speed	$3.54 \cdot 10^{-4}$	0	0	3.80	59.80
S-3	Tabu 2s	Length, Congestion	$1.24 \cdot 10^{-2}$	0	0	5.20	185.33
S-3	Tabu 10s	Congestion	$2.65 \cdot 10^{-3}$	0	0	5.00	188.68
S-3	RVNS 2s	Conflict time	$7.52 \cdot 10^{-3}$	0	0	5.20	190.85
S-3	RVNS 10s	Congestion, Speed	$8.02 \cdot 10^{-5}$	0	0	5.20	186.63
N-1	Tabu 2s	Random, Congestion	$6.91 \cdot 10^{-4}$	0	0	4.20	125.05
N-1	Tabu 10s	Reverse congestion	$6.91 \cdot 10^{-4}$	0	0	4.20	125.05
N-1	RVNS 2s	Conflict time	$1.19 \cdot 10^{-1}$	0	0	3.80	154.79
N-1	RVNS 10s	Congestion	$2.25 \cdot 10^{-4}$	0	0	4.20	124.86
N-2	Tabu 2s	Length	$9.90 \cdot 10^{-4}$	0	0	4.00	106.31
N-2	Tabu 10s	Conflict time	$9.90 \cdot 10^{-4}$	0	0	4.00	106.31
N-2	RVNS 2s	Conflict time	$7.52 \cdot 10^{-4}$	0	0	4.00	106.42
N-2	RVNS 10s	Length	$1.32 \cdot 10^{-4}$	0	0	4.00	106.06
N-3	Tabu 2s	Reverse congestion	$2.15 \cdot 10^{-2}$	0	0	6.00	219.10
N-3	Tabu 10s	Conflict time	$7.14 \cdot 10^{-3}$	0	0	5.40	203.28
N-3	RVNS 2s	Conflict time	$9.28 \cdot 10^{-3}$	0	0	5.20	205.11
N-3	RVNS 10s	Speed	$7.40 \cdot 10^{-3}$	0	0	5.20	207.49
Overall 2s		**Conflict time**	$\mathbf{1.93 \cdot 10^{-2}}$	**0**	**0**	**4.20**	**140.27**
Overall 10s		**Congestion**	$\mathbf{5.07 \cdot 10^{-3}}$	**0**	**0**	**4.23**	**135.90**
Overall Tabu		**Reverse congestion**	$\mathbf{7.05 \cdot 10^{-3}}$	**0**	**0**	**4.30**	**137.32**
Overall RVNS		**Conflict time**	$\mathbf{1.64 \cdot 10^{-2}}$	**0**	**0**	**4.20**	**138.84**
Overall		**Conflict time**	$\mathbf{1.22 \cdot 10^{-2}}$	**0**	**0**	**4.25**	**138.08**

Table 6.8: Parallel algorithm results on the RAS-Based instances.

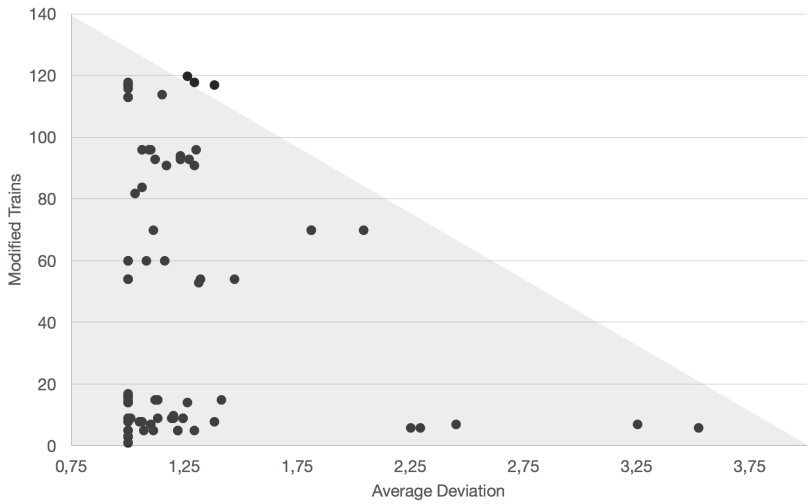


Figure 6.11: Scatter graph (without outliers) showing the correlation between solution quality and number of modified trains.

(algorithm and time limit) provide the best solution in some instances, with the "Congestion" sorting being the one appearing most frequently overall, and also when aggregating by policy or by time limit.

For what concerns the choice of the heuristic policy, from the detailed results we can notice that Tabu and RVNS turn out to provide the smallest instance-by-instance deviations almost the same number of times: namely, both 15 times for 2s, and 18 vs 17 times for 10s. From Table 6.7 we can see, however, that Tabu provides smaller deviations, at the cost of modifying more trains. A small further reduction could be achieved by an hypothetical algorithm that ran the Tabu and RVNS policies in parallel (thereby using 8 concurrent threads): such an algorithm would achieve an average deviation of 0.17 for 2s, and 0.06 for 10s. Finally, as we clearly expected, a considerable improvement is obtained by letting the algorithm run for longer: the average deviation for all methods ran for 10s is less than half than that for all methods ran for 2s.

Figure 6.11 displays the relation between the deviations achieved by the parallel algorithm and the number of trains whose schedules have been modified. All the solutions described in Table 6.9 and Table 6.10 are reported in the figure. The figure seems to suggest that, despite not having included the number of modified trains as a penalty term in the objective funciont (see Section 6.4.4) there are no solutions in which a lot of trains are modified and, despite that, bad solutions are obtained. This can be seen by noticing that the upper-right triangle of the graph is empty. In summary, the figure seem to suggest that three scenarios can happen. The first, best scenario is that a high quality solution is found and few trains are modified (bottom-left cluster of points); in the second scenario a high quality solution is found, but a lot of trains have to be modified (points on the top-left corner); finally, rarely a bad solution is found, but in this case only few trains are modified (bottom-right points).

Table 6.8 is analogous to Table 6.7 and reports aggregate results for the parallel algorithm

on the RAS-based instances. Notice that, due to the nature of the instances, the deviation are smaller compared to those reported in Table 6.7, and all conflicts were resolved in each instance. Also, the average number of modified trains is smaller for RAS instances than for Alstom instances; this is not surprising, as the number of train in the nominal timetable was also smaller. Average delays are, on the other hand, higher, probably due to the fact that the time horizon is longer for the RAS-based instances.

As in the case of the Alstom instances, there is a lot of variability in which initial sorting criterion leads to the best solution; not only some criterion works best with particular instances, but also the combination of instance and policy seems to influence the effectiveness of the sorting criterion. This confirms the negative results obtained by the classification algorithms, and the potential of a (simple) parallel implementation in order to obtain good practical results.

6.7 Conclusions

In this paper we have presented a fast algorithm for resolving conflicts in real-time train timetabling. The algorithm is capable of handling several constraints that arise in real-world applications. The underlying model, based on a time-space graph, is quite flexible since it supports both micro- and macroscopic modelling, and even a mix of the two. Computational results, conducted on instances representing real-world scenarios, show that the model can resolve all solvable conflicts in very short computing times, which are compatible with a real-time context. An industrial implementation of the presented algorithm has been integrated in the ICONIS system of Alstom and will be deployed in several real-world contexts.

The model could be further expanded to take into consideration, e.g., the energy efficiency of the generated schedules, or their robustness with respect to future disturbances. Another interesting research avenue concerns the development of relaxations of the problem, based on (mixed integer) linear programming, that could provide lower bounds on the objective function value.

Acknowledgements

The authors are grateful to Alstom Transport for providing the test instances used for the computational validation of our method.

Appendix: Detailed results

Tables 6.9 and 6.10 provide instance-by-instance results for Tabu and RVNS policies, respectively. Column “Sorting” specifies which was the sorting method employed in the thread that produced the overall best solution. Column “Dev”, analogously to what presented in previous tables, is the ratio between the objective value produced by the parallel algorithm and the best known objective value for the same instance. Column “Conflicts” lists the number of hard and soft constraints violated in the produced solution. Column “Inf Trains” tells how

many trains remain infeasible (i.e., with hard constraints violations). Column “Mod Trains” gives the number of trains whose schedules have been modified. Finally, column “Avg Delay” lists, in time units, the average delay (if > 0) or advance (if < 0) that a train reported when arriving at its destination. A † next to a deviation indicates the best deviation produced for the corresponding instance, for that time limit. When Tabu and RVNS attained the same deviation, the † is present in both tables.

Notice that there are instances for which no method was able to resolve all Conflict timelicts (even when using a time limit of 60s). Manual inspection of these Conflict timelicts has Conflict timeirmed that they are, indeed, unavoidable.

Tables 6.11 and 6.12 are analogous to Tables 6.9 and 6.10, but provide detailed results relative to the RAS-based instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 2s progressive-2	N1	Length	0.20	0	0	9	94.14
	N2	Length	†0.13	0	0	15	198.53
	N3	Reverse congestion	†0.00	3	1	8	40.71
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Random	†0.00	0	0	16	102.63
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Length	†0.05	0	0	8	70.34
	N8	Congestion	†0.24	0	0	9	3.87
	N9	Congestion	†1.25	0	0	6	32.07
	N10	Congestion	0.22	0	0	5	9.19
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	408.53
	L1	Random	†0.00	1	0	117	-13.71
	L2	Reverse congestion	†0.10	0	0	96	-39.17
	L3	Random	†0.00	2	0	113	87.02
	L4	Reverse congestion	0.29	0	0	91	33.66
	L5	Length	0.38	0	0	117	59.70
	L6	Length	0.27	0	0	93	44.33
	P1	Reverse congestion	0.47	0	0	54	243.21
	P2	Reverse congestion	†0.31	0	0	53	256.07
	P3	Congestion	0.16	0	0	60	141.05
	P4	Random	1.04	0	0	70	348.33
	Overall		0.22	0.35	0.04	42.00	95.35

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 10s progressive-2	N1	Reverse speed	0.20	0	0	10	95.17
	N2	Conflict time	†0.00	0	0	15	192.35
	N3	Congestion	†0.00	3	1	9	49.29
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Congestion	†0.00	0	0	16	102.63
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Congestion	†0.01	0	0	9	70.34
	N8	Congestion	†0.24	0	0	9	3.87
	N9	Congestion	†0.38	0	0	8	10.34
	N10	Congestion	†0.22	0	0	5	9.19
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	305.74
	L1	Reverse congestion	†0.00	1	0	117	-23.14
	L2	Reverse congestion	0.09	0	0	96	-42.23
	L3	Conflict time	†0.00	2	0	113	106.49
	L4	Reverse congestion	0.17	0	0	91	28.85
	L5	Reverse speed	0.29	0	0	118	73.71
	L6	Congestion	0.12	0	0	93	30.34
	P1	Reverse speed	†0.00	0	0	54	233.84
	P2	Conflict time	†0.00	0	0	54	236.25
	P3	Congestion	†0.00	0	0	60	138.87
	P4	Reverse speed	†0.11	0	0	70	273.12
	Overall		0.08	0.35	0.04	42.30	85.55

Table 6.9: Parallel algorithm results for the Tabu solver on the Alstom instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 2s linear-2	N1	Reverse congestion	†0.19	0	0	9	94.14
	N2	Reverse congestion	0.26	0	0	14	229.41
	N3	Reverse congestion	†0.00	3	1	9	48.75
	N4	Reverse congestion	0.07	0	0	5	15.00
	N5	Congestion	0.41	0	0	15	273.95
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Reverse speed	0.10	0	0	7	88.45
	N8	Conflict time	0.52	0	0	6	49.35
	N9	Conflict time	0.25	0	0	7	20.69
	N10	Conflict time	†0.11	0	0	5	6.77
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Congestion	†0.00	2	0	14	406.76
	L1	Conflict time	†0.00	1	0	118	-5.25
	L2	Conflict time	0.30	0	0	96	-17.48
	L3	Reverse speed	†0.00	2	0	113	81.07
	L4	Reverse congestion	†0.23	0	0	93	36.87
	L5	Reverse speed	†0.26	0	0	120	49.17
	L6	Reverse speed	†0.23	0	0	94	55.63
	P1	Congestion	†0.00	0	0	54	235.71
	P2	Conflict time	0.32	0	0	54	236.25
	P3	Conflict time	†0.08	0	0	60	150.97
	P4	Congestion	†0.81	0	0	70	284.38
	Overall		0.35	0.35	0.04	42.17	104.27

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 10s progressive-3	N1	Random	†0.13	0	0	9	97.76
	N2	Random	0.12	0	0	15	198.53
	N3	Speed	†0.00	3	1	9	47.68
	N4	Congestion	†0.00	0	0	5	15.00
	N5	Conflict time	†0.00	0	0	16	101.05
	N6	Congestion	†0.00	0	0	1	8.33
	N7	Conflict time	0.06	0	0	8	88.45
	N8	Speed	1.45	0	0	7	28.55
	N9	Speed	1.29	0	0	6	33.62
	N10	Congestion	0.29	0	0	5	9.68
	N11	Congestion	†0.00	0	0	3	19.29
	N12	Congestion	†0.00	0	0	3	30.00
	N13	Random	†0.00	2	0	17	202.06
	L1	Conflict time	†0.00	1	0	116	-21.96
	L2	Congestion	†0.06	0	0	96	-46.75
	L3	Conflict time	†0.00	2	0	113	60.34
	L4	Conflict time	†0.03	0	0	82	19.01
	L5	Congestion	†0.15	0	0	114	14.21
	L6	Speed	†0.06	0	0	84	38.96
	P1	Congestion	†0.00	0	0	54	232.50
	P2	Conflict time	†0.00	0	0	54	234.91
	P3	Random	†0.00	0	0	60	134.52
	P4	Congestion	0.81	0	0	70	284.38
	Overall		0.19	0.35	0.04	41.17	79.57

Table 6.10: Parallel algorithm results for the RVNS solver on the Alstom instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 2s progressive-2	S-1-1	Length	$4.10 \cdot 10^{-2}$	0	0	5	200.20
	S-1-2	Reverse congestion	†0	0	0	4	109.23
	S-1-3	Length	$3.32 \cdot 10^{-2}$	0	0	5	78.11
	S-1-4	Reverse congestion	$2.55 \cdot 10^{-2}$	0	0	5	186.38
	S-1-5	Reverse congestion	†0	0	0	4	84.69
	S-2-1	Reverse congestion	$3.37 \cdot 10^{-3}$	0	0	2	791
	S-2-2	Reverse congestion	$3.44 \cdot 10^{-3}$	0	0	2	48.95
	S-2-3	Reverse congestion	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Reverse congestion	$1.16 \cdot 10^{-2}$	0	0	3	94.20
	S-2-5	Reverse congestion	$7.11 \cdot 10^{-4}$	0	0	2	53.97
	S-3-1	Reverse congestion	$1.99 \cdot 10^{-4}$	0	0	12	604.70
	S-3-2	Length	$2.41 \cdot 10^{-4}$	0	0	1	16.96
	S-3-3	Length	†0	0	0	2	88.69
	S-3-4	Congestion	$1.28 \cdot 10^{-2}$	0	0	7	95.06
	S-3-5	Congestion	$4.89 \cdot 10^{-2}$	0	0	4	121.25
	N-1-1	Random	$2.24 \cdot 10^{-4}$	0	0	5	168.34
	N-1-2	Length	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Congestion	$4.60 \cdot 10^{-4}$	0	0	3	82.44
	N-1-4	Random	$1.16 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Congestion	$4.85 \cdot 10^{-4}$	0	0	5	96.89
	N-2-1	Random	$2.22 \cdot 10^{-3}$	0	0	2	69.46
	N-2-2	Length	†0	0	0	4	97.16
	N-2-3	Length	$1.82 \cdot 10^{-3}$	0	0	3	67.38
	N-2-4	Random	†0	0	0	4	153.92
	N-2-5	Length	$9.13 \cdot 10^{-4}$	0	0	7	143.64
	N-3-1	Reverse congestion	$2.11 \cdot 10^{-2}$	0	0	13	690.42
	N-3-2	Reverse congestion	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Reverse congestion	†0	0	0	2	106.09
	N-3-4	Congestion	$8.59 \cdot 10^{-2}$	0	0	10	162.78
	N-3-5	Random	$3.64 \cdot 10^{-4}$	0	0	4	119.06
	Overall		$9.92 \cdot 10^{-3}$	0	0	4.37	138.55

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
Tabu 10s progressive-2	S-1-1	Reverse congestion	$5.62 \cdot 10^{-2}$	0	0	5	205.71
	S-1-2	Reverse congestion	†0	0	0	4	109.23
	S-1-3	Reverse congestion	$3.32 \cdot 10^{-2}$	0	0	5	78.11
	S-1-4	Congestion	$2.38 \cdot 10^{-2}$	0	0	5	185.61
	S-1-5	Reverse speed	†0	0	0	4	84.69
	S-2-1	Reverse speed	$3.37 \cdot 10^{-3}$	0	0	2	70.91
	S-2-2	Congestion	$3.44 \cdot 10^{-3}$	0	0	2	48.95
	S-2-3	Congestion	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Congestion	$9.02 \cdot 10^{-4}$	0	0	3	78.43
	S-2-5	Congestion	$7.11 \cdot 10^{-4}$	0	0	2	53.97
	S-3-1	Reverse speed	$1.99 \cdot 10^{-4}$	0	0	12	604.70
	S-3-2	Congestion	$2.41 \cdot 10^{-4}$	0	0	1	16.96
	S-3-3	Congestion	†0	0	0	2	117.50
	S-3-4	Congestion	$1.28 \cdot 10^{-2}$	0	0	7	95.06
	S-3-5	Congestion	†0	0	0	3	109.17
	N-1-1	Reverse congestion	$2.24 \cdot 10^{-4}$	0	0	5	168.34
	N-1-2	Reverse congestion	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Reverse congestion	$4.60 \cdot 10^{-4}$	0	0	3	82.44
	N-1-4	Conflict time	$1.16 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Conflict time	$4.85 \cdot 10^{-4}$	0	0	5	96.89
	N-2-1	Conflict time	$2.22 \cdot 10^{-3}$	0	0	2	69.46
	N-2-2	Conflict time	†0	0	0	4	97.16
	N-2-3	Conflict time	$1.82 \cdot 10^{-3}$	0	0	3	67.38
	N-2-4	Conflict time	0	0	0	4	153.92
	N-2-5	Conflict time	†$9.13 \cdot 10^{-4}$	0	0	7	143.64
	N-3-1	Reverse speed	$4.53 \cdot 10^{-4}$	0	0	12	657.50
	N-3-2	Conflict time	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Conflict time	†0	0	0	2	106.09
	N-3-4	Reverse speed	$3.47 \cdot 10^{-2}$	0	0	8	116.63
	N-3-5	Conflict time	$3.64 \cdot 10^{-4}$	0	0	4	119.06
	Overall		$5.99 \cdot 10^{-3}$	0	0	4.23	136.10

Table 6.11: Parallel algorithm results for the Tabu solver on the RAS-based instances.

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 2s linear-2	S-1-1	Reverse speed	$5.51 \cdot 10^{-2}$	0	0	5	204.39
	S-1-2	Conflict time	$5.37 \cdot 10^{-2}$	0	0	4	117.14
	S-1-3	Reverse speed	†$2.04 \cdot 10^{-2}$	0	0	5	73.57
	S-1-4	Reverse speed	$3.69 \cdot 10^{-2}$	0	0	5	190.15
	S-1-5	Conflict time	$1.28 \cdot 10^{-2}$	0	0	4	88.57
	S-2-1	Conflict time	†0	0	0	2	70.05
	S-2-2	Conflict time	†0	0	0	2	47.51
	S-2-3	Conflict time	$7.26 \cdot 10^{-4}$	0	0	2	50.81
	S-2-4	Conflict time	$9.02 \cdot 10^{-4}$	0	0	3	78.43
	S-2-5	Conflict time	$1.42 \cdot 10^{-4}$	0	0	2	53.40
	S-3-1	Reverse speed	†0	0	0	12	604.23
	S-3-2	Conflict time	$7.22 \cdot 10^{-4}$	0	0	1	17.20
	S-3-3	Conflict time	†0	0	0	2	117.50
	S-3-4	Conflict time	$3.57 \cdot 10^{-2}$	0	0	8	106.61
	S-3-5	Conflict time	$1.17 \cdot 10^{-3}$	0	0	3	108.69
	N-1-1	Conflict time	$5.81 \cdot 10^{-1}$	0	0	3	314.07
	N-1-2	Congestion	†0	0	0	4	106.27
	N-1-3	Conflict time	$5.06 \cdot 10^{-3}$	0	0	3	84.08
	N-1-4	Conflict time	$1.45 \cdot 10^{-3}$	0	0	4	171.13
	N-1-5	Conflict time	$5.10 \cdot 10^{-3}$	0	0	5	98.41
	N-2-1	Conflict time	$1.78 \cdot 10^{-3}$	0	0	2	69.36
	N-2-2	Conflict time	$6.33 \cdot 10^{-4}$	0	0	4	97.68
	N-2-3	Conflict time	$3.03 \cdot 10^{-4}$	0	0	3	66.86
	N-2-4	Conflict time	$3.83 \cdot 10^{-4}$	0	0	4	154.45
	N-2-5	Congestion	$6.64 \cdot 10^{-4}$	0	0	7	143.75
	N-3-1	Reverse speed	†0	0	0	12	657.15
	N-3-2	Conflict time	$1.95 \cdot 10^{-4}$	0	0	1	17.13
	N-3-3	Conflict time	†0	0	0	2	106.09
	N-3-4	Conflict time	$4.53 \cdot 10^{-2}$	0	0	7	125.96
	N-3-5	Reverse speed	$9.09 \cdot 10^{-4}$	0	0	4	119.21
	Overall		$2.87 \cdot 10^{-2}$	0	0	4.16	141.99

Test set	Instance	Sorting	Dev	Conflicts	Inf Trains	Mod Trains	Avg Delay
RVNS 10s progressive-3	S-1-1	Congestion	†0	0	0	5	185.71
	S-1-2	Length	$2.90 \cdot 10^{-2}$	0	0	4	113.67
	S-1-3	Speed	$3.22 \cdot 10^{-2}$	0	0	5	77.76
	S-1-4	Length	†$2.05 \cdot 10^{-2}$	0	0	5	184.59
	S-1-5	Speed	$3.02 \cdot 10^{-4}$	0	0	4	84.74
	S-2-1	Speed	$3.37 \cdot 10^{-3}$	0	0	2	70.91
	S-2-2	Speed	†0	0	0	2	47.51
	S-2-3	Speed	†0	0	0	2	50.24
	S-2-4	Speed	†0	0	0	3	77.10
	S-2-5	Speed	†0	0	0	2	53.25
	S-3-1	Length	$1.66 \cdot 10^{-4}$	0	0	12	604.58
	S-3-2	Speed	†0	0	0	1	16.96
	S-3-3	Speed	†0	0	0	2	117.50
	S-3-4	Congestion	†0	0	0	8	85.42
	S-3-5	Congestion	$2.35 \cdot 10^{-4}$	0	0	3	108.69
	N-1-1	Congestion	†0	0	0	5	168.25
	N-1-2	Congestion	$1.13 \cdot 10^{-3}$	0	0	4	106.44
	N-1-3	Congestion	†0	0	0	3	82.27
	N-1-4	Length	†0	0	0	4	170.62
	N-1-5	Length	†0	0	0	5	96.72
	N-2-1	Random	†0	0	0	2	68.94
	N-2-2	Length	$5.06 \cdot 10^{-4}$	0	0	4	97.58
	N-2-3	Length	†0	0	0	3	66.76
	N-2-4	Length	$1.53 \cdot 10^{-4}$	0	0	4	154.13
	N-2-5	Length	†0	0	0	7	142.91
	N-3-1	Length	$2.34 \cdot 10^{-2}$	0	0	12	697.16
	N-3-2	Speed	†0	0	0	1	17.08
	N-3-3	Speed	†0	0	0	2	106.14
	N-3-4	Speed	†$1.36 \cdot 10^{-2}$	0	0	7	98.21
	N-3-5	Speed	†0	0	0	4	118.86
	Overall		$4.15 \cdot 10^{-3}$	0	0	4.23	135.69

Table 6.12: Parallel algorithm results for the RVNS solver on the RAS-based instances.

Bibliography

[1] Rodrigo Acuna-Agost, Philippe Michelon, Dominique Feillet, and Serigne Gueye. A mip-based local search method for the railway rescheduling problem. *Networks*, 57(1): 69–86, 2011.

[2] Mohammed Aly. Survey on multi-class classification methods. Technical report, Caltech, 2015.

[3] Andrea Bettinelli, Alberto Santini, and Daniele Vigo. A real-time conflict solution algorithm for the Train Rescheduling Problem. *Transportation Research, Part B (under revision)*, pages 1–28, 2017.

[4] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: A generic framework for managing hardware affinities in hpc applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180–186. IEEE, 2010.

[5] Valentina Cacchiani, Alberto Caprara, and Paolo Toth. Scheduling extra freight trains on railway networks. *Transportation Research Part B: Methodological*, 44(2):215–231, 2010.

[6] Valentina Cacchiani, Dennis Huisman, Martin Kidd, Leo Kroon, Paolo Toth, Lucas Veelenturf, and Joris Wagenaar. An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B: Methodological*, 63:15–37, 2014.

[7] Gabrio Caimi, Marco Laumanns, Kaspar Schüpbach, Stefan Wörner, and Martin Fuchsberger. The periodic service intention as a conceptual framework for generating timetables with partial periodicity. *Transportation Planning and Technology*, 34(4):323–339, 2011.

[8] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Modeling and solving the train timetabling problem. *Operations research*, 50(5):851–861, 2002.

[9] Jens Clausen and Michael Perregaard. On the best search strategy in parallel branch-and-bound: Best-first search versus lazy depth-first search. *Annals of Operations Research*, 90:1–17, 1999.

[10] Francesco Corman and Lingyun Meng. A review of online dynamic models and algorithms for railway traffic management. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1274–1284, 2015.

[11] Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. Evaluation of green wave policy in real-time railway traffic management. *Transportation Research Part C: Emerging Technologies*, 17(6):607–616, 2009.

[12] Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. Centralized versus distributed systems to reschedule trains in two dispatching areas. *Public Transport*, 2(3):219–247, 2010.

[13] Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A tabu search algorithm for rerouting trains during rail operations. *Transportation Research Part B: Methodological*, 44(1):175–192, 2010.

[14] Francesco Corman, Andrea D'Ariano, Marco Pranzo, and Ingo A Hansen. Effectiveness of dynamic reordering and rerouting of trains in a complicated and densely occupied station area. *Transportation Planning and Technology*, 34(4):341–362, 2011.

[15] Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. Bi-objective conflict detection and resolution in railway traffic management. *Transportation Research Part C: Emerging Technologies*, 20(1):79–94, 2012.

[16] Andrea D'Ariano and Marco Pranzo. An advanced real-time train dispatching system for minimizing the propagation of delays in a dispatching area under severe disturbances. *Networks and Spatial Economics*, 9(1):63–84, 2009.

[17] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643–657, 2007.

[18] Andrea D'Ariano, Francesco Corman, Dario Pacciarelli, and Marco Pranzo. Reordering and local rerouting strategies to manage train traffic in real time. *Transportation Science*, 42(4):405–419, 2008.

[19] Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. Assessment of flexible timetables in real-time traffic management of a railway bottleneck. *Transportation Research Part C: Emerging Technologies*, 16(2):232–245, 2008.

[20] Twan Dollevoet, Dennis Huisman, Leo Kroon, Marie Schmidt, and Anita Schöbel. Delay management including capacities of stations. *Transportation Science*, 49(2):185–203, 2014.

[21] Wei Fang, Shengxiang Yang, and Xin Yao. A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2997–3016, 2015.

[22] IA Hansen and J Pachl. Railway timetabling & operations. *Eurailpress, Hamburg*, 2014.

[23] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José Pérez. Variable neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 61–86. Springer, 2010. ISBN 978-1-4419-1663-1.

[24] Railways Applications Section INFORMS. Problem description and released data set for the ras problem solving competition, 2012. URL https://www.informs.org/Community/RAS/Problem-Solving-Competition/2012-RAS-Problem-Solving-Competition.

[25] Syed Muhammad Zeeshan Iqbal, Håkan Grahn, Törnquist Krasemann, et al. A parallel heuristic for fast train dispatching during railway traffic disturbances: Early results. In *1st International Conference on Operations Research and Enterprise Systems, ICORES*, 2012.

[26] Syed Muhammad Zeeshan Iqbal, Håkan Grahn, and J Törnquist Krasemann. Multi-strategy based train re-scheduling during railway traffic disturbances. In *Proceedings of the 5th International Seminar on Rail Operations Modeling and Analysis (RailCopenhagen 2013, pp. 387-405), Technical University of Denmark, Denmark*, 2013.

[27] Satoshi Kanai, Koichi Shiina, Shingo Harada, and Norio Tomii. An optimal delay management algorithm from passengers' viewpoints considering the whole railway network. *Journal of Rail Transport Planning & Management*, 1(1):25–37, 2011.

[28] Leonardo Lamorgese and Carlo Mannino. The track formulation for the train dispatching problem. *Electronic Notes in Discrete Mathematics*, 41:559 – 566, 2013. doi: http://dx.doi.org/10.1016/j.endm.2013.05.138.

[29] Leonardo Lamorgese and Carlo Mannino. An exact decomposition approach for the real-time train dispatching problem. *Operations Research*, 63(1):48–64, 2015.

[30] Feng Li, Ziyou Gao, Keping Li, and Lixing Yang. Efficient scheduling of railway traffic based on global information of train. *Transportation Research Part B: Methodological*, 42(10):1008–1030, 2008.

[31] Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[32] Lingyun Meng and Xuesong Zhou. Robust single-track train dispatching model under a dynamic and stochastic environment: a scenario-based rolling horizon solution approach. *Transportation Research Part B: Methodological*, 45(7):1080–1102, 2011.

[33] Lingyun Meng and Xuesong Zhou. Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables. *Transportation Research Part B: Methodological*, 67:208–234, 2014.

[34] Shi Mu and Maged Dessouky. Scheduling freight trains traveling on complex networks. *Transportation Research Part B: Methodological*, 45(7):1103–1123, 2011.

[35] Paola Pellegrini, Grégory Marlière, and Joaquin Rodriguez. Optimal train routing and scheduling for managing traffic perturbations in complex junctions. *Transportation Research Part B: Methodological*, 59:58–80, 2014.

[36] Joaquín Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B: Methodological*, 41(2):231–245, 2007.

[37] Stefan Ropke and Alberto Santini. Parallel adaptive large neighbourhood search. Technical Report OR-16-11, DEI University of Bologna, 2016.

[38] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033 – 2049, 2007. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2005.12.009. URL http://www.sciencedirect.com/science/article/pii/S0377221705008507.

[39] Marcella Samà, Paola Pellegrini, Andrea D'Ariano, Joaquin Rodriguez, and Dario Pacciarelli. Ant colony optimization for the real-time train routing selection problem. *Transportation Research Part B: Methodological*, 85:89–108, 2016.

[40] Alberto Santini. cr-ras-derived-instances: Initial release, Feb 2017. URL https://doi.org/10.5281/zenodo.322571.

[41] Michael Schachtebeck and Anita Schöbel. To wait or not to wait, and who goes first? delay management with priority decisions. *Transportation Science*, 44(3):307–321, 2010.

[42] Anita Schöbel. Integer programming approaches for solving the delay management problem. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and ChristosD. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, volume 4359 of *Lecture Notes in Computer Science*, pages 145–170. Springer Berlin Heidelberg, 2007.

[43] Anita Schöbel. Capacity constraints in delay management. *Public Transport*, 1(2):135–154, 2009.

[44] Johanna Törnquist. Design of an effective algorithm for fast response to the rescheduling of railway traffic during disturbances. *Transportation Research Part C: Emerging Technologies*, 20(1):62–78, 2012.

[45] Johanna Törnquist and Jan Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B: Methodological*, 41(3):342–362, 2007.

[46] Johanna Törnquist and Jan A Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B: Methodological*, 41(3):342–362, 2007.

7 Acceptance criteria for ALNS: a benchmark on logistic problems

Abstract Adaptive Large Neighborhood Search (ALNS) is a useful framework for solving difficult combinatorial optimisation problems. As a metaheuristic, it consists of some components that must be tailored to the specific optimisation problem that is being solved, while other components are problem independent. The literature is sparse with respect to studies that aim to evaluate the relative merit of different alternatives for specific problem independent components. This chapter investigates one such component, the move acceptance criterion in ALNS, and compares a range of alternatives. Through extensive computational testing, the alternative move acceptance criteria are ranked in three groups, depending on the performance of the resulting ALNS implementations. Among the best variants, we find versions of criteria based on Simulated Annealing, Threshold Acceptance, and Record-to-Record Travel. Additional analyses focus on the search behavior, and multiple linear regression is used to identify characteristics of search behavior that are associated with good search performance.

7.1 Introduction

The Adaptive Large Neighborhood Search (ALNS) metaheuristic [27] has become a popular template for implementing heuristic solution methods, especially for vehicle routing applications [7, 12, 14, 22, 25]. The metaheuristic allows the use of problem specific knowledge when specifying operators for partially destroying and then repairing a solution to an optimisation problem. Problem independent components of the ALNS dictate how different destroy and repair operators are used and control the search trajectory. One presumably important component that influences the search trajectory is the move acceptance criterion. In the original ALNS, this criterion was based on Simulated Annealing [27], whereas earlier work on Large Neighborhood Search (LNS) by Shaw [31] accepted only improving solutions. Recently, some implementations have used the Record-to-Record Travel acceptance criterion

This chapter is based on the contents of: Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. Measuring the impact of acceptance criteria on the Adaptive Large Neighbourhood Search metaheuristic. *Submitted to the Journal of Heuristics*, pages 1–25, 2017.

instead [20], and in one case it was found to perform better than the standard Simulated Annealing criterion [13].

Currently, however, there are no guidelines available to recommend one acceptance criterion over another. This paper intends to fill this gap by investigating a large number of different move acceptance criteria by subjecting them to extensive computational testing. Through empirical experiments we attempt to 1) quantify the effect on performance from using different acceptance criteria, 2) suggest which move acceptance criterion is better suited for an implementation of ALNS, and 3) attempt to measure in which way the move acceptance criteria influence the search behavior.

In particular, two main hypotheses can be tested with respect to the choice of acceptance criterion in ALNS. First, a hypothesis is that the standard Simulated Annealing acceptance criterion is the best criterion, in that it leads to better solutions within a standard running time than when using any other criterion. This hypothesis is reasonable based on the fact that most publications describing ALNS implements this acceptance criterion. Second, a hypothesis is that the influence of the acceptance criterion on the performance and behavior of the search is negligible, that is, the effect size is small compared to random variations in search performance.

The remainder of this paper is structured as follows. In Section 7.2 we give a brief description of the ALNS metaheuristic; Section 7.3 lists the acceptance criteria we are comparing with this work. Sections Sections 7.4 and 7.5 describe the test problems and give details of the implementation of ALNS used to solve them. Section 7.6 explains the process with which we tuned the parameters related to the acceptance criteria. We report computational results in Section 7.7 and finally summarise our findings in the conclusions, in Section 7.8.

7.2 The ALNS Framework

ALNS was introduced by Ropke and Pisinger [27] and extends the LNS metaheuristic first proposed by Shaw [31]. In the LNS, we consider a neighbourhood which is implicitely defined by the sequential application of a *destroy* and a *repair* method. A destroy method turns a feasible solution into an incomplete solution, by destroying parts of it; a repair method then takes an incomplete solution and turns it into a feasible solution. In ALNS, we consider a collection of destroy and repair methods. A neighbourhood is implicitly defined for each possible pair of destroy and repair methods, assuming that any repair method is able to reconstruct a solution from an incomplete solution created by any destroy method.

Some element of randomness is commonly introduced in the process. This element is usually included in the destroy method, by randomising the the choice of which parts of the solution to destroy. In most implementations, the repair methods aim to, myopically, obtain a best possible solution starting from an incomplete solution; however, it is also possible that some stochastic element is introduced in the repair methods. At each iteration, the destroy and repair methods are chosen based on their past performance, reflected by a score: the methods are picked with a roulette-wheel selection, where the probabilities are directly

proportional to the scores. Initially all methods are assigned the same score.

Algorithm 2: General Framework

Input : Initial solution: x_0
Input : Initial acceptance parameters
Input : Initial destroy/repair scores
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Choose a destroy method d
  Choose a repair method r
  $x' \leftarrow r(d(x))$
  if *Accept new solution* x' **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(Destroy/repair scores)
  Update(Acceptance parameters)
  $i = i + 1$
end
return x^*

A synthetic formulation of the ALNS algorithm is given in Algorithm 2. Once the destroy and repair methods are chosen, a new solution (the *incumbent*) is produced. The algorithm then has to decide whether or not to replace the *current* solution with the incumbent — thus *accepting* or *rejecting* the new solution. The criterion used to decide whether or not the incumbent is accepted is therefore called the *acceptance criterion*. The criterion itself can base the acceptance decision on some internal state, which can vary during the course of the solution process. For example, a Simulated Annealing (SA) criterion has been the most popular choice when implementing ALNS: in the case of SA, the varying state is represented by the temperature, which starts at a high value and exponentially decreases during the execution of the algorithm.

When the incumbent is a new global best solution, the scores of the corresponding destroy and repair methods are increased by a relatively large value; otherwise, if the new solution is accepted, their scores are increased by a relatively smaller value; otherwise, if the new solution is not accepted, their scores are decreased.

In our implementation, the solution process ends when we reach a predetermined number of iterations. Other criteria that have been used include a hard time limit, and a predetermined number of consecutive iterations without improvement.

7.3 Acceptance Criteria

In this section we describe the different acceptance criteria tested within the ALNS framework. In the following we denote by $N(x)$ the neighbourhood of a solution x, defined by a selection of destroy and repair heuristics. The cost of a solution x is denoted by $f(x)$. We refer

to the current solution as x; when it is important to specify which iteration of the ALNS algorithm we are considering, we use the notation x_i, where i is the iteration number. The new incumbent solution chosen by the destroy and repair heuristics in $N(x)$ is denoted by x', while we indicate the best encountered solution as x^*. The initial solution is denoted by x_0. Finally, K is the total number of iterations. In the pseudo-code, we will assume that we are minimising the objective function $f(\cdot)$.

The acceptance criteria depend on a given number of parameters, that in our case range from 0 to 4. Some acceptance criteria make use of an internal state, which varies during the solution process, and we assume that the internal state is updated at each iteration of the ALNS algorithm. Alternative criterion-based approaches exist in the literature. For example, one could decide to update certain values of the internal state only when there is apparent convergence with the current settings. Since these strategies cannot be applied uniformly across all the acceptance criteria, we resort to our simpler approach.

Since we are dealing with problem instances that are very diverse in nature and size, we update the internal state used by the acceptance criteria using information relative to the cost of either the best or the current solution, rather than absolute numbers.

7.3.1 Hill Climbing

Hill Climbing (HC), presented in Algorithm 3, accepts an incumbent solution iff it is better than the current one.

Algorithm 3: Hill Climbing

Input : Initial solution: x_0

$x = x_0$ /* Initialise current solution */
$i = 1$ /* Initialise iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $f(x') \leq f(x)$ **then**
    $x = x'$
  end
  $i = i + 1$
end
return x

7.3.2 Random Walk

At the other end of the spectrum from HC, there is Random Walk (RW), presented in Algorithm 4. In this case, we accept every incumbent solution.

7.3.3 Late Acceptance Hill Climbing

This criterion, presented in Algorithm 5, is similar to HC, but the new incumbent solution is compared to what was the current solution L iterations ago. In order to implement this

Algorithm 4: Random Walk

Input : Initial solution: x_0

```
1  x = x_0                                  /* Initialise current solution */
2  x* = x_0                                 /* Initialise best solution */
3  i = 1                                    /* Initialise iteration count */
4  while i ≤ K do
5  |   Pick x' ∈ N(x)
6  |   x = x'
7  |   if f(x) < f(x*) then
8  |   |   x* = x
9  |   end
10 |   i = i + 1
11 end
12 return x*
```

acceptance criterion, it is necessary to keep a circular list of length L that stores the last L current solutions. The criterion was first introduced by Burke and Bykov [3, 4].

Algorithm 5: Late Acceptance Hill Climbing

Input : Initial solution: x_0

Input : List length: L

```
1  x = x_0                                  /* Initialise current solution */
2  x* = x_0                                 /* Initialise best solution */
3  x_{-1}, ..., x_{-L+1} = x_0              /* Initialise list */
4  i = 1                                    /* Iteration count */
5  while i ≤ K do
6  |   Pick x' ∈ N(x)
7  |   if f(x') ≤ f(x_{i-L}) then
8  |   |   x = x'
9  |   end
10 |   if f(x) < f(x*) then
11 |   |   x* = x
12 |   end
13 |   i = i + 1
14 end
15 return x*
```

Parameters related to acceptance: This acceptance criterion only uses parameter: the length L of the look-back list.

Variants: The standard version of this acceptance criterion would not accept the incumbent in case $f(x_{i-L}) < f(x') < f(x)$. As proposed by Burke and Bykov [4], the criterion can be emended to accept the incumbent if *either* it is better than the current solution L iterations ago, *or* it is better than the current solution at the present iteration. In this variant, called **Improved LAHC**, we edit line 7 to become $f(x') \leq f(x_{i-L}) \ \vee \ f(x') \leq f(x)$ (where $\vee$ denotes logical or).

7.3.4 Threshold Acceptance

With the Threshold Acceptance (TA) criterion introduced by Dueck and Scheuer [9] and presented in Algorithm 6, an incumbent solution is accepted if the gap between the incumbent and the current solution is smaller than a threshold T. The threshold starts at a large value and decreases at every iteration.

Algorithm 6: Threshold Acceptance

Input : Initial solution: x_0
Input : Initial threshold: T
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $\frac{f(x')-f(x)}{f(x')} < T$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(T)
  $i = i + 1$
end
return x^*

Parameters related to acceptance: The user-provided parameters are the start threshold T^{start} and the end threshold T^{end}. The initial threshold T is set to its start value. At every iteration, the threshold is updated to move towards its end value.

Variants: We tested two rates of decay: linear and exponential. In the first case, the **Linear Threshold Acceptance** method, we update the threshold as: $T \leftarrow T - (T^{\text{start}} - T^{\text{end}})/K$. In the second case, the **Exponential Threshold Acceptance** method, we update it as $T \leftarrow T \cdot (T^{\text{end}}/T^{\text{start}})^{1/K}$.

7.3.5 Simulated Annealing

Simulated Annealing (SA), presented in Algorithm 7, is the acceptance criterion most commonly used within the ALNS framework. It was originally introduced by Kirkpatrick et al. [17] and it was used with the ALNS since its debut by Ropke and Pisinger [27]. The basic idea behind SA is similar to TA: moves to solutions that are worse than the current one are allowed, but the probability of doing so depends on the state of the search and on the gap between $f(x)$ and $f(x')$.

Parameters related to acceptance: The probability that a new solution of value $f(x')$ is accepted is

$$e^{\frac{f(x)-f(x')}{T}}$$

Algorithm 7: Simulated Annealing

Input : Initial solution: x_0
Input : Initial temperature: T
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if rand$(0,1) \leq e^{\frac{f(x)-f(x')}{T}}$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(T)
  $i = i + 1$
end
return x^*

Given a reference solution value z, if we wanted to accept with probability $p \in [0,1]$ incumbent solutions of cost $f(x') = hz$, we would have to set the temperature T according to:

$$p = e^{\frac{z-hz}{T}} \Rightarrow \ln p = \frac{z(1-h)}{T} \Rightarrow T = \frac{z(1-h)}{\ln p}$$

If we use the reference probability $p = 0.5$ this becomes

$$T = \frac{z(1-h)}{\ln 0.5} \tag{7.1}$$

We can therefore use two user-provided parameters $h^{\text{start}}, h^{\text{end}}$ that define how much worse solutions we accept with probability 0.5 at the beginning and the end of the procedure. The corresponding start and end temperatures T^{start} and T^{end} can then be calculated using (7.1).

Variants: It remains an open question how to choose the reference value z. One option is to use the initial solution: $z = f(x_0)$. The parameter T should then be initialised as T^{start} and then updated at every iteration, as $T \leftarrow T \cdot (T^{\text{end}}/T^{\text{start}})^{1/K}$. We refer to this method, introduced as the default acceptance criterion for ALNS by Ropke and Pisinger [27], simply as **Exponential Simulated Annealing**. A variant of this method has been proposed by Pisinger and Ropke [23], where the authors noticed that the start and end temperature values can be sensitive to the size of the instance. How this *size* is defined is problem dependent (for example, it can be the number of customers in a Vehicle Routing Problem). In the following we just assume that it is a positive real number $s \geq 1$. In the variant of SA that we called **Instance-Scaled Exponential Simulated Annealing**, we divide the start and end temperature by a coefficient s^M, where $M \in \mathbb{N}$ is a parameter. Since Pisinger and Ropke [23] only considered the case where $M = 1$, we take this as the base case upon which we build the following additional variations. The first variation builds on the observation that the best known solution at a certain iteration could be much better than the initial one.

Therefore, the reference value z can be updated every time the best solution value improves, as $T^{\text{end}} = (f(x^*) \cdot (1-h))/\ln 0.5$. This variant, which we call **Exponential Simulated Annealing With Adaptive Probability** coincides with the base method if the value of the initial solution is never improved. Similarly to what we did for TA, we also considered a version of SA where the decrease between start and end temperature is linear. We named this version **Linear Simulated Annealing**. The update function for T is $T \leftarrow T - (T^{\text{start}} - T^{\text{end}})/K$. Another common variant is SA with reheating, discussed by Connolly [6]. Reheating is used to escape local minima in later phases of the exploration, when the temperature is too small to accept a (worsening) diversifying solution. In our implementation we perform reheating a fixed number of times R. When reheating occurs, the temperature is set to the temperature T^* recorded the last time the best solution was improved, multiplied by a coefficient $r > 1$:

$$T \leftarrow rT^* \quad \text{(every } K/(R+1) \text{ iterations)}$$

We call this variant **Exponential Simulated Annealing With Reheating**. On top of the parameters h^{start} and h^{end}, this variant has two additional parameters R and r.

7.3.6 Great Deluge

With the Great Deluge (GD) criterion, introduced by Dueck [8] and presented in Algorithm 8, an incumbent solution is accepted only if its cost is smaller than a threshold, called the *water level*. The water level starts at a high value and decreases at each iteration.

Algorithm 8: Great Deluge

Input : Initial solution: x_0
Input : Initial water level: W
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $f(x') < W$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(W)
  $i = i + 1$
end
return x^*

Parameters related to acceptance: The two key parameters used for GD are the initial water level and the decrease rate. The initial water level is set to $W = \alpha \cdot f(x_0)$, where $\alpha > 1$ is a user-provided parameter. The water level is then decreased at each iteration, $W \leftarrow W - \beta(W - f(x))$, according to another parameter $\beta \in (0,1)$.

7.3.7 Non-Linear Great Deluge

The Non-Linear Great Deluge criterion (NLGD), presented in Algorithm 9, builds on the same idea of the GD, with a few variations. The water level decreases more quickly in the beginning of the search process, more slowly towards the end, and can also increase. The NLGD was introduced by Landa-Silva and Obit [18] for a course timetabling problem; in our implementation we change some of the fixed values, which the authors tuned for their specific problem, and we replace them with parameters.

Algorithm 9: Non-Linear Great Deluge

Input : Initial solution: x_0
Input : Initial water level: W
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $f(x') < W \ \vee \ f(x') < f(x)$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(W)
  $i = i + 1$
end
return x^*

The general form of this acceptance criterion is similar to the criterion in Algorithm 8. The only difference is that the acceptance criterion checks that *either* the new solution has a cost lower than the current water level, *or* it improves over the current solution. This is done because in NLGD the water level is not guaranteed to be above the cost of the current solution.

Parameters related to acceptance:

The initial water level is chosen similarly as for GD: $W = \alpha \cdot f(x_0)$, with a user-provided parameter $\alpha > 1$. Three additional parameters — β, γ, and δ — are used to update the water level at each iteration, according to the decision flow in Algorithm 10: if the new incumbent solution is worse than the water level, then the water level tends to increase, to increase the chance of accepting new solutions. If the last solution is better than the water level, but not much better (the gap is smaller than β), then again we increase the water level, for similar reasons. On the other hand, if the gap is larger than β, we decrease the water level and the decrease function is exponential.

Algorithm 10: Update(W)

$G = \frac{W - f(x')}{W}$ /* Gap between water level and incumbent */
if $G < \beta$ **then**
  return $W + \gamma \cdot |f(x') - W|$ /* Re-increase W */
else
  return $W \cdot e^{-\delta \cdot f(x^*)} + f(x^*)$ /* Exponentially decrease W */
end

7.3.8 Record-to-Record Travel

The Record-to-Record Travel (RRT) criterion presented in Algorithm 11 is similar to TA, but the incumbent solution is accepted if the gap between the incumbent and the best (rather than the current) solution is smaller than a threshold T. The threshold starts at a large value and decreases at every iteration to reach its predetermined value at the end of the search process.

Algorithm 11: Record-to-Record Travel

Input : Initial solution: x_0
Input : Initial threshold: T
$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $\frac{f(x') - f(x^*)}{f(x')} < T$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(T)
  $i = i + 1$
end
return x^*

Parameters related to acceptance: The user-provided parameters are the start threshold T^{start} and the end threshold T^{end}. The initial threshold T is set to its start value and, at each iteration, moves towards the end value.

Variants: Analogous to what was done for TA, we tested two rates of decay that give rise to two variants that we call **Linear Record-to-Record Travel** and **Exponential Record-to-Record Travel**.

7.3.9 Worse Accept

The Worse Accept (WA) criterion presented in Algorithm 12 tries to increase diversification by accepting an incumbent solution if it improves over the current one, or — regardless of

its cost — with a given probability, p. This probability is higher at the beginning and smaller at the end of the solution process. This is, to our best knowledge, the first time that such a method is considered in the literature.

This criteria is particularly suited in cases when the objective value of the problem typically holds a few discrete values, and passing from a value to the next better one is a relatively rare occurrence. An example of such a problem is the Vertex Colouring Problem (VCP), in which one has to produce a colouring of a graph, using the smallest number of colours. WA was employed as the acceptance criterion in an ALNS-based metaheuristic for the Partition Colouring Problem (a generalisation of the VCP) by Furini et al. [10].

Algorithm 12: Worse Accept

Input : Initial solution: x_0
Input : Initial probability: p

$x = x_0$ /* Initialise current solution */
$x^* = x_0$ /* Initialise best solution */
$i = 1$ /* Iteration count */
while $i \leq K$ **do**
  Pick $x' \in N(x)$
  if $f(x') < f(x) \vee \text{rand}(0,1) < p$ **then**
    $x = x'$
  end
  if $f(x) < f(x^*)$ **then**
    $x^* = x$
  end
  Update(p)
  $i = i + 1$
end
return x^*

Parameters related to acceptance: The user-provided parameters are the start probability p^{start} and the end probability p^{end}.

Variants: The probability decay, similarly to what done for other methods, can be linear or exponential. This gives rise to two criteria: **Linear Worse Accept** and **Exponential Worse Accept**.

7.3.10 Parameter space reduction

For the linear variants of methods TA, SA, WA and RRT, it is sensible to set the end parameter (be it threshold, temperature or probability) to values very close to zero. We can therefore reduce the dimension of the parameter space, by simply fixing these end parameters to 0. The resulting new methods are referred to by using the additional suffix "**(fixed end)**". Notice that, on the other hand, an exponential decay function can never reach the value 0, by definition.

7.4 Test Problems

To evaluate the different acceptance criteria, we consider ALNS implementations for two different combinatorial optimisation problems, as presented below.

7.4.1 Capacitated Vehicle Routing Problem

In the Capacitated Vehicle Routing Problem (CVRP) we have to deliver goods from a depot to a set of customers, using an unlimited fleet of identical vehicles. Each customer demands a certain quantity of goods and the vehicles have a limited capacity. Our task is to construct routes starting and ending at the depot that minimise the total travel distance and that obey the capacity of the vehicles. We assume that travel distances are symmetric in the sense that the distance from A to B is the same as the distance from B to A. The problem can be modelled on a directed graph $G = (N, A)$ where the node set is $N = \{0, \dots, n\}$ and node 0 represents the depot, while nodes $C = \{1, \dots, n\}$ represent the customers. Each customer $i \in C$ has an associated demand $q_i \geq 0$ and the vehicles all have the same capacity $Q \geq \max_{i \in C} q_i$.

In the literature on heuristics for the CVRP, researchers have typically also considered instances that include a distance or duration limit for each route. In the standard benchmark instances, customers have a service time and for each route the sum of service times plus distance driven has to be less than or equal to a threshold L. For more information the reader is referred to Irnich et al. [16] and Laporte et al. [19].

7.4.2 Capacitated Minimum Spanning Tree Problem

In the (symmetric) Capacitated Minimum Spanning Tree (CMST) we have to construct a spanning tree subject to a capacity constraint. The problem is defined on a undirected graph $G = (N, E)$ where N is the node set and E are the edges. For each edge $e \in E$ we are given an associated cost $c_e \geq 0$. In the node set $N = \{0, \dots, n\}$, node 0 is the root node. The remaining nodes $i \in N \setminus \{0\}$ are associated with a demand $d_i \geq 0$ and we are given a maximum demand or capacity Q. Removing node 0 from any spanning tree results in the tree splitting into one or more connected components. In the CMST, the solution has to satisfy the property that the sum of the demands of each component (or sub-tree) is less than or equal to Q (capacity constraints). We seek the spanning tree that minimizes the sum of edge costs while satisfying capacity constraints. For more information on this problem, see Uchoa et al. [32].

7.5 ALNS applied to Test Problems

In the following we describe details of ALNS implementations for each of the two optimisation problems that we are solving. We point out that we used the parallel version of ALNS described in Ropke and Santini [29], with the number of parallel threads set to 8.

7.5.1 ALNS for the CVRP

Let n be the number of customers in the instance. We determine an upper bound for the number of customers to remove based on two parameters: an absolute upper bound $\bar{\omega}^+$ and a relative one ω^+. The upper bound is then $n^+ = \min\{\bar{\omega}^+, \omega^+ n\}$. Similarly a lower bound is based on the parameters $\bar{\omega}^-$ and ω^-; the lower bound is $n^- = \min\{n^+, \max\{\bar{\omega}^-, \omega^- n\}\}$. Based on the upper and lower bound we select the number of customers to remove, r, as a uniformly random number in the interval $\{n^-, \ldots, n^+\}$.

The destroy method used are: random removal, relatedness removal (introduced by Shaw [31]), and history-based removal. These methods are described in detail in Ropke and Pisinger [28, Section 5]. The repair method used is called regret repair, first introduced for vehicle problems by Potvin and Rousseau [24] and described in detail in Ropke and Pisinger [27, Section 3.2.2]. A steepest descent algorithm based on a small neighborhood is also implemented to improve the solution found by the regret heuristic. The descent algorithm uses the 2-opt neighborhood, both considering the intra-route and the inter-route variant (also known as 2-opt*, see Laporte et al. [19]). In order to save running time, it is not used every time a partial solution has been repaired, but only with a given probability $p^{\text{2-opt}}$.

A random starting solution is created by constructing routes iteratively. Let U be the set of customers that are still not placed in the solution. Initially U contains all customers. In order to start a new route, a random seed customer is selected from U. Customers are then added to the route until the capacity or the length constraint on the route disallow further insertions. When choosing the customer to insert into a growing route, the algorithm simply selects the customer whose insertion increases the cost of the route the least. Whenever a route is full, a new route is created following the same procedure. This process continues until all customers have been inserted.

7.5.2 Simple LNS for the CVRP

A simplified version of the ALNS is also considered for the CVRP. The reason for this is that the full ALNS was developed using the SA acceptance criterion, and that the selection of components in the full ALNS could therefore be biased towards components that fit well with the behavior of the SA criterion. The simple LNS for the CVRP uses a single destroy and a single repair method. The destroy method is random removal and the repair method is the deterministic regret method. The repair method does not include the local improvement method. The number of customers to remove and the initial solution are found in the same way as for the more complex ALNS method. We sometimes refer to this combination of an ALNS implementation and test problem as *Simple CVRP*.

7.5.3 CMST

To the best of our knowledge, the first application of the ALNS metaheuristic to the CMST problem is presented in Ropke and Santini [29]. In the following, we give a brief summary of the implementation, while referring the reader to the cited article for more details.

The number of nodes of the graph to remove is determined in the same way as for the CVRP (see Section 7.5.1). The destroy methods used are relatedness removal and history-based

removal, which are analogous to the CVRP methods with the same names. Similarly, the repair method, regret repair, is analogous to the method used for the CVRP. Furthermore, we also used a greedy insertion repair method. The solutions produced by the repair methods are improved by solving a minimum spanning tree problem for each sub-tree of the solution.

Unlike what is done for the CVRP, the initial solution is created deterministically by a two-stage procedure that first estimates the number of sub-trees that need to be created, and then assigns nodes to the subtrees.

7.5.4 Problem-specific parameters

Some parameters of the ALNS implementations, relative to the problem-specific destroy and repair heuristics, and to local improvement methods, are kept at fixed values. Table 7.1 describes the values of these parameters.

Problem	Param type	Parameter	Values
CMST	Destroy	Number of nodes to destroy	$\bar{\omega}^+ = 30, \omega^+ = 0.4, \bar{\omega}^- = 5, \omega^- = 0.1$
CMST	Destroy	Destroy close nodes	$\eta = \frac{n}{2}, p^{\text{fix}} = 4$
CMST	Destroy	Historical node-pair destroy	$p^{\text{hist}} = 5$
CMST	Repair	Regret repair	$p^{\text{regret}} = 1.5$ (stochastic version)
CVRP	Destroy	Number of nodes to destroy	$\bar{\omega}^+ = 50, \omega^+ = 0.4, \bar{\omega}^- = 10, \omega^- = 0.1$
CVRP	Destroy	Relatedness destroy method	$p^{\text{rel}} = 5$
CVRP	Destroy	Historical node-pair destroy	$p^{\text{hist}} = 5$
CVRP	Repair	Regret repair	$p^{\text{regret}} = 1.5$ (stochastic version)
CVRP	Local impr.	2-opt* local search	$p^{\text{2-opt}} = 0.1$

Table 7.1: Problem-specific parameters which have been kept fixed.

7.6 Parameter Tuning

With a few exceptions, all acceptance criteria described in Section 7.3 depend on one or more parameters. In order to tune these parameters an algorithmic approach is preferred to a manual one in order to avoid bias toward acceptance criteria that the authors know well. A substantial amount of literature is available on algorithms for automatic parameter tuning, and some prominent examples are described in the works by Birattari et al. [2] and Hutter et al. [15]. In this work we have implemented a simple iterated local search procedure to perform parameter tuning, as described below.

Given an acceptance criterion and a problem chosen among the ones we consider in this work (CMST, CVRP, and Simple CVRP), let N be the number of parameters we are tuning. Let n be the number of integer parameters and r the number of real-valued parameters. We assume without loss of generality that the parameters are numbered $\alpha_1, \dots, \alpha_n, \alpha_{n+1}, \dots, \alpha_{n+r}$, and that $N = n + r$. The parameter space will then be $\mathscr{P} = \mathbb{N}^n \times \mathbb{R}^r$.

The aim of the parameter tuning is to explore the parameter space, starting from an initial parameter assignment $\alpha^0 = (a_1^0, \dots, a_N^0) \in \mathscr{P}$, in a certain number $M \in \mathbb{N}$ of iterations, and return the assignment that gives, *on average*, the best results for the acceptance criterion and problem considered. Let $I_1, \dots, I_K$ be the instances used for parameter tuning and let $B_1, \dots, B_K$ be the best objective function values known from the literature for the instances

(these might not be the optimal ones, if the instance is open). For any given parameter assignment α, the algorithm is (re-)run $\lambda \in \mathbb{N}$ times, unchanged, on each instance. This produces K average results, one for each instance, calculated as

$$A_{\alpha,k} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} v_{\alpha,i,k}$$

where $v_{\alpha,i,k}$ is the solution value obtained by the algorithm for instance I_k at the i-th rerun, with parameter assignment α.

We can then calculate the deviation from the best known result, for each instance:

$$D_{\alpha,k} = \frac{A_{\alpha,k} - B_k}{A_{\alpha,k}}$$

The score of assignment α is calculated as the average deviation across all instances:

$$S_\alpha = \frac{1}{K} \sum_{k=1}^{K} D_{\alpha,k}$$

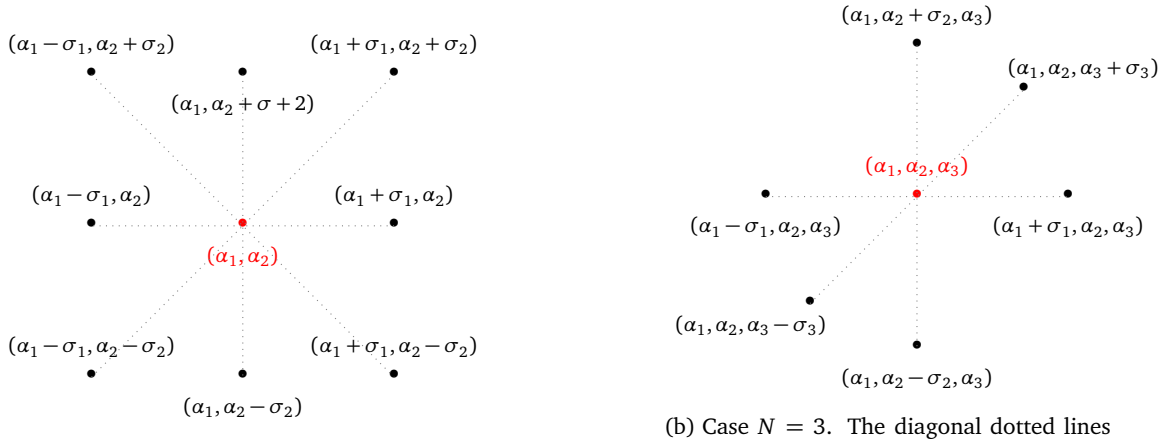
The lower the score and, in particular, the closer it is to 0, the better is the parameter assignment α.

Algorithm 13: Parameter Tuning Algorithm

Input : Initial parameters α^0
Input : Initial steps: σ^0
for $k = 1, \ldots, M$ **do**
  $\alpha^{\text{new}} = \text{BestInNb}(\alpha^{k-1}, \sigma^{k-1})$
  if $\alpha^{\text{new}} \neq \alpha^{k-1}$ **then**
    $\alpha^k = \alpha^{\text{new}}$
    $\sigma^k = \sigma^{k-1}$
  else
    $\alpha', \alpha'' = \text{BestTwo}()$
    $\alpha^k = \text{NewCentre}(\alpha', \alpha'')$
    $\sigma^k = \text{NewSteps}(\alpha', \alpha'')$
    if $\alpha^k = \alpha^{\text{new}}$ or $\text{StepsTooSmall}(\sigma^k)$ **then**
      $\alpha^k = \text{Diversify}(\alpha^k)$
      $\sigma^k = \sigma^0$
    end
  end
end
return $\arg\min_{k=1,\ldots,M} \{S_{\alpha^k}\}$

A general overview of the parameter tuning algorithm is given in Algorithm 13. An initial parameter assignment α^0 is given, together with an initial step σ^0. The step defines the neighbourhood of the current assignment:

$$\mathcal{N}(\alpha) = \left\{ (\alpha'_1, \ldots, \alpha'_N) \mid \alpha'_i - \alpha_i \in \{-\sigma_i, 0, \sigma_i\} \ \forall i = 1, \ldots, N \right\} \tag{7.2}$$



(a) Case $N = 2$.

(b) Case $N = 3$. The diagonal dotted lines represent movement along a third axis.

Figure 7.1: Representation of neighbourhood $\mathcal{N}(\alpha)$.

The neighbourhood is defined by all possible combination of moves, in all the directions defined by the components of the parameter vector, each by its corresponding step, with $\sigma_1^0,\dots,\sigma_n^0 \in \mathbb{N}$ and $\sigma_{n+1}^0,\dots,\sigma_N^0 \in \mathbb{R}$. For larger values of N, the exploration of the neighbourhood defined above is computationally expensive. Therefore, for values of $N \geq 3$, we define the alternative neighbourhood:

$$\begin{aligned}\mathcal{N}(\alpha) = \{(\alpha_1',\dots,\alpha_N') \mid \\ \exists i \in \{1,\dots,N\} : \alpha_i' - \alpha_i \in \{-\sigma_i, 0, \sigma_i\} \text{ and} \\ \forall j \neq i \ \ \alpha_j' = \alpha_j\}\end{aligned} \tag{7.3}$$

According to definition (7.3), therefore, we can only move along one direction at a time. Figure 7.1a and Figure 7.1b give a graphical representation of $\mathcal{N}(\alpha)$ for $N = 2$ and $N = 3$.

At each iteration of the algorithm, the next parameter assignment is chosen in the neighbourhood of the current one (line 2) as the one with the best score:

$$\alpha^{k+1} = \arg\min\left\{S_{\alpha'} \mid \alpha' \in \mathcal{N}(\alpha^k)\right\}$$

When $\alpha^{k+1} = \alpha^k$, we have reached a local optimum and the search must be interrupted and restarted somewhere else in the parameter space. In order to do this, we retrieve the best and second-best parameter configuration encoutered during the whole search, α' and α'' respectively (line 7), and we set the current parameter configuration as the centre of mass between α' and α'' (line 8):

$$\alpha^k = \left(\frac{\alpha_1' + \alpha_1''}{2},\dots,\frac{\alpha_N' + \alpha_N''}{2}\right)$$

where integer components are rounded to the nearest integer. The step sizes are also recalculated (line 9) and set as:

$$\sigma^k = \left(\frac{|\alpha_1' - \alpha_1''|}{3},\dots,\frac{|\alpha_N' - \alpha_N''|}{3}\right)$$

and, again, integer components are rounded. If, after recalculating α^k, all steps are below their minimum step size (which is a predetermined parameter), or if it happened that α^k did not change (line 10) we proceed with a stronger diversification (line 11) and we reset the step sizes (line 12). The strong diversification consists in setting:

$$\alpha^k = \left(\alpha_1^{k-1} + \rho_1 \sigma_1^0, \ldots, \alpha_N^{k-1} + \rho_N \sigma_N^0\right)$$

where each ρ_i is taken randomly from the intervals $[-3,-1] \cup [1,3]$.

Table 7.2 summarises the results of parameter tuning for the three problems considered, using six tuning instances for each problem. Column "Acceptance Criterion" shows the acceptance criteria, column "Score" gives the value of S_{α^*} for the best parameter assignment $\alpha^* \in \mathscr{P}$, while column "Parameters" gives the values of the parameters in α^*, using the same notation as in Section 7.3. The maximum number of tuning iterations has been set to $M = 20$, the number of reruns to $\lambda = 10$ and the number of iterations of each run (exit criterion) to 150,000.

When the number of parameters is less than three, it is also possible to easily visualise the progress of the parameter tuning algorithm, constructing a heat map with the score of each parameter assignment. Figure 7.2 shows such an example, for CVRP with the acceptance criterion Linear Record-to-Record Travel, where the horizontal axis represents the values of T^{start} and vertical axis represents T^{end}.

7.7 Results

The computational experiments have been conducted on the following instances. For CMST: 104 instances, available as the `capmst` test set in the OR Library of Beasley [1], containing from 41 to 200 nodes. For CVRP: 14 instances by Christofides et al. [5]; 13 instances by Rochat and Taillard [26]; 20 instances by Golden et al. [11]; 12 instances by Li et al. [21]; 100 instances by Uchoa et al. [33]. The CVRP instances contain between 50 and 1200 customers. The number of iterations and reruns were the same as used for parameter tuning: 150,000 iterations and 10 reruns.

Table 7.3 summarises the main results, reporting for each acceptance criterion the average deviation to the best known solution from both the average (column "aDev") and the best (column "bDev") solution obtained over the 10 runs for each instance. The results are shown separately for the CMST, the CVRP using a full ALNS, and the CVRP using a simple LNS. The last column ("aTime") reports the average solution time. Notice that the Random Walk criterion has consistently higher running time, and this is due to a technical reason in the implementation of the algorithm: every time a solution is accepted (which is, for Random Walk, at every iteration) a potentially expensive copy is performed, to store the solution object and replace the current solution object.

The results have further been analysed using the Wilcoxon signed-rank test, by comparing each pair of acceptance criteria under the null-hypothesis that the deviations between the average solution found and the best known solution are drawn from identical distributions. Figure 7.3 summarises the Wilcoxon test for the CMST, with one node per acceptance criterion and an arc going from the better criterion to the worse criterion if the null-hypothesis is

	CMST		CVRP		Simple LNS for CVRP	
Acceptance Criterion	Score	Parameters	Score	Parameters	Score	Parameters
GD	$2.216 \cdot 10^{-2}$	$\alpha = 1.0167, \beta = 0.0001$	$1.386 \cdot 10^{-2}$	$\alpha = 1.0167, \beta = 0.0002$	$3.362 \cdot 10^{-2}$	$\alpha = 1.1241, \beta = 0.0002$
HC	$4.563 \cdot 10^{-2}$		$1.890 \cdot 10^{-2}$		$4.845 \cdot 10^{-2}$	
LAHC	$1.960 \cdot 10^{-2}$	$L = 22500$	$1.397 \cdot 10^{-2}$	$L = 15000$	$3.516 \cdot 10^{-2}$	$L = 10833$
Improved LAHC	$2.024 \cdot 10^{-2}$	$L = 9180$	$1.340 \cdot 10^{-2}$	$L = 4166$	$3.472 \cdot 10^{-2}$	$L = 4248$
NLGD	$2.794 \cdot 10^{-2}$	$\alpha = 2.1714, \beta = 0.0465, \gamma = 0.1057, \delta = 0.0096$	$1.470 \cdot 10^{-2}$	$\alpha = 1.2500, \beta = 0.0075, \gamma = 0.0208, \delta = 0.0100$	$3.481 \cdot 10^{-2}$	$\alpha = 1.1042, \beta = 0.0050, \gamma = 0.0000, \delta = 0.0183$
RW	$5.828 \cdot 10^{-2}$		$3.062 \cdot 10^{-2}$		$4.730 \cdot 10^{-2}$	
Lin. RRT	$1.776 \cdot 10^{-2}$	$T^{\text{start}} = 0.0750, T^{\text{end}} = 0.0037$	$9.060 \cdot 10^{-3}$	$T^{\text{start}} = 0.0222, T^{\text{end}} = 0.0000$	$2.333 \cdot 10^{-2}$	$T^{\text{start}} = 0.0176, T^{\text{end}} = 0.0000$
Lin. RRT (fixed end)	$1.773 \cdot 10^{-2}$	$T^{\text{start}} = 0.0500$	$8.733 \cdot 10^{-3}$	$T^{\text{start}} = 0.0167$	$2.405 \cdot 10^{-2}$	$T^{\text{start}} = 0.0222$
Exp. RRT	$2.044 \cdot 10^{-2}$	$T^{\text{start}} = 0.0250, T^{\text{end}} = 0.0289$	$1.133 \cdot 10^{-2}$	$T^{\text{start}} = 0.0042, T^{\text{end}} = 0.0376$	$2.679 \cdot 10^{-2}$	$T^{\text{start}} = 0.0125, T^{\text{end}} = 0.0906$
Exp. SA with Ad. Probab.	$1.649 \cdot 10^{-2}$	$h^{\text{start}} = 9.7500, h^{\text{end}} = 2.0093$	$1.218 \cdot 10^{-2}$	$h^{\text{start}} = 4.7500, h^{\text{end}} = 0.6944$	$2.862 \cdot 10^{-2}$	$h^{\text{start}} = 20.273, h^{\text{end}} = 0.5141$
Exp. SA	$1.698 \cdot 10^{-2}$	$h^{\text{start}} = 0.1128, h^{\text{end}} = 0.0104$	$1.130 \cdot 10^{-2}$	$h^{\text{start}} = 0.1211, h^{\text{end}} = 0.0004$	$2.647 \cdot 10^{-2}$	$h^{\text{start}} = 0.1367, h^{\text{end}} = 0.0008$
Lin. SA	$1.606 \cdot 10^{-2}$	$h^{\text{start}} = 11.500, h^{\text{end}} = 1.7917$	$1.132 \cdot 10^{-2}$	$h^{\text{start}} = 3.7500, h^{\text{end}} = 0.4097$	$2.788 \cdot 10^{-2}$	$h^{\text{start}} = 9.0000, h^{\text{end}} = 0.0000$
Lin. SA (fixed end)	$1.651 \cdot 10^{-2}$	$h^{\text{start}} = 12.193$	$1.180 \cdot 10^{-2}$	$h^{\text{start}} = 6.8152$	$2.750 \cdot 10^{-2}$	$h^{\text{start}} = 12.347$
Instance-scaled Exp. SA	$1.601 \cdot 10^{-2}$	$h^{\text{start}} = 13.507, h^{\text{end}} = 2.0903, M = 1.0000$	$1.122 \cdot 10^{-2}$	$h^{\text{start}} = 4.2083, h^{\text{end}} = 0.6181, M = 1.0000$	$2.733 \cdot 10^{-2}$	$h^{\text{start}} = 14.229, h^{\text{end}} = 0.6250, M = 1.0000$
Exp. SA with Reheating	$1.611 \cdot 10^{-2}$	$h^{\text{start}} = 12.000, h^{\text{end}} = 1.8750, r = 3.5000, R = 1.0000$	$1.138 \cdot 10^{-2}$	$h^{\text{start}} = 13.500, h^{\text{end}} = 0.6250, r = 0.5000, R = 1.0000$	$2.821 \cdot 10^{-2}$	$h^{\text{start}} = 12.750, h^{\text{end}} = 0.7500, r = 2.4167, R = 2.5833$
Lin. TA	$1.648 \cdot 10^{-2}$	$T^{\text{start}} = 0.0708, T^{\text{end}} = 0.0014$	$1.099 \cdot 10^{-2}$	$T^{\text{start}} = 0.0250, T^{\text{end}} = 0.0000$	$2.599 \cdot 10^{-2}$	$T^{\text{start}} = 0.0212, T^{\text{end}} = 0.0003$
Lin. TA (fixed end)	$1.667 \cdot 10^{-2}$	$T^{\text{start}} = 0.0875$	$1.123 \cdot 10^{-2}$	$T^{\text{start}} = 0.0292$	$2.597 \cdot 10^{-2}$	$T^{\text{start}} = 0.0208$
Exp. TA	$2.259 \cdot 10^{-2}$	$T^{\text{start}} = 0.0125, T^{\text{end}} = 0.0023$	$1.296 \cdot 10^{-2}$	$T^{\text{start}} = 0.0016, T^{\text{end}} = 0.0017$	$3.087 \cdot 10^{-2}$	$T^{\text{start}} = 0.0033, T^{\text{end}} = 0.0059$
Exp. WA	$1.794 \cdot 10^{-2}$	$p^{\text{start}} = 0.7851, p^{\text{end}} = 0.0979$	$1.754 \cdot 10^{-2}$	$p^{\text{start}} = 0.0500, p^{\text{end}} = 0.0150$	$3.121 \cdot 10^{-2}$	$p^{\text{start}} = 1.0000, p^{\text{end}} = 0.1090$
Lin. WA	$1.819 \cdot 10^{-2}$	$p^{\text{start}} = 0.6580, p^{\text{end}} = 0.0430$	$1.744 \cdot 10^{-2}$	$p^{\text{start}} = 0.1500, p^{\text{end}} = 0.0167$	$2.974 \cdot 10^{-2}$	$p^{\text{start}} = 1.0000, p^{\text{end}} = 0.0022$
Lin. WA (fixed end)	$1.867 \cdot 10^{-2}$	$p^{\text{start}} = 0.5500$	$1.426 \cdot 10^{-2}$	$p^{\text{start}} = 1.0000$	$3.046 \cdot 10^{-2}$	$p^{\text{start}} = 0.9833$

Table 7.2: Parameter tuning results sumamry for CMST, CVRP and Simple LNS for CVRP.

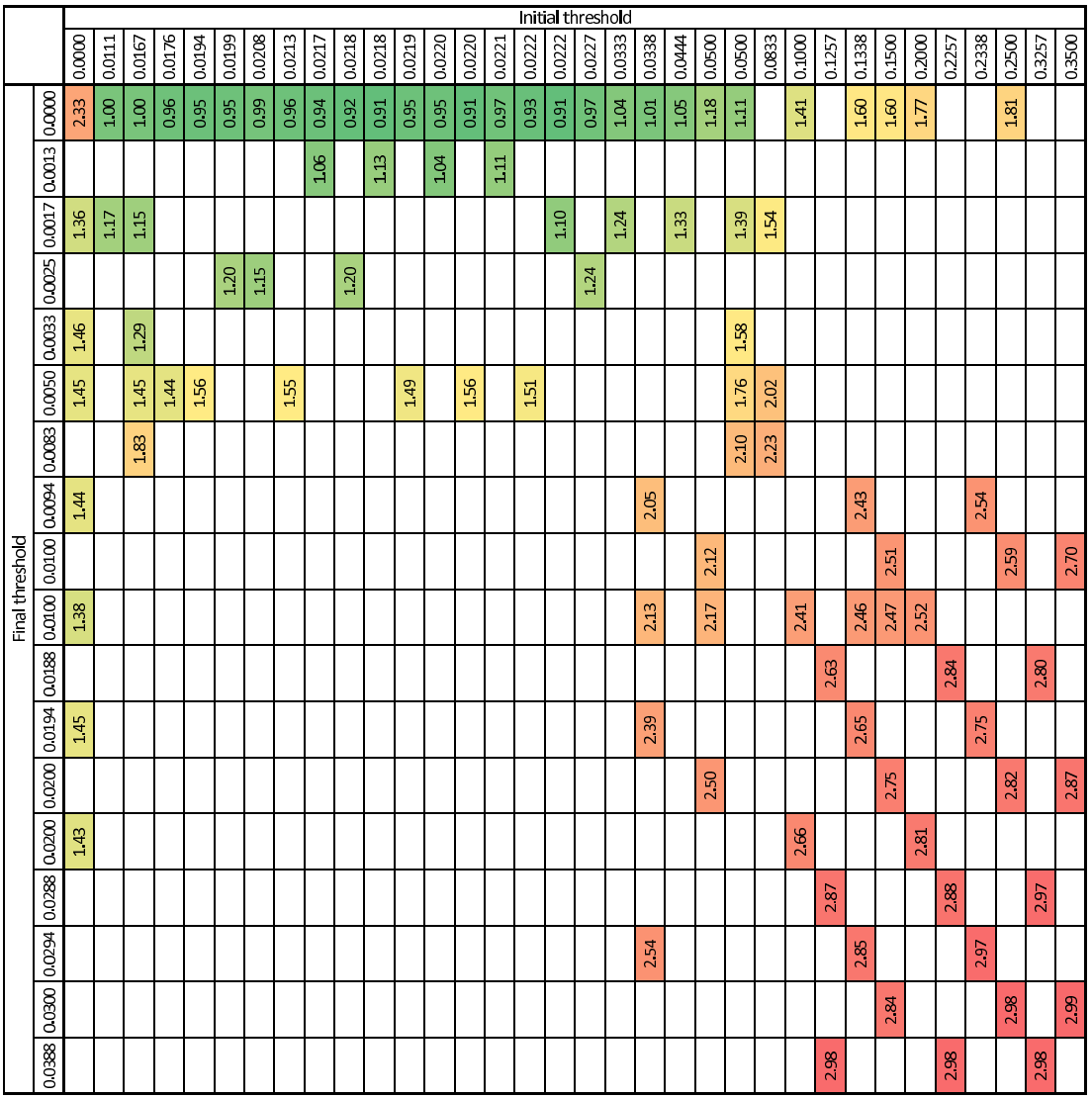


Figure 7.2: Parameter tuning heatmap for CVRP and acceptance criterion "Linear RRT". The horizontal and vertical axes represent, respectively, parameters T^{start} and T^{end}. The tuning algorithm only attempted parameter configurations indicated by the filled cells. Each filled entry reports average deviation S_α (in percentage) for the corresponding parameter assignment α, and the cells are colored according to average deviation with green cells indicating the best results. Notice that labels on both axes are unevenly distributed and that the tuning algorithms attempts some parameter configuration where the initial threshold is larger than the final threshold.

rejected at a 0.05 significance level. The same is shown for the full ALNS for CVRP in Figure 7.4 and for the simple LNS for CVRP in Figure 7.5.

One of the goals of this study was to quantify the effect that different move acceptance criteria have on the performance of an ALNS. From Table 7.3 it is clear that the consequences of using a substandard move acceptance criterion can be quite large. There are two criteria that are clearly much worse than all the others: RW and HC, whose average performance is between one and two percentage points worse than the best acceptance criteria. Even when disregarding RW and HC, the difference between the best criteria and the worst of the rest is more than 0.5 percentage points for the full ALNS implementations, and even larger for the simpler LNS method.

Another goal of the study was to determine which move acceptance criterion is best suited for the ALNS. The results are not entirely clear at this point, but by extracting information from the Wilcoxon signed-rank tests, some conclusions can be reached. The simple criteria RW and HC are clearly inferior to the alternatives. The order of the other acceptance criteria vary between problems, but they can be separated in two groups: criteria that are close to being top ranked for at least one problem, and criteria that are always mediocre. In the first category we find variants of SA, RRT, and TA, and in the latter category we find variants of LAHC, GD, NLGD, and WA.

Differentiating between the three best types of acceptance criteria is not straightforward: a variant of SA is best for CMST, whereas a variant of RRT is best for CVRP. On the other hand, a version of TA is better than RRT on CMST and better than SA on CVRP. Further analysis of these three criteria may be necessary. As each of SA, RRT, and TA were implemented in different variants, it is possible to compare whether linear or exponential versions are better, and whether it is better to fix the end point (fixed end), or to allow the parameter tuning process to potentially find better end points for the control parameters: The linear version of RRT is better than the exponential version of RRT, with statistical significance for each of CMST, CVRP and simple CVRP. The linear version of TA is better than the exponential version of TA, again with statistical significance for all three test sets. There are never any statistically significant differences between the exponential and linear versions of SA. Regarding versions with fixed end, no interesting pattern emerges: it seems that the parameter tuning process was able to obtain similar performance whether or not the end point for the control parameter was fixed.

Regarding the two hypotheses stated in the introduction, we cannot reject the notion that SA is one of the best move acceptance criteria as, even though linear RRT is performing better for CVRP, linear SA is better for CMST. On the other hand, we can reject the hypothesis that the effect of the move acceptance criterion is small compared to random effects when solving each instance: we find clear evidence that some move acceptance criteria perform worse than others, for example that WorseAccept is worse than linear SA with statistical significance.

A third goal of this study was to measure how different move acceptance criteria may influence the search behavior. To analyse this, statistics were collected during each run and analysied using multiple linear regression. In the regression, the dependent variable is the deviation between the average objective function in a run and the best known solution value. Hence, there is one observation for each combination of an instance and a move acceptance criterion. Eleven independent variables are included, corresponding to the following statistics

CMST

Acceptance Criterion	aDev %	bDev %	aTime (s)
Lin. SA	0.399	0.108	9.367
Instance-scaled Exp. SA	0.400	0.150	9.223
Lin. SA (fixed end)	0.407	0.119	9.224
Exp. SA	0.409	0.127	9.087
Lin. TA (fixed end)	0.418	0.119	9.470
Exp. SA with Reheating	0.428	0.174	9.086
Lin. RRT	0.473	0.213	7.888
Lin. TA	0.474	0.120	9.156
Exp. SA with Ad. Probab.	0.509	0.159	8.665
Lin. RRT (fixed end)	0.514	0.234	7.691
Lin. WA (fixed end)	0.518	0.203	8.186
Exp. WA	0.552	0.181	8.394
Lin. WA	0.566	0.195	8.361
Improved LAHC	0.644	0.221	7.156
Exp. RRT	0.646	0.269	6.758
LAHC	0.655	0.244	7.380
GD	0.682	0.371	6.586
Exp. TA	0.759	0.315	8.818
NLGD	0.995	0.492	7.665
HC	2.226	1.215	6.586
RW	2.824	2.305	12.110

CVRP

Acceptance Criterion	aDev %	bDev %	aTime (s)
Lin. RRT (fixed end)	0.391	0.112	17.871
Lin. RRT	0.423	0.148	18.443
Lin. TA	0.497	0.179	20.056
Lin. TA (fixed end)	0.511	0.197	20.285
Exp. SA with Reheating	0.527	0.175	18.508
Lin. SA	0.527	0.167	17.500
Exp. SA	0.529	0.173	18.374
Lin. SA (fixed end)	0.538	0.200	18.461
Instance-scaled Exp. SA	0.542	0.159	17.328
Exp. RRT	0.551	0.126	16.308
Exp. SA with Ad. Prob.	0.578	0.212	17.243
Lin. WA (fixed end)	0.661	0.301	19.263
LAHC	0.716	0.282	17.056
Improved LAHC	0.720	0.307	17.719
GD	0.726	0.463	17.901
Exp. TA	0.735	0.276	16.416
Lin. WA	0.963	0.496	16.348
NLGD	0.989	0.393	15.453
Exp. WA	1.147	0.510	14.285
HC	1.163	0.557	14.008
RW	2.583	2.226	24.143

Simple LNS for CVRP

Acceptance Criterion	aDev %	bDev %	aTime (s)
Lin. RRT (fixed end)	0.754	0.241	11.685
Lin. RRT	0.768	0.218	11.547
Exp. RRT	0.939	0.315	10.421
Lin. TA (fixed end)	0.972	0.358	13.497
Lin. TA	0.973	0.328	13.529
Exp. SA	1.062	0.363	13.202
Instance-scaled Exp. SA	1.076	0.399	13.129
Lin. SA (fixed end)	1.086	0.443	13.507
Lin. SA	1.112	0.427	13.206
Exp. SA with Reheating	1.150	0.445	12.744
Lin. WA (fixed end)	1.270	0.580	10.216
Exp. SA with Ad. Prob.	1.398	0.526	12.979
Exp. TA	1.425	0.591	12.165
NLGD	1.695	0.713	11.033
GD	1.709	1.189	11.958
LAHC	1.870	0.986	8.208
Improved LAHC	1.879	0.988	7.329
Lin. WA	2.461	1.272	6.347
Exp. WA	2.516	1.312	6.153
HC	2.595	1.381	5.810
RW	3.946	3.340	15.126

Table 7.3: Final results for CMST, CVRP and Simple LNS for CVRP.

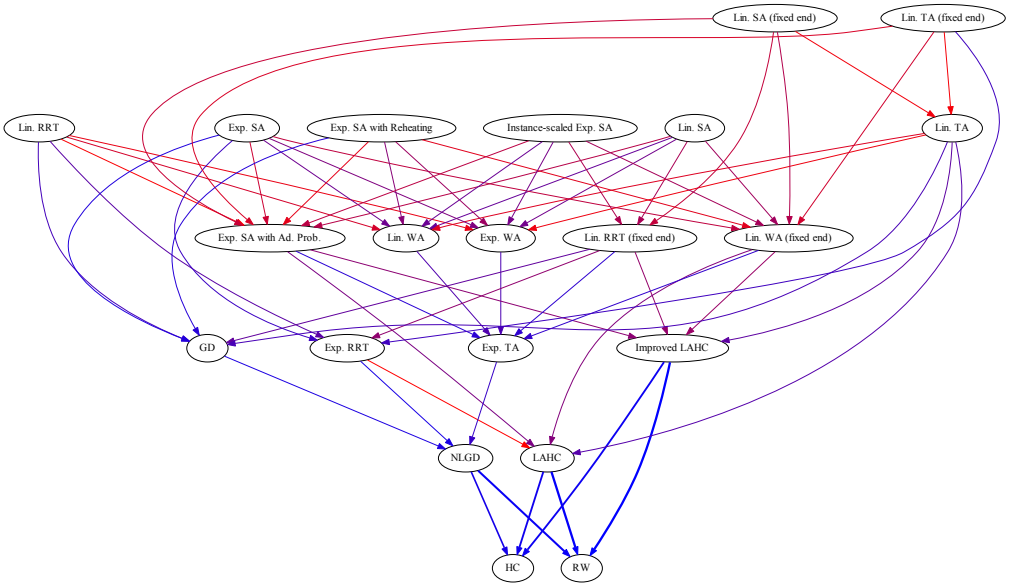


Figure 7.3: Graph based on the Wilcoxon test for problem CMST and using the deviation between the average run and the overall best. Methods on top dominate methods on the bottom. Bluer and thicker arcs mean that the difference in deviation is greater.

Figure 7.4: Graph based on the Wilcoxon test for problem CVRP and using the deviation between the average run and the overall best. Methods on top dominate methods on the bottom. Bluer and thicker arcs mean that the difference in deviation is greater.

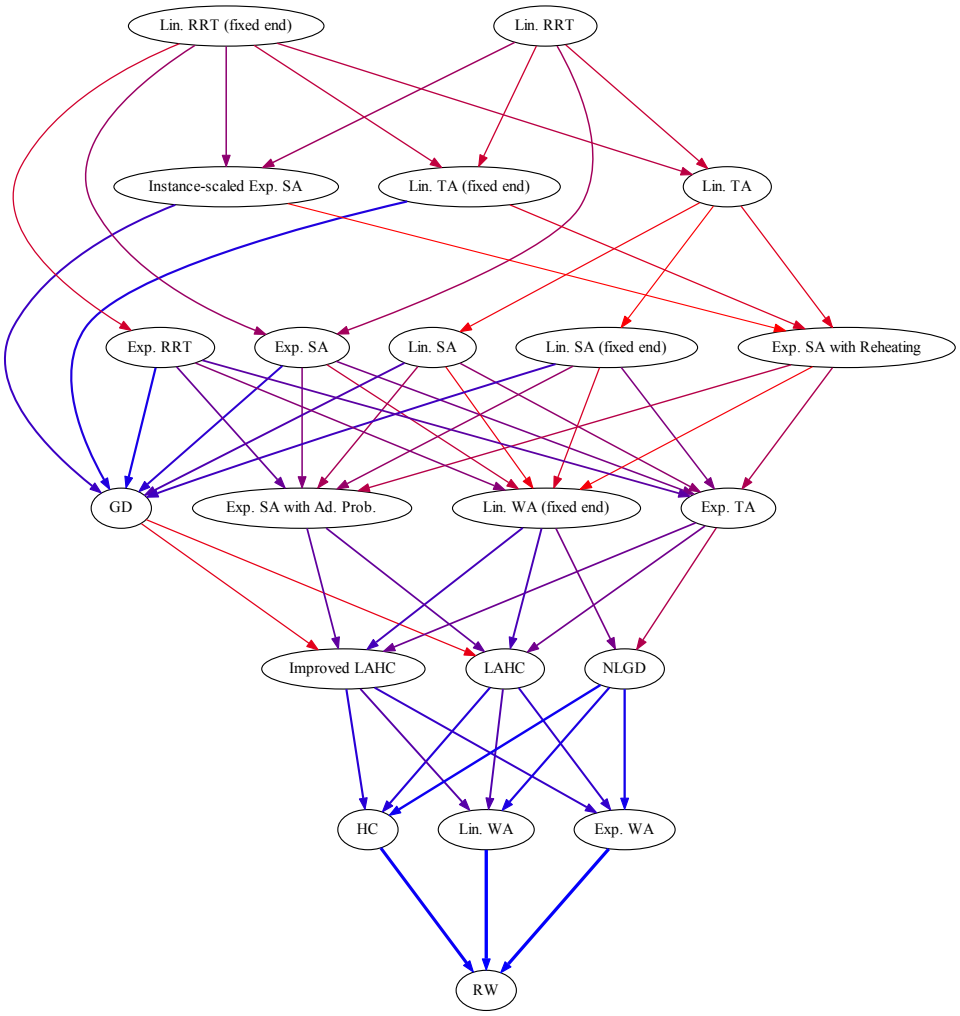


Figure 7.5: Graph based on the Wilcoxon test for the Simple LNS for CVRP and using the deviation between the average run and the overall best. Methods on top dominate methods on the bottom. Bluer and thicker arcs mean that the difference in deviation is greater.

calculated for each run: the iteration of the last accepted move, the iteration of the last improved best found, the longest streak of rejected moves, the maximum distance between accepted moves, the total distance between accepted solutions, the maximum distance from the initial solution, the number of solutions accepted, the number of times that the best solution was improved, the number of times that the current solution was improved, the relative average accepted objective function value, and the relative average rejected objective function value. The distance between solutions is calculated as the Hamming distance where each edge is represented by a binary digit. The relative objective function value of a move is calculated as the ratio of the new solution and the old solution, so that values greater than one imply worsening moves.

Regression coefficients are determined using the method of ordinary least squares, which implies minimising the sum of the squares of the error terms $\sum_{i=1}^{N} \varepsilon_i^2$ where N is the number of observations, and the model is:

$$y_i = \lambda_0 + \lambda_1 x_{i,1} + \ldots + \lambda_{11} x_{i,11} + \varepsilon_i \quad i = 1, \ldots, N \tag{7.4}$$

with λ_0 being the intercept and $\lambda_1, \ldots, \lambda_{11}$ the parameters to estimate, y_i the observed values of the dependent variables and x_{ij} the observed values of the independent variables.

After running the regression analysis with all the independent variables, the variables that did not have regression coefficients significantly different from 0, at a 0.05 significance level, were removed and the regression repeated. To better gauge the relative importance of the different independent variables, the values of each of them were normalized by subtracting the population mean and dividing by the standard deviation.

The results of the regression analyses are summarised in Table 7.4. A negative regression coefficient means that a higher value of the corresponding independent variable is associated with a better performance. There are some differences between the results for each of CMST, CVRP and Simple CVRP, but also some consistent similarities: a worse performance is associated with high values of the iteration of the last accepted solution and the iteration of the last improvement of the best solution found. This may indicate that an intensification phase with a high probability of rejecting solutions should not be delayed for too long. Higher values for the length of the longest streak of rejected moves is associated to a worse performance, meaning that move acceptance criteria should be designed so as to avoid being stuck in the same solution for too many iterations. Increased values of the maximum distance between accepted moves are associated with improved performance. This may suggest that move acceptance should not be based solely on the quality of the resulting solution but also, to some extent, on how similar the new solution is to the current one. For the other independent variables, the results are less clear. The relative average objective function value of rejected solutions is found to influence the performance: as the regression coefficients are negative, good performance is found when the solutions rejected are worse. This could simply mean that it is good that those solutions are not accepted. There is also a trend that a higher number of accepted solutions leads to better performance.

	CMST		CVRP		Simple LNS for CVRP	
Independent Variable	Regression Coeff.	p-value	Regression Coeff.	p-value	Regression Coeff.	p-value
(Intercept)	0.005	—	0.006	—	0.013	—
Iter. Last Accept.	0.005	0.000	0.001	0.020	0.005	0.001
Iter. Last Impr. Best	0.003	0.000	0.001	0.000	0.002	0.000
Longest Reject Streak	0.005	0.001	0.002	0.001	0.005	0.001
Max. Dist. btw Accepted	−0.001	0.000	−0.006	0.000	−0.004	0.000
Max. Dist. from Init.	−0.002	0.001	0.009	0.000	0.004	0.000
Tot. Dist. by Accept.					0.001	0.000
Num. Sol. Accept.	−0.001	0.001			−0.001	0.000
Num. Sol. Impr. Best	0.007	0.000	−0.002	0.000	0.004	0.000
Num. Sol. Impr. Current					−0.003	0.000
Rel. Avg. Accept. Obj.	−0.002	0.000	0.011	0.000	0.013	0.000
Rel. Avg. Reject. Obj.			−0.012	0.000	−0.016	0.000

Table 7.4: Regression analysis results from CMST, CVRP and Simple LNS for CVRP. The dependent variable is the deviation between the average run and the overall best. The table only includes values for the significant independent variables.

7.8 Conclusions

Many different move acceptance criteria are available when implementing a heuristic based on the ALNS framework. These include Hill Climbing (HC), Random Walk (RW), Late Acceptance Hill Climbing (LAHC), Threshold Acceptance (TA), Simulated Annealing (SA), Great Deluge (GD), Non-Linear Great Deluge (NLGD), and Record-to-Record Travel (RRT). In addition, a new criterion called Worse Accept (WA) was introduced in this paper. Based on current literature, it is difficult to ascertain whether any of these are better choices than the others in the context of the ALNS framework.

We presented a large computational study, where the results point out that HC and RW are bad choices for a move acceptance criterion in three different settings, including an ALNS for a capacitated minimum spanning tree problem (CMST), an ALNS for the capacitated vehicle routing problem (CVRP), and a simple LNS for the CVRP. In the same tests, SA, RRT, and TA performed best, whereas LAHC, GD, NLGD, and WA performed better than HC and RW but worse than SA, RRT, and TA. Several sub-variants of these move acceptance criteria were also tested and analyzed.

It was found that the effect of using different move acceptance criteria can be fairly large, affecting the average gap to the best known solutions by more than 0.5 percentage points. Multiple linear regression was used to find relationships between the performance of the move acceptance criteria and statistics gathered during the runs. Better performance is associated with 1) accepting the last move in an early iteration, 2) finding the last improvement of the best solution in an early iteration, 3) not having long streaks of rejecting moves, 4) having a short maximum distance between accepted solutions, and 5) having high relative average objective function values for rejected solutions.

We also observed that linear versions, where the crucial parameter for acceptance changes linearly from a start to an end value, of many well-established criteria fare better than or similarly to the standard exponential versions. Furthermore, the linear versions have the advantage that the end value for the aforementioned parameter can often be fixed to zero.

Such an approach does not lead to deteriorated solution quality, but reduces the dimension of the parameter space by one.

To summarise, we can make the following reccommendations for implementing an ALNS heuristic:

- Use an acceptance criterion based on SA, TA, or RRT. If time permits, it may pay off to attempt all three.
- Use a linear acceptance parameter function endiing at zero: this reduces the number of parameters by one and makes tuning easier, without sacrificing on the solution quality.

Of course the conclusions drawn from the experiments described in this paper will not necessarily apply to all other implementations, and we expect these reccommendations to be most useful when solving problems related to the CVRP and the CMST.

Bibliography

[1] J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072, 1990.

[2] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated f-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[3] E.K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *PATAT 2008 Conference, Montreal, Canada*, 2008.

[4] E.K. Burke and Y. Bykov. The late acceptance hill-climbing heuristic. Technical Report CSM-192, University of Stirling, Tech. Rep, 2012.

[5] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. John Wiley & Sons, 1979.

[6] D. Connolly. General purpose simulated annealing. *Journal of the Operational Research Society*, 43(5):495–505, 1992.

[7] E. Demir, T. Bektaş, and G. Laporte. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2): 346–359, 2012.

[8] G. Dueck. New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104:86–92, 1993.

[9] G. Dueck and T. Scheuer. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90: 161–175, 1990.

[10] F. Furini, E. Malaguti, and A. Santini. Exact and heuristic algorithms for the partition colouring problem. *Submitted to Computers and Operations Resarch*, pages 1–17, 2016.

[11] B. L. Golden, E. A. Wasil, J. P. Kelly, and I. M. Chao. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In T. Crainic and G. Laporte, editors, *Fleet management and logistics*, pages 33–56. Springer, 1998.

[12] P. Grangier, M. Gendreau, F. Lehuédé, and L.-M. Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91, 2016.

[13] A. Hemmati and L.M. Hvattum. Evaluating the importance of randomization in adaptive large neighborhood search. *International Transactions in Operational Research*, 2016. forthcoming.

[14] V.C Hemmelmayr, J.-F. Cordeau, and T.G. Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers and operations research*, 39(12):3215–3228, 2012.

[15] F. Hutter, H.H Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[16] S. Irnich, P. Toth, and D. Vigo. The family of vehicle routing problems. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, chapter 1, pages 1–33. SIAM, 2nd edition, 2014.

[17] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[18] D. Landa-Silva and J.H. Obit. Great deluge with non-linear decay rate for solving course timetabling problems. In *Intelligent Systems, 2008. IS'08. 4th International IEEE Conference*, volume 1, pages 8–11. IEEE, 2008.

[19] G. Laporte, S. Ropke, and T. Vidal. Heuristics for the vehicle routing problem. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications*, chapter 4, pages 87–116. SIAM, 2nd edition, 2014.

[20] H. Lei, G. Laporte, and B. Guo. The capacitated vehicle routing problem with stochastic demands and time windows. *Computers and Operations Research*, 38(12):1775–1783, 2011.

[21] F. Li, B. Golden, and E. Wasil. Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers and Operations Research*, 32(5):1165–1179, 2005.

[22] L.F. Muller, S. Spoorendonk, and D. Pisinger. A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, 218(3):614–623, 2012.

[23] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8):2403–2435, 2007.

[24] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

[25] G.M. Ribeiro and G. Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 39 (3):728–735, 2012.

[26] Y. Rochat and É. D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics*, 1(1):147–167, 1995.

[27] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[28] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006.

[29] S. Ropke and A. Santini. Parallel adaptive large neighbourhood search. *in preparation*, 2016.

[30] Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. Measuring the impact of acceptance criteria on the Adaptive Large Neighbourhood Search metaheuristic. *Submitted to the Journal of Heuristics*, pages 1–25, 2017.

[31] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming)*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431, 1998.

[32] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M. De Aragao, and D. Andrade. Robust branch-cut-and-price for the capacitated minimum spanning tree problem over a large extended formulation. *Mathematical Programming*, 112(2):443–472, 2008.

[33] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, A. Subramanian, and T. Vidal. New benchmark instances for the capacitated vehicle routing problem. Technical report, UFF, Rio de Janeiro, Brazil, 2014. URL http://www.optimization-online.org/DB_HTML/2014/10/4597.html.