Erasmus University Rotterdam

Operations Research and Quantitative Logistics

Master's thesis

Robust Heuristics for a Multi-Constrained Stochastic Team Orienteering Problem with Heterogeneous Resources

Author: **Martin van Meerkerk (416298)**

Supervisors:
Dr. D. Huisman (EUR)
Dr. A.I. Barros (TNO)
Ir. A.A.F. Bloemen (TNO)

Second reader:
Dr. R. Spliet (EUR)

October 20, 2015

Abstract

In many practical applications, tasks have to be assigned to multiple resources. In a standard *Team Orienteering Problem* (TOP), the aim is to assign tasks to resources in such a way that the value of the scheduled tasks is maximized. We consider an extensive variant of the TOP, which incorporates constraints regarding time windows and endurance, and for which the heterogeneous resources are allowed to recharge at their depots during their routes. For this problem, we propose a heuristic that combines Lagrangian Relaxation and Column Generation (LRCG), and a Tabu Search heuristic. The latter heuristic can handle resource-dependent values and sets of recurrent tasks with a non-linear profit function as well.

Due to stochasticity of travel and service times, we need to take the robustness of solutions into account as well. For this purpose, we introduce the *risk function* as a new robustness concept, which aims to avoid the execution of tasks with a low value at the cost of having to remove a more important task in the future. As an additional robustness concept, we discuss setting a lower bound on the worst-case performance of a solution. We generate regular and robust solutions using the Tabu Search heuristic, applied to several test instances. For adjusting the generated solutions in real-time scenarios implied by simulations of travel and service times, we apply passive rescheduling policies.

The results show that incorporating the risk function leads to a significant improvement of the real-time performance. In particular, a higher objective value is obtained and fewer tasks have to be removed. Moreover, important tasks are performed in an early stage of the route more often. For some instances, combining the risk function with a requirement on the worst-case performance improves the real-time results even further. When applying the LRCG heuristic to the deterministic problem, improvements of the subgradient algorithm lead to better feasible solutions and stronger upper bound estimators for the optimal objective value.

Preface

After completing my master's studies in Mathematics, I decided to enroll for the master Operations Research and Quantitative Logistics at the Erasmus University of Rotterdam (EUR), combined with a part-time job at the Decision Support department of KLM. In a seminar at the EUR, the subject of this thesis was introduced. It was a pleasure to follow this seminar, thanks to the challenging problem and the successful cooperation with Ravi, Gerhard, and Thomas. I am grateful that TNO has given me the opportunity to continue working on this subject at the Military Operations department.

In particular, I would like to thank Ana and Axel for their useful feedback during our weekly meetings, especially in the phase of developing methodologies to tackle the problem. I would like to thank Dennis for his valuable comments on earlier versions of this thesis and for reminding me that I could only accomplish a part of my ideas in the limited time period. Furthermore, I would like to thank Remy for his time to read my thesis. Last, but not least, I would like to thank my wife Petra for supporting me during the entire thesis project.

Contents

Chapter 1

Introduction

In many practical applications, such as preparing military missions and scheduling police forces, tasks have to be assigned to resources. In addition to limited availability of resources, operational constraints and capabilities complicate the problem. For instance, a helicopter cannot perform traffic regulation tasks and might only have fuel for at most three consecutive hours. Moreover, tasks often have a limited time window in which they can be addressed. Therefore, often only a subset of the available tasks can be performed. Another complicating factor is the uncertainty of travel and service times in practice. For instance, travel times might vary due to traffic jams or bad weather conditions, and a task to find a certain person might turn out to be easier or harder than expected.

In our problem, to which we refer as the *Security Routing* problem, we want to assign tasks to heterogeneous resources in such a way that the resulting task values are maximized, while satisfying operational constraints like time windows. For each task, there is a set of resources capable of performing the task, each with a different expected service time. The resources also differ in travel speed, base depot, and endurance. We allow each resource to return to its depot for recharging, and to continue performing tasks after staying at the depot for some minimal rest time. However, we require all resources to be back at their depots at the end of some planning horizon. The problem is stochastic in the sense that travel and service times might deviate from their expected values.

We also study an extended problem, to which we refer as the *Extended Security Routing* problem, in which resource-dependent task values and recurrent tasks are incorporated. Resource-dependent task values reflect that one resource might be better in handling a certain task than another resource. For instance, patrolling an area by foot might be better than by helicopter if a part of the search area is indoor. Recurrent tasks reflect that it might be desirable to return to the same location multiple times. For instance, for observing whether a road surface changes over time in order to detect if mines have been placed

alongside the road, it is valuable to take pictures at different times at the exact same location. Since having only one or a few pictures might not be reliable, it makes sense to model the advantages of multiple visits as a non-linear convex profit function for such tasks.

Because of the stochastic nature of the Security Routing problem, an offline stage and an online stage can be distinguished. The offline stage refers to the initial assignment of tasks to resources. In addition to the total value of the scheduled tasks, the robustness of the solution has to be taken into account in this stage. In particular, the aim of the offline stage is to address as many important tasks as possible, while limiting the vulnerability of the solution against delays. The online stage refers to adjusting the generated solution in real-time, based on realizations of stochastic travel and service times. The aim of the online stage is to maximize the total value of completed tasks, while sticking to the original plan as much as possible.

In this thesis, we propose solution methods for both stages. Since the Security Routing problems extends the *Team Orienteering Problem (TOP)*, which was shown to be NP-hard in Golden et al. [1987], we devised heuristics rather than exact methods. For the offline stage, we propose a heuristic based on Lagrangian Relaxation and Column Generation (LRCG) based on Fisher [1981] and Pirkwieser and Raidl [2009], and a Tabu Search heuristic inspired by Xu and Chiu [2001] and Cordeau and Laporte [2005]. The LRCG heuristic addresses the original Security Routing problem, while the Tabu Search heuristic can be applied to the Extended Security Routing problem as well.

For the trade-off between the maximum task value and the robustness of solutions in the offline stage, we propose two robustness concepts. We introduce the concept of a *risk function*, of which the main idea is to allow the assignment of a task to a resource only if the risk of having to remove a more important task for this resource in real-time remains acceptably low, where the acceptance threshold depends on the ratio of the corresponding task values. We also discuss a *worst-case fraction*, which sets a requirement on the performance of a solution in the worst-case scenario.

We incorporate these robustness concepts in the Tabu Search heuristic for the offline stage. For investigating the effect of these robustness concepts in real-time, we generate offline solutions with and without these concepts. We compare their performance by applying rescheduling policies for the online stage to these solutions, based on real-time scenarios implied by simulations of travel and service times. We propose passive rescheduling policies in the sense that the only action is to delete tasks, in order to allow for a fair judgement of the effect of the robustness concepts.

The Security Routing problem is an extensive version of the TOP, which incorporates time windows for tasks, heterogeneity and recharging possibilities for resources, and stochasticity of travel and service times. These individual aspects have been discussed in other articles, but to the best of our knowledge,

the TOP induced by the combination of all of these aspects does not appear in existing literature. Resource-dependent task values and recurrent tasks have hardly been discussed in existing literature either, and only few articles propose solution approaches for both the offline and the online stage. Moreover, a major contribution of this thesis is the introduction of the risk function as a new robustness concept.

The outline of the thesis is as follows. In Chapter 2, we describe the Security Routing problem and the Extended Security Routing problem in more detail. In Chapter 3, we review the literature related to our problem. Afterwards, we proceed with solution methods and robustness concepts for the offline stage, of which an overview can be found in Figure 1.1.

In Chapters 4 and 5, we discuss the LRCG heuristic and the Tabu Search heuristic, respectively, of which only the latter is suitable for the Extended Security Routing problem. In these chapters, we focus on the optimal objective value in a deterministic setting. In Chapter 6, we propose two robustness concepts that address the stochasticity of travel and service times: the risk function and the worst-case fraction. In Chapter 7, we discuss how to generate robust solutions with the Tabu Search heuristic. In particular, we explain how to incorporate the robustness concepts. Afterwards, we switch to the online stage and discuss rescheduling policies in Chapter 8.

In Chapter 9, we describe the data instances that we used for our experiments to test the quality of the solution methods and robustness concepts. The results of these experiments are discussed in Chapter 10. Finally, in Chapter 11, we summarize the thesis and point to directions for further research.

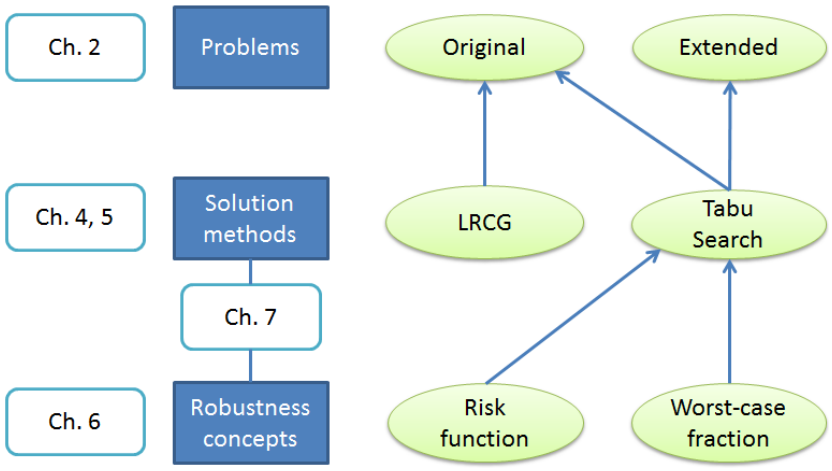

Figure 1.1: An overview of the solution methods and robustness concepts for the offline stage of the (Extended) Security Routing problem.

Chapter 2

Problem description

In this chapter, we formalize the Security Routing problem and the Extended Security Routing problem. The general idea is to assign tasks to heterogeneous resources in such a way that operational constraints are satisfied and resulting task values are maximized.

In Section 2.1, we introduce some terminology and formalize the constraints for the Security Routing Problem. Furthermore, we pay special attention to the stochasticity of travel and service times. In Section 2.2, we discuss the Extended Security Routing problem, in which resource-dependent task values and recurrent tasks appear.

2.1 Security Routing problem

A *location* j refers to either a task or a depot. For each resource, we determine a *route* that consists of an ordered sequence of locations. Moreover, we construct a *schedule* for visiting the locations in the route. We allow each resource to recharge at its depot and to resume performing tasks afterwards. We define a *trip* as the subroute between two depot visits, with no other depot visit in between. A *solution* consists of the set of routes and corresponding schedules for all resources. In Appendix A, a graphical representation of these notions is shown, as well as an overview of a selection of variables.

For each task $i \in \mathcal{I}$, a geographical location, a time window $[L_i, U_i]$, and a value v_i are specified. For each resource $r \in \mathcal{R}$, the travel speed, the base depot, the endurance E_r, and the minimal rest time R_r are specified. The endurance can be interpreted as the maximum trip duration. To each depot, a time window $[0, T]$ is associated, where T is the end of the planning horizon.

For each combination of a task i and a resource r, an expected service time $\bar{s}_{ir}$

is given, provided that resource r can perform task i. For a resource r and any two locations i and j, an expected travel time $\bar{t}_{ijr}$ can be computed based on the travel speed of r and the Euclidean distance between i and j.

For each resource, we need to determine a route and a corresponding schedule. Each schedule must satisfy the following constraints:

- *Time windows:* Each task i should start within its time window $[L_i, U_i]$.
- *Endurance:* Each trip should last at most E_r.
- *Minimal rest time:* Each rest period should last at least R_r.
- *Planning horizon:* The schedule should start and end within the planning horizon $[0, T]$.

A solution is feasible if and only if all of its routes have a feasible corresponding schedule and each task appears at most once in the solution.

Stochasticity of travel and service times

Notice that a feasible schedule that is based on expected travel and service times might become infeasible in real-time, due to the stochasticity of travel and service times. As in Evers et al. [2014a] and Ke et al. [2013], we assume interval data for the uncertain travel and service times, i.e. realizations will deviate at most some fixed percentage from their expected values. We denote the corresponding fractions by σ_{TT} and σ_{ST}, respectively. Because of the stochastic travel and service times, we divide the Security Routing problem into an offline stage and an online stage.

In the offline stage, we construct a feasible solution, of which the quality is determined by the total value of the scheduled tasks and the robustness of the schedules. The robustness of a solution can be determined by its performance in the online stage, in which the solution is adjusted based on a realized scenario with respect to travel and service times. The quality of a realized solution for the online stage is determined by the total value of the completed tasks and its closeness to the initial solution, which reflects the wish to stick to the original plan as much as possible in practice.

We assume that tasks can be aborted, but that we only obtain a value for completed tasks. We stress that in this stochastic problem, the performance of solutions in the online stage is more important than the theoretical objective value in the offline stage.

2.2 Extended Security Routing problem

Additional ingredients of the Extended Security Routing problem are resource-dependent task values and the presence of recurrent tasks.

As in the original problem, each task i has some base value v_i. However, in the extended problem, we let the actual value for the task depend on the resource to which it is assigned. If resource r can handle task i with a quality q_{ir}, then the value of assigning task i to resource r is now given by $v_{ir} = q_{ir}v_i$. Thus, assigning a task i to some resource r might yield a higher value than assigning task i to another resource r'.

We define a *recurrent task* to be a task that has the same location as some other task. We define a *recurrent set* to be a set of recurrent tasks with the same location. The character of recurrent tasks (e.g. road observation tasks) is such that it makes little sense to perform two recurrent tasks with the same location immediately after each other. Therefore, we require that recurrent tasks with the same location have disjoint time windows.

We assume that if k out of $|S|$ tasks of a recurrent set $S \subset \mathcal{I}$ are scheduled, then their total value is given by

$$v(S) = \frac{k}{2|S| - k} \sum_{i \in S} \sum_{r \in \mathcal{R}} x_{ir} v_{ir}.$$

In this formula, $x_{ir} = 1$ if task $i \in S$ is assigned to resource $r \in \mathcal{R}$, and 0 otherwise. Observe that visiting a subset of the tasks from a recurrent set contributes to the objective in a non-linear fashion, which reflects that partial information might not be that useful. See Figure 2.1 for a simple example in which all tasks from the recurrent set have the same value.

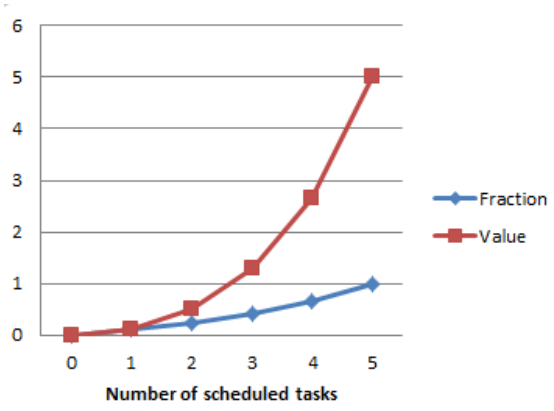

Figure 2.1: Example for the fractions $k/(2|S| - k)$ and the total value $v(S)$ for a recurrent set S with 5 tasks, each with value $v_{ir} = 1$.

Chapter 3

Literature review

In this chapter, we review the existing literature related to our problem and proposed methods. First of all, we classify the Security Routing problem in Section 3.1. In Section 3.2, we discuss existing literature on the additional ingredients for the Extended Security Routing problem, namely resource-dependent values and recurrent tasks. In Section 3.3, we discuss the combination of Lagrangian Relaxation and Column Generation. In Section 3.4, we cover concepts to handle stochastic travel and service times.

3.1 Classification of the problem

The Security Routing problem can be regarded as several closely related problems. On a global level, the problem can be seen as a variant of the Vehicle Routing Problem (VRP). In a recent survey of Rich VRPs by Caceres-Cruz et al. [2014], a wide variety of variants to the VRP is discussed. Archetti et al. [2014] identify the Orienteering Problem (OP), the Profitable Tour Problem, and the Prize Collecting Traveling Salesman Problem as the main three basic single-vehicle routing problems with profits. The authors state that little research has been devoted to multiple-vehicle routing problems with profits, such as the Team Orienteering Problem (TOP), introduced by Chao et al. [1996].

The Orienteering Problem was introduced by Tsiligirides [1984] and was shown to be NP-hard by Golden et al. [1987]. In the survey of the (T)OP by Vansteenwegen et al. [2011], special attention is given to the variants with time windows, abbreviated by (T)OPTW. Compared to the standard TOPTW, the additional ingredients of the Security Routing problem are endurance, multiple depots, heterogeneous resources, multi-trips, and stochastic travel and service times.

To the best of our knowledge, no research has been conducted yet to such an extensive variant of the TOPTW, although some articles consider a subset of

the additional problem aspects. Since the Security Routing problem extends the TOPTW, which it itself an extension of the OP, the Security Routing problem is NP-hard.

3.2 Extended Security Routing problem

In the Extended Security Routing problem, resource-dependent task values and recurrent tasks with a non-linear profit function are incorporated. In this section, we focus on the literature regarding these additional problem aspects.

3.2.1 Resource-dependent values

Most articles on the OP assume a fixed profit for visiting a node. However, some articles assume time-dependent profits (e.g. Tang et al. [2007]) or stochastic profits (e.g. İlhan et al. [2008]). To the best of our knowledge, resource-dependent profits are only addressed in the following two articles.

Xu and Chiu [2001] discuss a Field Technician Scheduling Problem (FTSP) in which resource-dependent values are constructed using base values for jobs and skill levels of technicians for certain jobs, analogous to the procedure discussed in Section 2.2. The authors propose a Local Search (see e.g. Lenstra [2003]) heuristic with the local moves Addition, Swap, Change, and Exchange. The latter two moves are concerned with moving tasks to different resources, which makes these moves suitable for handling resource-dependent values. Therefore, we adopt these four moves in our Tabu Search algorithm.

Flushing et al. [2014] assume that tasks can be partially completed. The completion rate of a task, and hence the associated collected profit, depends on the time an agent spends on the task and on the efficacy with which the agent can perform the task. Since we assume that a task only contributes to the objective if it is fully completed, this model does not fit our problem setting.

3.2.2 Recurrent tasks

Most articles on the OP assume a linear objective function, but there are some exceptions. In the *Generalized Orienteering Problem (GOP)* discussed by Wang et al. [2008], the objective is a nonlinear function of the node scores on different attributes, like beauty and cultural significance in a context of visiting cities. In the *Orienteering Problem with Variable Profits (OPVP)* in Erdogan and Laporte [2013], each node i has a collection parameter $\alpha_i \in [0, 1]$. For each node visit, which requires a fixed service time, a value of $100\alpha_i$ percent of the remaining profit is obtained. Erdogan and Laporte [2013] also discuss an alternative model,

similar to the one in Flushing et al. [2014] in the sense that the value obtained from a node depends on the time spent in the node. If either none or all of the recurrent tasks from a set should be performed, *Disjunctive Programming* techniques (see Grossmann [2002]) could be useful.

In the Satellite Orbit Problem in the ROADEF challenge of 2003, a schedule along Earth observation requests needs to be constructed for a single satellite. See ROADEF [2002] for a formal problem description. A request can be a single target or a polygon that consists of multiple 'strips'. Since partial images have little value, the profit of a polygon profit is a convex function. The problem is similar to an Orienteering Problem with Time Windows, but with a non-linear objective now. The aim is to select a subset of requests with maximum profit. Note that no stochasticity is involved in this problem.

Kuipers [2003] won the challenge by using a Simulated Annealing (see e.g. van Laarhoven and Aarts [1987]) algorithm, in which a candidate solution is constructed by removing/inserting a (limited) random number of random requests, while the number of removed/inserted strips in a polygon request is random as well. Cordeau and Laporte [2005] finished second by proposing a Tabu Search algorithm that allows infeasible solutions during search, with a self-adjusting penalty parameter that increases (decreases) while the solution is infeasible (feasible). They propose to restart at a perturbed version of the best known solution after some iterations without improvement, and to periodically reorder jobs within a route. We incorporate both procedures in our Tabu Search heuristic. We extend the perturbation procedure by adding a random number of recurrent tasks in order to help recurrent tasks to appear in the solution, inspired by Kuipers [2003].

Bianchessi et al. [2007] generalize this Tabu Search approach to multiple satellites performing multiple pre-fixed orbits and apply a branch-and-price approach to obtain upper bounds to their linear objective function. Their problem probably resembles our problem setting most of all existing articles, but still differs in several important respects. A major difference is that the orbits are pre-fixed regarding time and location, so that endurance constraints do not play a role here. In particular, recharging moments and departure times from a depot do not have to be chosen, as opposed to our multi-trip problem. Furthermore, no stochasticity is involved in their problem either and they assume homogeneous satellites with respect to speed, profits and service times. On the other hand, they do consider compulsory priority requests that and they require strips of a single polygon to be performed consecutively by a single satellite.

3.3 Lagrangian Relaxation and Column Generation

Combining Lagrangian Relaxation and Column Generation (LRCG) is a rather recent solution approach to solve large-scale problems. It has been applied successfully in several domains, including crew scheduling (see e.g. Kroon and Fischetti [2001] and Abbink et al. [2011]). In Huisman et al. [2005], the approach is discussed in detail within a Dantzig-Wolfe decomposition framework.

As in the LRCG approaches of Potthoff et al. [2010] and Veelenturf et al. [2014], we will formulate the Security Routing problem as a Set Covering Problem with additional constraints and the possibility to leave some tasks uncovered, and relax the covering constraints. For the Lagrangian Dual problem, we use the well known subgradient method (see e.g. Fisher [1981]). For the pricing problem, which will be an elementary shortest path problem with resource constraints, we use a labeling algorithm based on the one applied to a VRP with time windows in Pirkwieser and Raidl [2009].

3.4 Handling stochastic travel and service times

Due to stochastic travel and service times, we need to take robustness of the schedules into account when determining the quality of a solution, in addition to the task values. Two classical ways to handle uncertainties are Robust Optimization (see e.g. Ben-Tal et al. [2009]) and Stochastic Programming (see e.g. Birge and Louveaux [2011]). For Robust Optimization, a solution should be feasible for every scenario within a pre-defined scenario set. The aim of Stochastic Programming is to maximize the expected profit over all possible scenarios. In Evers et al. [2012], Robust Optimization and Stochastic Programming are compared for the OP under uncertainty.

Even though stochastic travel and service times are very common in practice, little research has been conducted to (T)OP variants under uncertainty. Both Evers et al. [2014a] and Evers et al. [2014b] consider an OP with stochastic travel and service times, which they solve using a Robust Optimization approach and a Stochastic Programming approach, respectively. To the best of our knowledge, no research has been conducted yet to such an extensive variant of the TOP with stochastic travel and service times as the Security Routing problem.

In Breugem et al. [2015], the concept of *t-recoverability* is introduced, which builds on the concepts of recoverable robustness (see Liebchen et al. [2009]) and quasi-robustness (see Veelenturf et al. [2014]). A route is defined to be t-recoverable if it has a feasible mean-case schedule, and a feasible worst-case schedule for a reduced route that is obtained by deleting all tasks below a certain threshold value. However, this concept has some limitations.

First of all, the concept was designed for a problem with strict priority distinctions, in the sense that a task with value v_i is more important than the set of all tasks with a lower value than v_i. For more general values, the construction of a reduced route makes less sense, since the deleted tasks might only be slightly less important than the other tasks.

Furthermore, the construction of two different schedules causes some problems when translating the concept to a rescheduling policy. When we use the mean-case schedule, any decision to perform a lower priority task can lead to infeasibilities for subsequent high priority tasks, even with the smallest delay. When we start with the worst-case schedule for the reduced route, it is not obvious that low priority tasks can be reinserted even in favourable scenarios, since the mean-case schedule can be significantly different.

A major contribution of this thesis is the introduction of the *risk function*. The main idea is to skip a task with low value if the risk of having to skip a more important task in the future is too high. This concept can be applied to any value structure and has an immediate translation to a rescheduling policy. Thus, the risk function can be regarded as a generalization and improvement of the t-recoverability concept.

We also propose the use of a *worst-case fraction*, which sets a lower bound on the obtained objective value in the worst-case scenario in real-time. It can be regarded as a lower bound on a normalized multiple-route version of the Profit At Certainty (with $\alpha = 1$) as defined in Evers et al. [2014b].

Chapter 4

LRCG for the original problem

For the original Security Routing problem, a procedure that combines Lagrangian Relaxation and Column Generation (LRCG) was proposed by Breugem et al. [2015]. In addition to generating feasible solutions, this approach is used to generate upper bound estimators on the optimal objective value. In this chapter, we build further on this LRCG approach and propose some adjustments that might improve these results.

In Lagrangian Relaxation (see e.g. Fisher [1981]) of a Linear Programming (LP) problem, a complicating constraint set is incorporated in the objective of the LP problem. Column generation (see e.g. Desaulniers et al. [2006]) is used to handle problems with a huge solution space, by only considering promising columns that might improve the current objective value. These two methods have been successfully combined into a LRCG approach in several articles, including Kroon and Fischetti [2001] and Abbink et al. [2011].

In Section 4.1, we provide an integer linear programming (ILP) formulation for the original Security Routing problem. As the number of possible feasible routes explodes for larger instances, it is untractable to solve the ILP by standard procedures, which motivates the use of the LRCG approach. In Section 4.2, we discuss the Lagrangian Subproblem and the Lagrangian Dual problem. In Section 4.3, we discuss the restricted master problem and the pricing problem. In Section 4.4, we explain how the LRCG approach can be used to obtain upper bound estimators on the maximum total task value that can be obtained. In Section 4.5, we discuss a greedy heuristic to obtain feasible solutions for the original ILP, which is necessary because solutions to Lagrangian Subproblems are generally infeasible for the original ILP.

The LRCG approach we propose is different from the one in Breugem et al.

[2015] in several aspects. First of all, the parts related to the t-recoverability concept are removed. Furthermore, we introduce an adapted subgradient algorithm for the Lagrangian Dual problem.

We stress that we will only combine the Tabu Search algorithm (see Chapter 5) with the robustness concepts and rescheduling policies that we propose in Chapters 6 and 8. Thus, the focus of this LRCG approach is on the deterministic version of the original Security Routing problem. Furthermore, we ignore the planning horizon constraint in this chapter.

4.1 ILP formulation

In the Security Routing problem, we need to assign a feasible route to each resource, while maximizing the total value of the scheduled tasks. Let $\mathcal{R}$ be the set of resources and let K_r be the set of feasible routes for resource $r \in \mathcal{R}$. Let $\mathcal{I}$ be the set of tasks and let a value v_i be defined for each task $i \in \mathcal{I}$. The binary decision variables x_{kr} indicate which routes appear in the solution. In particular, x_{kr} is 1 if route k is assigned to resource r, and 0 otherwise. Let a binary parameter a_{ikr} be 1 if task i is assigned to a route k for resource r, and 0 otherwise. Let a penalty z_i be 1 if task i remains unscheduled, and 0 otherwise. Using this notation, we can formulate the Security Routing problem as the following ILP:

$$\min \quad \sum_{i \in \mathcal{I}} v_i z_i \tag{4.1}$$

$$\text{s.t.} \quad \sum_{r \in \mathcal{R}} \sum_{r \in K_r} a_{ikr} x_{kr} + z_i \geq 1 \quad \forall i \in \mathcal{I} \tag{4.2}$$

$$\sum_{k \in K_r} x_{kr} = 1 \quad \forall r \in \mathcal{R} \tag{4.3}$$

$$x_{kr} \in \{0,1\} \quad \forall r \in \mathcal{R}, k \in K_r \tag{4.4}$$

$$z_i \in \{0,1\} \quad \forall i \in \mathcal{I} \tag{4.5}$$

The objective (4.1) indicates that we minimize the value of the tasks that remain unscheduled, which is equivalent to maximizing the value of the scheduled tasks. Constraints (4.2) indicate that each task is assigned to at most one resource. Constraints (4.3) inidicate that exactly one route is assigned to each resource.

Notice that the ILP is formulated as a Set Covering Problem with additional constraints, with the possibility to leave some tasks unscheduled, as in Potthoff et al. [2010].

4.2 Lagrangian Relaxation

We apply Lagrangian relaxation to turn the ILP into a Lagrangian Subproblem that is easy to solve, as we discuss in Section 4.2.1. The Lagrangian Subproblem is repeatedly solved for different sets of Lagrangian multipliers in the Lagrangian Dual problem, which we discuss in Section 4.2.2.

In the LRCG algorithm, we iteratively solve the Lagrangian Dual problem and generate new columns. In particular, both the Lagrangian Subproblem and the Lagrangian Dual problem are solved using a restricted subset of columns. In this section, suppose that in the n-th iteration in the LRCG algorithm, a subset of columns K_r^n for each resource r is given, of which we denote the union by $K^n = \cup_{r \in \mathcal{R}} K_r^n$.

4.2.1 Lagrangian Subproblem

The ILP is difficult to solve, due to the constraints that each task should be assigned to at most one resource. Therefore, we relax constraints (4.2) in a Lagrangian way, which yields the following Lagrangian subproblem for a given set of Lagrangian multipliers $\mathbf{\Lambda} = (\lambda_i)_{i \in \mathcal{I}}$:

$$\begin{aligned} \Theta_n(\mathbf{\Lambda}) = \min \quad & \sum_{i \in \mathcal{I}} \lambda_i + \sum_{i \in \mathcal{I}} (v_i - \lambda_i)\, z_i \\ & + \sum_{r \in \mathcal{R}} \sum_{k \in K_r^n} \left(- \sum_{i \in \mathcal{I}} \lambda_i a_{ikr} \right) x_{kr} \qquad (4.6) \\ \text{s.t.} \quad & (4.3) - (4.5) \end{aligned}$$

Observe that the reduced costs of a route $k \in K_r$ are given by:

$$\zeta_{kr}(\mathbf{\Lambda}) = - \sum_{i \in \mathcal{I}} \lambda_i a_{ikr}. \qquad (4.7)$$

The optimal solution for the Lagrangian subproblem can be found easily. To each resource r, we assign the route with minimal reduced cost, which is equivalent to the route with maximum task value. That is, we set $x_{\hat{k}r} = 1$ for the route $\hat{k} = \arg\min\{\zeta_{kr}(\mathbf{\Lambda}) \mid k \in K_r^n\}$, and $x_{kr} = 0$ for all $k \in K_r^n \backslash \{\hat{k}\}$. Furthermore, we set $z_i = 1$ if and only if $v_i - \lambda_i < 0$.

In general, solutions to the Lagrangian Subproblem will tend to include tasks with high value in multiple routes. Therefore, the solutions are typically infeasible for the original ILP, due to violation of the constraints (4.2).

4.2.2 Lagrangian Dual problem

As the Lagrangian Subproblem is a relaxation of the original ILP, solutions to the Lagrangian Subproblem can be used as a lower bound for the objective value that can be obtained in the original ILP, restricted to the current subset of columns K^n in the n-th LRCG iteration. The aim of the Lagrangian Dual problem is to find the strongest of these lower bounds:

$$\Theta_n^* = \max_{\mathbf{\Lambda} \geq \mathbf{0}} \Theta_n(\mathbf{\Lambda}) \tag{4.8}$$

There is no analytical solution for this problem to find the maximum objective value for a Lagrangian Subproblem over all possible multipliers $\mathbf{\Lambda} \geq \mathbf{0}$. Therefore, we are forced to approximate Θ_n^* by an estimator $\hat{\Theta}_n$. The approximation is made by using a *subgradient algorithm*, based on Fisher [1981]. The procedure is shown in Algorithm 1, of which we explain the steps below.

After initializing the parameters (Step 0), we repeatedly solve the Lagrangian subproblem for different Lagrangian multipliers $\mathbf{\Lambda}^j = (\lambda_i^j)_{i \in \mathcal{I}}$ (Step 1). The multipliers for the next iteration are determined using the subgradient y^j that indicates in which direction the optimum can be found (Step 2).

In Breugem et al. [2015], the step size parameter α was decreased every time that no improvement of $\hat{\Theta}_n$ was found for γ consecutive iterations, which is the classical approach (see e.g. Fisher [1981]). However, a drawback of this procedure is that the step size might become very small too quickly. Since a small step size leads to small differences between Lagrangian multipliers over different iterations, having a small step size implies that $\hat{\Theta}_n$ will probably not be improved significantly anymore, and the subgradient algorithm is terminated. If this happens too quickly, this leads to a significant underestimation of Θ_n^*.

Caprara et al. [1999] propose to update α periodically after p iterations, based on the best and worst obtained lower bounds in this period. Inspired by this procedure, we propose a periodic update procedure based on the obtained lower bounds as well (Step 3), which allows us to guide the search process better. After each period of p iterations, we check whether the obtained lower bound has improved sufficiently, and we only decrease α if this is not the case.

Note that this is different from just increasing the parameter γ. In fact, increasing γ could cause that the step size parameter remains too high too long, as the obtained lower bounds often fluctuate over different iterations, especially when the step size is still large. By comparing the average lower bounds over the first few iterations and the last few iterations, we aim to compensate for this effect.

Another measure that protects against terminating the search too early and underestimating Θ_n^*, is taken in the termination criteria (Step 4). If α drops below a threshold value ε, we only stop the search if there was not enough improvement over the last p' iterations.

Algorithm 1 Subgradient algorithm to solve Lagrangian Dual

Step 0: *Initialization*
Initialize parameters $j_{max}, \delta, \varepsilon, \mathbf{\Lambda}^0, UB, acceptPerc$.
Initialize parameters $\alpha^0, p, p', avgIts, minImprove, decrFactor$.

Step 1: *Solve Lagrangian subproblem*
Compute lower bound $\Theta_n(\mathbf{\Lambda}^j)$ with optimal solution x^j and z^j.
Compute subgradient $y_i^j = 1 - \sum_{r \in \mathcal{R}} \sum_{k \in K_r^n} a_{ikr} x_{kr}^j - z_i^j$

Step 2: *Compute Lagrangian multipliers*
Compute $\lambda_i^{j+1} = \lambda_i^j + \alpha^j \frac{UB - \Theta_n(\mathbf{\Lambda}^j)}{\sum_{i \in \mathcal{I}} (y_i^j)^2} y_i^j$

Step 3: *Update step size parameter*
if $j = kp$ for some $k \in \mathbb{N}$ **then**
  Compute $begin = \sum_{q=(k-1)p}^{(k-1)p+avgIts-1} \Theta_n(\mathbf{\Lambda}^q)$.
  Compute $end = \sum_{q=kp-avgIts+1}^{kp} \Theta_n(\mathbf{\Lambda}^q)$.
  Compute $perc = sign(begin) * (end - begin)/begin$.
  if $perc < minImprove$ **then**
    Set $\alpha^{j+1} = decrFactor * \alpha^j$.
  end if
end if

Step 4: *Termination criteria*
if $\sum_{i \in I} (y_i^j)^2 \leq \delta, j \geq j_{max}$ or $\frac{|UB - \Theta_n(\mathbf{\Lambda}^j)|}{\Theta_n(\mathbf{\Lambda}^j)} < acceptPerc$ **then**
  Stop.
else
  if $\alpha^j < \varepsilon$ **and** $j \geq p'$ **then**
    Compute $avgBegin = \sum_{q=j-p'+1}^{j-p'+avgIts} \Theta_n(\mathbf{\Lambda}^q)/avgIts$.
    Compute $maxEnd = \max\{\Theta_n(\mathbf{\Lambda}^q) \mid q > j - p'\}$.
    Compute $perc = sign(avgBegin) * (maxEnd - avgBegin)/avgBegin$.
    if $perc < minImprove$ **then**
      Stop.
    end if
  end if
  Return to step 1.
end if

4.3 Column generation

Due to the large number of possible routes for large instances, it is untractable to solve the ILP over all possible routes. Therefore, we apply Column Generation in order to only consider the most promising columns. Below, we give a global overview of the LRCG algorithm, after which we discuss the pricing problem in more detail in Section 4.3.1.

In the n-th column generation iteration, we solve the Restricted Master Problem (RMP) of the relaxed ILP over the current subset $K^n = (K_r^n)_{r \in \mathcal{R}}$ of generated columns. Afterwards, we extend K^n by generating new promising columns in the pricing problem, in which the quality of columns is based on Lagrangian multipliers λ obtained in the Lagrangian Dual problem. We terminate the LRCG algorithm if we are unable to generate additional promising columns in the pricing problem.

4.3.1 Pricing problem

In the pricing problem, we aim to find columns that can form a valuable addition to the current subset of columns. After describing how to measure the quality of a column, we define when we consider a column to be profitable. Afterwards, we explain how to construct and solve the pricing problem.

Profitable columns

The quality of a column can be measured by its reduced costs with respect to the Lagrangian multipliers λ that correspond to the strongest lower bound obtained in the Lagrangian Dual problem. Recall from equation (4.7) that the reduced costs of a route k for resource r are given by $\zeta_{kr}(\mathbf{\Lambda}) = -\sum_{i \in \mathcal{I}} \lambda_i a_{ikr}$.

As we consider a minimization problem, we aim for the columns with minimal reduced costs. As in Potthoff et al. [2010], we consider a generated column k for resource r to be profitable if and only if k is better than the current columns for resource r, i.e. if $\zeta_{kr}(\mathbf{\Lambda}) < \zeta_{k'r}(\mathbf{\Lambda})$ for all routes k' in the current subset K_r^n.

Constructing the pricing problem graph

The pricing problem is separable per resource and can be modeled as an elementary shortest path problem with resource constraints (ESPPRC). In particular, we need to construct a graph in which a path corresponds to a route, which should satisfy the constraints regarding time windows and endurance.

We search for a shortest path in a graph in which the nodes represent tasks and depots. A path through this graph corresponds to a route, as it defines a

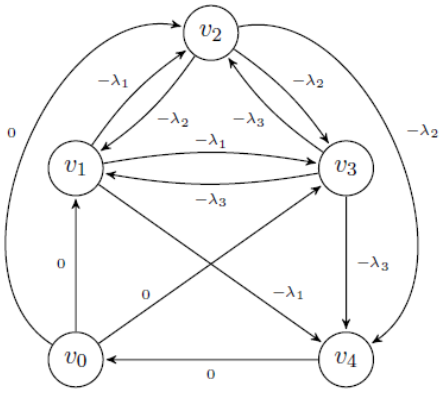

Figure 4.1: Taken from Breugem et al. [2015]: An example of a pricing problem graph, with depot nodes v_0, v_4 and task nodes v_1, v_2, v_3.

sequence of locations. This route should be feasible with respect to time windows and endurance. Furthermore, we require each path to start at the node v_0 that represents the starting depot, and to be elementary in the sense that each task is visited at most once. The number of visits to depot nodes is not restricted, in order to allow for recharging depot visits in multi-trip routes.

In each path, an arc (i, j) indicates that node j is visited after node i in the corresponding route. By setting $c_{ij} = -\lambda_i$ for tasks i and $c_{ij} = 0$ for depots, we ensure that the length of the path is equal to the reduced costs of the corresponding route. Consequently, the shortest path in the graph corresponds to the route with minimal reduced costs. Since we require paths to be elementary in the sense that each task is visited at most once, the shortest path is well-defined, even though the arc weights c_{ij} can be negative.

In Figure 4.1, we show a small example of a graph for the pricing problem. In this example, the aim is to find an elementary shortest path from depot node v_0 to the duplicate depot node v_4, of which the corresponding route must be feasible with respect to time windows and endurance.

Solving the elementary shortest path problem with resource constraints

For solving the above problem, we use a dynamic programming approach based on Pirkwieser and Raidl [2009]. In this algorithm, partial paths are extended using a labelling approach. Since the problem is NP-hard, due to the constraints on time windows and endurance, we use a heuristic dominance rule to eliminate unpromising partial paths in an early stage. Note that this might lead to a sub-optimal result, in the sense that we miss the route with minimal reduced costs, due to eliminating a corresponding partial path too early.

To each partial path, we assign a label that contains information about the corresponding route. When we extend the partial path $[p]$ by an arc (v_i, v_j), we update this label by the equations below:

$$V_j([p] \cup v_j) = V_i([p]) \cup v_j \tag{4.9}$$
$$C_j = C_i + c_{ij} \tag{4.10}$$
$$arr_j = arr_i + wait_i + \bar{s}_{ir} + \bar{t}_{ijr} \tag{4.11}$$
$$wait_j = \max\{0, L_j - arr_j\} \tag{4.12}$$
$$start_j = \max\{arr_j, L_j\} \tag{4.13}$$
$$W_j = W_i + wait_j \tag{4.14}$$
$$F_j = \min\{F_i, W_j + (U_j - start_j)\} \tag{4.15}$$
$$D_j = start_j - \min\{F_j, W_j\} \tag{4.16}$$

For the obtained partial path with end node v_j, V_j and C_j represent the visited nodes and the cost of this partial path, respectively. Assuming that we leave depot v_0 at time 0, $arr_j, wait_j$ and $start_j$ represent the arrival time, waiting time, and starting time at node j, respectively.

W_j represents the total waiting time in the current trip to j. F_j represents the *forward time slack*, introduced in Savelsbergh [1992], which is the maximum duration by which the previous depot visit can be extended. By reducing waiting times, this can lead to a shorter duration D_j of the current trip to j. If the node v_j is a depot, a trip is completed. After recharging at this depot, the partial path may be extended further by an arc (v_j, v_k) for some task v_k. Therefore, we reset the labels W_j, F_j and D_j at the end of each trip.

If v_j is a task, we do not consider extension by the arc (v_i, v_j) for partial paths $[p]$ that already contain task v_j, in order to avoid visiting the same task twice. The extension of $[p]$ by the arc (i, j) is infeasible if its time window is violated or if the duration of the current trip to j exceeds the endurance of the resource, i.e. if $arr_j > U_j$ or if $D_j > E_r$. In that case, we delete the obtained label.

In order to keep the number of stored labels limited, we use dominance rules. If a partial path $[p]^1$ dominates a partial path $[p]^2$ that ends at the same node v_j, we delete the label for $[p]^2$. We use the following rules to indicate that $[p]^1$ dominates $[p]^2$:

1. $[C_j]^1 \leq [C_j]^2 \wedge [D_j]^1 \leq [D_j]^2$
2. $[C_j]^1 \leq [C_j]^2 \wedge [D_j]^1 \leq [D_j]^2 \wedge [start_j]^1 \leq [start_j]^2$

Rule 1 states that $[p]^1$ dominates $[p]^2$ if it has lower cost and a lower duration. Rule 2 requires additionally that the starting time at node v_j is earlier for $[p]^1$. Since using an exact dominance rule leads to excessive running times, we use the heuristic dominance rules above, although this might imply that we miss the shortest path.

We summarize the complete algorithm for the pricing problem below. We keep

track of a set U of uncompleted partial paths, which initially only contains the empty path $\{v_0\}$. In each iteration of the dynamic programming algorithm, we choose a partial path $[p] \in U$. For each possible extension by an arc, we update the label of $[p]$ if the extension is feasible. Furthermore, for a given dominance rule, we check whether the new label is dominated by another partial path or dominates other partial paths itself. During this procedure, each partial path that ends at a depot is added to the set P of completed paths. When no uncompleted partial paths remain, the algorithm returns the path from P with the lowest costs. We add the corresponding route k to the current subset of columns for this resource if and only if the column is profitable. If no profitable column can be found when using rule 1, we might want to repeat the above process with the less strict dominance rule 2.

4.4 Upper bound estimator on total task value

Recall from Section 4.2.2 that in the n-th column generation iteration, each solution to a Lagrangian Subproblem is a lower bound for the objective value of the original ILP, restricted to the current subset of columns K^n. The approximation $\hat{\Theta}_n$ obtained by the subgradient algorithm is the strongest lower bound in the n-th iteration. However, $\hat{\Theta}_n$ cannot be used as a lower bound on the original ILP over *all* columns, as the addition of new columns typically leads to a decreased value $\hat{\Theta}_{n+1} < \hat{\Theta}_n$ in a subsequent iteration.

If we would be able to guarantee that we always find the route with minimal reduced costs in the pricing problem, we could use the obtained Lagrangian Dual solution $\hat{\Theta}_N$ in the final column generation iteration N as a lower bound on the objective value of the original ILP over all possible routes. However, due to the use of heuristic dominance rules in the pricing problem, we cannot provide this guarantee. Nevertheless, $\hat{\Theta}_N$ is still useful as a lower bound estimator for the optimal objective value for the original ILP, in which we aim at minimizing the total value of all unassigned tasks. For the equivalent maximization problem of maximizing the total value of all scheduled tasks, we can use $(\sum_{i \in \mathcal{I}} v_i) - \hat{\Theta}_N$ as an *upper* bound estimator for the optimal objective value.

4.5 Generating feasible solutions

In each iteration of the subgradient algorithm for the Lagrangian Dual problem, we generate a solution by solving a Lagrangian Subproblem. In general, this solution is infeasible for the original ILP, due to violation of the covering constraints (4.2). Given Lagrangian multipliers $\mathbf{\Lambda}$, we use the following greedy procedure to obtain feasible solutions for the original ILP:

1. For the remaining resource $r \in \mathcal{R}$ with the lowest travel speed, choose the route k (i.e. set $x_{kr} = 1$) with minimal reduced costs $\zeta_{kr}(\mathbf{\Lambda})$.
2. For all tasks i in route k, set $\lambda_i = 0$. Delete resource r from $\mathcal{R}$. If $\mathcal{R}$ is still non-empty, return to Step 1. Otherwise, go to Step 3.
3. For each task i, set $z_i = 0$ if task i is assigned to some resource, and set $z_i = 1$ otherwise.

We apply this procedure once in every iteration of the subgradient algorithm. We sort the resources in increasing order of travel speed, since faster resources typically have a larger set of tasks which they can perform within their endurance. Therefore, faster resources are typically more flexible when searching for a route that complements the already chosen routes well.

In Step 2, we set $\lambda_i = 0$ for chosen tasks i in order to reflect that performing a task only contributes to the total task value at most once. However, this does not imply that tasks cannot appear multiple times in solutions obtained by the above procedure.

Chapter 5

Tabu Search for the extended problem

In this chapter, we discuss a Tabu Search algorithm that can be used for the Extended Security Routing problem. Obviously, it can handle the original Security Routing problem as well. The aim of this heuristic is to find good solutions quickly, especially compared to the running time for the LRCG algorithm.

A Tabu Search algorithm is an extended version of Local Search algorithms (see e.g. Lenstra [2003]). In Local Search algorithms, one iteratively updates a solution by searching for a better solution in the *neighborhood* of the current solution, which contains solutions that are close to the current solution in some (problem-specific) sense. A basic Local Search algorithm terminates when no improving solution can be found in the neighborhood.

In Tabu Search algorithms (see e.g. Glover and Laguna [2013]), moves that deteriorate the objective value of the solution are allowed. The introduction of a *tabu list* prevents cycling in the sense of returning to previously visited solutions quickly. Typically, this tabu list contains more specific information than entire solutions, because of memory issues. The aim of allowing deteriorating moves and avoiding cycling is to escape from local optima.

In Section 5.1, we discuss the neighborhood and the tabu list we use. In Section 5.2, we discuss the construction of a feasible schedule for a given route. In Section 5.3, we discuss a fast feasibility check for task insertions, suitable for our multi-trip and multi-constrained problem. Intensification and diversification mechanisms are discussed in Section 5.4. We finalize this chapter in Section 5.5 by describing some speed-ups of the algorithm.

In Algorithm 2, we provide a global overview of our Tabu Search heuristic to tackle the Extended Security Routing problem. This heuristic extends the one proposed in Breugem et al. [2015] significantly. We incorporate a reordering

procedure of tasks and we pay special attention to recurrent tasks in a random perturbation procedure. Regarding the schedules, we incorporate latest return times to avoid endurance and planning horizon violations, and we allow for extended depot visits in order to repair endurance problems. Moreover, we propose a fast feasibility check for task insertions, which includes a framework to handle the multi-trip character of routes.

Algorithm 2 Tabu Search heuristic for constructing offline solutions

```
1: Input: Task set 𝓘 and resource set 𝓡.
2: Input: List routeList that contains an empty route for each resource r ∈ 𝓡.
3: Output: An offline solution for the Extended Security Routing problem.

4: Initialize best solution in (sub)search: subBest = best = routeList
5: while maximum running time is not reached do
6:   for all local move types in the neighborhood (Section 5.1) do
7:     Sort moves in decreasing order of potential objective gain (Section 5.5).
8:     for all moves do
9:       if move has potential to yield best solution of iteration then
10:        Randomly choose insertion position and alternative route.
11:        Apply fast feasibility check for task insertion(s) (Section 5.3.1).
12:        if move is feasible then
13:          Construct a feasible schedule (Section 5.2)
14:          Update best solution of iteration and go to next local move type.
15:        else
16:          Try different insertion positions and alternative routes.
17:        end if
18:      end if
19:    end for
20:  end for

21:  Update current routeList according to the performed move.
22:  Update tabu list (Section 5.1) and alternative routes (Section 5.3.2).
23:  If improved, update subBest and best.

24:  if no improvement of subBest or best in last few iterations then
25:    if subBest has not been reordered before then
26:      Intensification: apply reorder procedure to subBest (Section 5.4.1).
27:    else
28:      Diversification: apply random perturbation to best (Section 5.4.2).
29:    end if
30:    Start new subsearch, with updated subBest and tabu list.
31:  end if
32: end while
```

5.1 Neighborhood and tabu list

We start the search with empty routes for each resource. In each iteration, we move to the best non-tabu solution in our neighborhood. This neighborhood consists of the following local moves, adopted from the Local Search heuristic in Xu and Chiu [2001]:

- *Addition:* add an unscheduled task to a route
- *Swap:* swap a scheduled task with an unscheduled task
- *Change:* change the resource by which a task is performed
- *Exchange:* exchange two tasks currently assigned to different resources

See Figure 5.1 for a graphical representation of the local moves. Observe that the moves Change and Exchange are perfectly suitable for handling resource-dependent values.

If task i leaves resource r after a Change or Exchange move, we declare all solutions that assign i to r tabu for some fixed number of iterations. Similarly, if task i leaves the set of unscheduled tasks after an Addition or Swap move, we declare leaving i unscheduled tabu for a while.

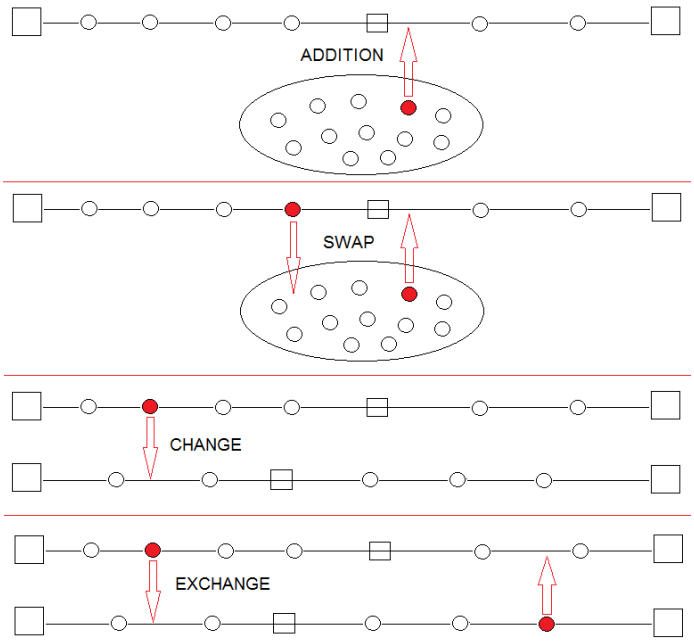

Figure 5.1: The local moves for the Tabu Search algorithm.

5.2 Schedules

In this section, we discuss the construction of a feasible schedule for a given route, assuming a *mean-case* scenario in which the realized travel and service times are equal to their expected values. First, we introduce some notation and corresponding formulas of time variables in Section 5.2.1. These time variables include *maximum shift variables* that will be used for a fast feasibility check for task insertion (see Section 5.3), and *latest return times* to a depot in order to avoid violations of constraints on endurance and the planning horizon. In Section 5.2.2, we describe an exact procedure to generate a feasible schedule, which includes the option to extend depot visits in order to repair endurance problems.

5.2.1 Time variables

We introduce several time variables that we determine when constructing a schedule. For the corresponding formulas below, recall from Section 2.1 that $\bar{t}_{ijr}$ denotes the expected travel time from location i to location j for resource r, $\bar{s}_{ir}$ denotes the expected service time for task i and resource r, and L_j denotes the release time of task j. Furthermore, recall that a location can refer to either a task or a depot.

For tasks, we assume that we never wait longer than required by its release time and that we immediately leave the task after completing it. This yields the following formulas for a task j that immediately follows a location i:

$$\text{Arrival time :} \quad arr_j = dep_i + \bar{t}_{ijr} \tag{5.1}$$

$$\text{Waiting time :} \quad wait_j = (L_j - arr_j)^+ \tag{5.2}$$

$$\text{Starting time :} \quad start_j = arr_j + wait_j \tag{5.3}$$

$$\text{Departure time :} \quad dep_j = start_j + \bar{s}_{jr} \tag{5.4}$$

The validity of above formulas for a depot j depends on its position. For the first depot visit, arrival and starting times are not defined, while we initialize its waiting time and departure time by setting them both to be zero. For any recharging depot j, the above formulas for $arr_j, start_j$ and dep_j are still valid when we let s_{jr} refer to the minimal rest time R_r. Initially, the waiting time is set to zero again, although it might be useful to set $wait_j > 0$ for a depot visit j that is not the final depot visit. For the final depot visit, the above formula for the arrival time is valid, while none of the other times is defined.

Additionally, we define the *maximum shifts* ms_i^{TW} and ms_i^{END} to be the maximum time by which the *starting time* of location i can be delayed, while preserving feasibility with respect to time windows and endurance, respectively. These maximum shifts can be regarded as the multi-trip and multi-constraint versions of the ones proposed by Vansteenwegen et al. [2009]. They are determined while constructing a schedule, as described in Section 5.2.2.

Latest return times

Recall that for a resource r with endurance E_r, each trip duration should be at most E_r. In practice, it is highly undesirable to violate this constraint. For instance, an unmanned aerial vehicle (UAV) should not go out of fuel above hostile territory. Although some buffers are used in practice, it is important to keep a sharp eye on trip durations. Similarly, violating the planning horizon is undesirable as well. For instance, resources with a secret mission at night should be back at the depot before dawn.

Therefore, we introduce *latest return times* for endurance and planning horizon, denoted by LRT^{END} and LRT^{HOR}, respectively. For both constraints, we require for each task that we can still arrive at the depot in time after completion of the task, assuming a worst-case travel time to the depot. Consider a trip from depot d to depot d', in a route for resource r with endurance E_r and planning horizon T. For each task i in the trip, the latest return times are given by:

$$LRT_i^{END} = dep_d + E_r - (1 + \sigma_{TT})\bar{t}_{id'r} \quad (5.5)$$

$$LRT_i^{HOR} = T - (1 + \sigma_{TT})\bar{t}_{id'r} \quad (5.6)$$

Recall from Section 2.2 that σ_{TT} represents the maximum deviation fraction for travel times. LRT_i^{END} and LRT_i^{HOR} represent the latest departure time from task i at which we can guarantee a timely arrival at depot d, with respect to endurance and the planning horizon, respectively. To be feasible with respect to endurance and the planning horizon, we require that a schedule satisfies $dep_i \leq LRT_i^{END}$ and $dep_i \leq LRT_i^{HOR}$ for all tasks i.

It does not suffice to put this restriction only on the last task of each trip for endurance and only on the last task of the route for the planning horizon. For succeeding tasks i and j in the same route, it is possible that $dep_i > LRT_i$ and $dep_j \leq LRT_j$. This can especially occur if σ_{TT} is large and i is far from the depot while j is not, as in Figure 5.2.

Due to imposing these additional requirements on a schedule, we can obtain a lower total task value in general. On the other hand, violating the constraints on endurance and the planning horizon becomes highly unlikely. Furthermore, performing tasks far from the depot at the end of a trip is discouraged, since these tasks have a large worst-case travel time to the depot.

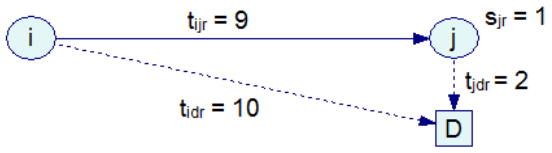

Figure 5.2: Tasks i and j in a trip to depot D.

5.2.2 Constructing a feasible schedule for a given route

We propose a constructive algorithm to generate a feasible schedule for a given route. In this algorithm, we start from scratch at time 0 and add the locations of the route one by one. After each addition, we check whether the schedule remains feasible. The algorithm is exact in the sense that if it fails to find a feasible schedule, there cannot exist a feasible schedule. In Algorithm 3, the full procedure is shown. We discuss the algorithm below, in which the line numbers refer to specific parts of Algorithm 3.

After an initialization step for the first depot visit, we loop through all other locations j in the candidate route. If j is a task, we check whether we arrive before its deadline and whether we depart in time compared to its latest return time for the planning horizon. If not, the route is infeasible (line 12-15).

If j is a depot, we check whether the trip to j is feasible with respect to endurance (line 24-30). Initial violations might be repaired by extending the visit to the previous depot d, since the trip duration becomes smaller when this extension leads to lower waiting times in the trip. For each task $i \in Trip_j$, a violation of the latest return time LRT_i^{END} can be repaired by extending the visit to d if and only if the extension removes a sufficient amount of waiting time before i and preserves feasibility with respect to time windows.

If an extension is necessary, we extend the visit to d by the *minimal* feasible extension size (line 31). This ensures that we can immediately terminate the algorithm in case of a time window violation, since the preceding depot visit(s) cannot be *reduced* and hence, the arrival time cannot be decreased. Algorithm 4 shows the corresponding update of time variables, taking into account that the extension delay decreases over the trip due to reduced waiting times.

Maximum shifts

Before explaining how to determine the maximum shifts, we introduce some terminology. The *time window buffer* of a task j with planned arrival time arr_j is given by $U_j - arr_j$. Similarly, the *endurance buffer* and *planning horizon buffer* are given by $LRT_j^{END} - arr_j$ and $LRT_j^{HOR} - arr_j$, respectively.

As we mentioned, the maximum shifts are determined during the construction of the schedule. When the loop in Algorithm 3 arrives at a depot j, we first compute $ms_{j'}^{TW}$ by backward recursion (line 17-22). We loop over *all* locations j' before depot j, since the maximum shift for a location might be affected by a small time window or planning horizon buffer for some later location, even if the later task is part of a different trip. This step is performed before the endurance check, since ms_d^{TW} serves as an upper bound on the extension size of the visit to depot d. For the update formula, notice that ms^{TW} incorporates buffers with respect to the planning horizon (line 19) and that waiting times are included (line 20), since delays might (partially) vanish due to waiting times.

Algorithm 3 Constructing a feasible schedule

Input: Planning horizon $[0, T]$, maximal travel time deviation fraction σ_{TT}.
Input: Resource r with endurance E_r.
Input: Candidate route with ordered location set L, with subsets $Trip_j \subset L$ that contains all tasks in a trip to depot j.
Input: Locations $i \in L$ (tasks and depots) with time windows $[L_i, U_i]$.
Input: For locations $i, j \in L$ and resource r: expected travel times $\bar{t}_{ijr}$, expected service times $\bar{s}_{ir}$.
Input: Previous depot d, previous location i, and $tripWait_i$ is the total waiting time in the trip up to and including task i.
Output: A feasible schedule for the route or an infeasibility message.

Initialize: $d = i = firstDepot$, $wait_i = dep_i = tripWait_i = 0$.

for all $j \in L \backslash \{firstDepot\}$ **do**
  Set $arr_j = dep_i + \bar{t}_{ijr}$ and $LRT_j^{HOR} = T - (1 + \sigma_{TT})\bar{t}_{id'r}$.
  if j is a task **then**
    if $arr_j > U_j$ **or** $arr_j > LRT_j^{HOR} - \bar{s}_{jr}$ **then**
      Stop. Route is infeasible w.r.t. time windows or planning horizon.
    end if
    Set $wait_j = (L_j - arr_j)^+$ and $tripWait_j = tripWait_i + wait_j$.
  else
    Set $ms_j^{TW} = T - start_j$ and $nextLoc = j$.
    for all locations $j' \in L$ before j **do**
      Set $taskSpace_{j'} = \min\{U_{j'}, LRT_{j'}^{HOR} - \bar{s}_{jr}\} - start_{j'}$.
      Set $ms_{j'}^{TW} = \min\{taskSpace_{j'}, wait_{nextLoc} + ms_{nextLoc}^{TW}\}$.
      Set $nextLoc = j'$.
    end for
    Compute $LRT_t^{END} = dep_d + E_r - (1+\sigma_{TT})\bar{t}_{ijr}$ for each task $t \in Trip_j$.
    for all tasks $i \in Trip_j$ **do**
      Set violation $viol_i = dep_i - LRT_i^{END}$.
      if $tripWait_i < viol_i$ **or** $ms_d^{TW} < viol_i$ **then**
        Stop. Route k is infeasible w.r.t. endurance.
      end if
    end for
    Extend visit to depot d by size $(\min\{viol_i \mid i \in Trip_j\})^+$.
    Set $ms_i^{END} = LRT_i^{END} - dep_i$ and $nextTask = i$.
    for all tasks $i' \in Trip_j$ before i **do**
      Set $ms_{i'}^{END} = \min\{LRT_{i'}^{END} - dep_{i'}, wait_{nextTask} + ms_{nextTask}^{END}\}$.
      Set $nextTask = i'$.
    end for
    Set $d = j$ and $tripWait_j = 0$.
  end if
  Set $i = j$ and $dep_i = arr_i + wait_i + \bar{s}_{ir}$.
end for

Algorithm 4 Extending a depot visit

1: **Input:** Trip from depot d to depot j with task set $Trip_j$.
2: **Input:** Current partial schedule up to the end of the trip.
3: **Input:** Extension size $extendBy$ of the visit to depot d.
4: **Output:** Updated time variables $arr, wait, ms^{TW}$.

5: Set $wait_d = wait_d + extendBy$ and $ms^{TW}_d = ms^{TW}_d - extendBy$.
6: Set $\Delta = extendBy$.
7: **for all** tasks $j' \in Trip_j$ **do**
8: Set $oldWait = wait_{j'}$.
9: Set $wait_{j'} = (oldWait - \Delta)^+$ and $arr_{j'} = arr_{j'} + \Delta$.
10: Set $\Delta = (\Delta - oldWait)^+$ and $ms^{TW}_{j'} = ms^{TW}_{j'} - \Delta$.
11: **end for**

If the endurance check is successfully passed, we use a similar backward recursion to compute the maximum shifts for endurance (line 31-35). Contrary to ms^{TW}, we only consider the tasks in the trip to j, since the latest return times are measured with respect to the departure time from the previous depot. Moreover, while delays can have a negative effect on time window buffers in a next trip, this is impossible for endurance buffers, since the entire trip is shifted.

5.3 Fast feasibility check for task insertions

A fast check whether a task insertion is feasible is indispensable to have a quick Tabu Search algorithm, because we search for the best solution in the entire neighborhood in each iteration and an attempt to insert the task can be made for any position in the current route. An even more important reason specific to our multi-trip problem is that when trying to add a task to a route, there are many possibilities regarding the positions of recharging depots, even if the order of the present tasks remains fixed.

In Section 5.3.1, we discuss a fast feasibility check for task insertion at a specific position in an otherwise fixed route. A framework to handle the positions of the recharging depots is discussed in Section 5.3.2.

5.3.1 Check for a fixed location sequence

The feasibility of an insertion might be checked by constructing a new schedule from scratch, using Algorithm 3 from Section 5.2.2. However, this is rather time-intensive, especially since we check a huge amount of insertions in each iteration, of which a large fraction is infeasible. Therefore, we would benefit from a procedure to judge the feasibility of an insertion at an early stage.

For this purpose, we propose a fast feasibility check that builds on the maximum shift variables. It can be considered as the multi-trip and multi-constraint version of the check proposed by Vansteenwegen et al. [2009]. It covers the constraints on time windows, endurance, and the planning horizon. In this subsection, we discuss attempts to insert a task a specific position in a fixed route. In particular, we do not reorder tasks or change the amount or positions of recharging depots in this stage.

Consider inserting a task j between visits i and k in a route for resource r. This insertion is feasible if and only if the inserted task j satisfies the following conditions:

- The inserted task j satisfies its own time window and latest return times.
- The delay for the visits after j does not lead to an infeasibility with respect to time windows and endurance.

For the first condition, we compare the expected arrival time $\overline{arr}_j = dep_i + \bar{t}_{ijr}$ to the time window deadline U_j. Furthermore, we compare the expected departure time $\overline{dep}_j = \max\{\overline{arr}_j, L_j\} + \bar{s}_{jr}$ to the latest return times LRT_j^{END} and LRT_j^{HOR}, computed by equations (5.5) and (5.6). The first condition is satisfied if and only if the following two conditions hold:

$$\begin{aligned} \overline{arr}_j &\leq U_j && (5.7)\\ \overline{dep}_j &\leq \min\{LRT_j^{END}, LRT_j^{HOR}\} && (5.8) \end{aligned}$$

For the second condition, use $wait_j = (L_j - arr_j)^+$ to define the *delay* caused by inserting a task j between i and k by

$$Delay_{ijkr} = (\bar{t}_{ijr} + \bar{t}_{jkr} - \bar{t}_{ikr}) + wait_j + \bar{s}_{jr}.$$

Inserting task j between i and k delays the arrival time at visit k by exactly $Delay_{ijkr}$. The key of the fast feasibility check is that by the recursive construction of the maximum shift variables, it suffices to compare the delay only to the maximum shift of the next task k. In particular, the second condition is satisfied if and only if the following two conditions hold:

$$\begin{aligned} Delay_{ijkr} &\leq wait_k + ms_k^{TW} && (5.9)\\ Delay_{ijkr} &\leq wait_k + ms_k^{END} && (5.10) \end{aligned}$$

Thus, inserting j between i and k is feasible if and only if conditions (5.7) - (5.10) hold. If a time window check fails, the insertion is infeasible. If only an endurance check fails, the insertion is feasible if and only if the depot visit before j can be extended sufficiently. After successfully passing the feasibility check (and before any robustness check), we are sure that a feasible schedule exists for the extended route, which we construct using Algorithm 3. If the insertion is infeasible, we try another insertion position of j. If we find a feasible insertion, we do not try other insertion positions for this task anymore.

5.3.2 Handling the positions of recharging depots

The above feasibility check suffices for single-trip problems, when performed for any possible insertion position. However, in our multi-trip problem, failing this feasibility check does not necessarily mean that the task sequence is infeasible. In particular, the insertion might actually be feasible for a different structure for recharging depots in the route, regarding their amount and their positions. Trying all possible amounts and positions for every initially infeasible task sequence would significantly slow down the search process, since this can boil down to checking a huge number of routes with the same task sequence.

As an alternative, we introduce a set $\mathcal{A}_r$ of **alternative routes** for resource r. This set consists of feasible routes with the same task sequence as the current route in the search process, but with different amounts and positions of recharging depots. In order to keep the size of the set limited, we only allow 'reasonable' amounts of recharging depots in the sense that they differ at most one from the amount of recharging depots in the current route.

After each search iteration, we update the set of alternative routes for each changed route. For each reasonable amount d of recharging depots and any possible insertion position(s) of these depot visits, we try to construct a feasible schedule by Algorithm 3. The set of alternative routes consists of all such routes for which we find a feasible schedule. Additionally, for each task i in a changed route for resource r, we apply the same procedure as above to find a set $\mathcal{A}_{ir}$ of *alternative routes after removal* of this task i.

In the next search iteration, the generated routes are used for checking the feasibility of task insertions. In particular, adding a task to a route for resource r by an Addition and Change move is feasible if and only if the task can be inserted in any route in $\mathcal{A}_r$. Adding a task j to a route for resource r after removal of a task i, by a Swap or Exchange move, is feasible if and only if j can be inserted in any route in $\mathcal{A}_{ir}$.

One of the main advantages of the alternative routes framework is that we discard the positions of recharging depots that are already infeasible before inserting any task. Hence, we significantly reduce the amount of options to try for a task insertion. Moreover, the constructed schedules for alternative routes allow for a fast feasibility check of a task insertion, as described in Section 5.3.1. Thus, we only need to construct a schedule from scratch within the search iteration as soon as we are certain that the task insertion is feasible.

5.4 Intensification and diversification

In our Tabu Search algorithm, we apply intensification and diversification mechanisms that try to guide the search process to a global optimum. The aim of diversification methods is to explore regions of the solution space that have not

been addressed yet, or have a low probability of being addressed in the future. The aim of intensification methods is to focus on a promising part of the solution space, in order to improve a relatively good solution even further.

In Section 5.4.1, we discuss an intensification procedure of reordering tasks within routes. In Section 5.4.2, we discuss a diversification procedure of random perturbation. This procedure includes adding a random number of currently unscheduled recurrent tasks to the solution. Both procedures are inspired by Cordeau and Laporte [2005].

The role of these mechanisms in the Tabu Search process was already shown in line 24-31 of Algorithm 2. We interrupt the search process and apply a reorder or perturbation procedure after some number of iterations without improvement of the previous solution, the best solution in a subsearch, or the best solution in the overall search process. In that case, we reorder the best solution in the current subsearch if that has not been done before. Otherwise, we do not expect to find a better solution quickly in this region of the solution space and we apply a perturbation procedure to the overall best solution. In both cases, we start a new subsearch within the Tabu Search process.

5.4.1 Intensification: reorder tasks within routes

When we encounter a feasible task insertion in the search process, we immediately consider the insertion to be successful. Thus, we ignore the possibility of inserting the task at different insertion positions or in different alternative routes. Moreover, we choose the order in which we try insertion positions randomly. Due to 'wrong' decisions in these respects, it is likely that we obtain routes that have a sub-optimal order of tasks. In particular, different task orders might be better in the sense of larger time window buffers, lower total travel time, or a larger space to insert additional tasks. This motivates the application of a reordering procedure of tasks within routes.

Suppose we would like to apply a reordering procedure on a route k for resource r. Especially for fast resources, routes typically contain many tasks and enumeration of all possible task sequences is extremely time-intensive. Therefore, we apply a Local Search procedure in which we use route k as a starting point. We use a neighborhood that consists of the single move *Relocate*, in which we relocate a task to a different position. This move can be applied in both directions, as can be seen in Figure 5.3.

Some candidate moves have more potential to improve the task order than others. We prefer task orders for which the time window deadlines are in increasing order and the total travel time is low. Therefore, we first try the most promising moves in each search iteration. We start with moves that improve both the deadline order and the total travel time. If no feasible move can be found, we proceed with other moves that improve the deadline order. If this does not yield a feasible move either, we continue with all other possible orders.

For each move, we check its feasibility by constructing a schedule from scratch by Algorithm 3, for any reasonable amounts and positions of recharging depots. If we find a feasible schedule, we store the obtained set of alternative routes and terminate the reorder iteration immediately by moving to the accepted task sequence. We terminate the reorder process if we cannot find a feasible new task sequence or if the maximum number of reorder iterations is reached. Since the reorder procedure involves constructing a large number of schedules from scratch, the maximum number of iterations should remain small.

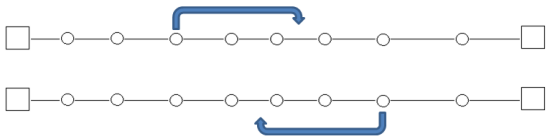

Figure 5.3: Two examples of a Relocate move in the reorder procedure.

5.4.2 Diversification: random perturbation

If we do not expect the solution in the current subsearch to improve quickly, and the best solution in the current subsearch has already been reordered, we randomly perturb the overall best solution found so far. Thus, we hope to find improvements of the overall best solution in regions of the solution space that are yet to be explored.

In the perturbation procedure, we pay special attention to recurrent tasks. Due to the non-linear profit function in recurrent sets, the objective increase by including a single task from a recurrent set is relatively low. However, as soon as a larger part of the recurrent set is included in the solution, it might actually turn out to be worthwhile to include these recurrent tasks in the solution. Therefore, we include a step to add a random number of recurrent tasks in the perturbation procedure, inspired by the local moves in Kuipers [2003].

The full perturbation procedure can be described by the following three steps:

1. Take the current best solution and an empty tabu list as a starting point.
2. Remove a random fraction of tasks from the best solution, and declare assigning these tasks to their old resource tabu. Find a set of alternative routes for each reduced route.
3. Choose a random number of available recurrent sets, and for each set, try to insert a random number of recurrent tasks that were not removed in Step 2. Declare deleting these tasks from the schedule tabu.

For Step 2, we set an upper bound on the random fraction of tasks to remove. We let this upper bound decrease over time, in order to switch the focus from diversification to intensification.

5.5 Speed-ups

In addition to the fast feasibility check and the alternative routes framework from Section 5.3, we incorporated several other speed-ups in our Tabu Search algorithms, including storage of lists of infeasible additions and computation of potential objective increase. Both of these speed-ups aim to reduce the number of task insertion attempts. Moreover, we must ensure that the alternative routes framework does not take too much time and memory.

For each resource r, we store a list of tasks that cannot be added to the route of r currently. This list of *infeasible additions* is used to avoid repeated attempts to add a task to a route by an Addition move or a Change move, which we already found to be infeasible. Moreover, for each task i that is currently assigned to some resource r, we store a list of tasks that cannot be added to the route of r after removal of i. Similarly, this list is used to avoid repeated attempts to insert a task in a route by a Swap move or an Exchange move. Whenever a task is removed from a route, a previously infeasible insertion might become feasible. Therefore, we clean all lists for infeasible additions concerning this route in this case.

Furthermore, for each of the four local move types, we sort the candidate moves in decreasing order of their *potential objective increase*. This value can be computed based on the current solution and the tasks related to the candidate move. Consequently, if we find a feasible candidate move of some type, we can ignore all other candidate moves of the same type, since they cannot lead to a higher objective value.

For routes that contain many tasks and many recharging depots, the number of possible alternative routes can be huge. Thus, the computation time for finding all possible alternative routes can become unacceptably high and storing all of them leads to memory issues. Therefore, we apply a greedy method to find alternative routes quickly after each search iteration and in the reordering procedure, while keeping the maximum number of alternative routes limited. In this stage, we do not change the current number of recharging depots.

As an additional intensification procedure, we apply a procedure in which we take some more time to find alternative routes and allow to store a larger number of them. If the number of tasks and depot visits in the routes is small enough, we find all possible alternative routes in which the number of recharging depot visits differs at most one from the amount in the current route. Otherwise, we apply a greedy procedure to find alternative routes with the least possible number of recharging depots, and we store all feasible routes obtained by adding a recharging depot visit to the current route. This procedure is applied to the best solution in the current subsearch after a certain number of iterations without improvement, provided that it has not been applied to this solution before. Otherwise, we proceed to the reordering procedure.

Chapter 6

Robustness concepts

Since the travel and service times are uncertain, the optimal solution for the deterministic problem might perform bad in practice. For instance, a resource might need to skip an important task if it arrives too late or if a timely arrival at the depot is in danger. Therefore, it is important to take the uncertainties into account in the offline stage, in order to construct solutions that are *robust* against deviations from expected travel and service times in real-time.

However, by focusing on robustness, we might derive a solution which is too conservative, with a small total task value. Thus, a trade-off exists in the offline stage between the total task value and the robustness of the solution. In this chapter, we discuss two robustness concepts to handle this trade-off. Both concepts try to bridge the gap between mean-case scheduling and worst-case scheduling, of which the former easily leads to cancellation of high priority tasks in real-time and the latter is very conservative.

In Section 6.1, we provide a working example for illustration of the concepts in the subsequent sections. In Section 6.2, we introduce the new concept of the *risk function*, which only allows to perform a low priority task before an important task if this does not endanger the execution of the important task too much. In Section 6.3, we discuss using a *worst-case fraction* that only allows solutions that can still obtain a certain fraction of their original value in the worst-case scenario. The risk function and the worst-case fraction impose an additional requirement on a feasible schedule and a feasible solution, respectively.

By incorporating the robustness concepts in the Tabu Search algorithm, as we explain in Chapter 7, we aim for a good online performance. For investigating the effect of the robustness concepts in practice, we generate offline solutions with and without the robustness concepts and compare their performance in real-time. For this purpose, we apply rescheduling policies which we discuss in Chapter 8, based on scenarios implied by multiple simulations of travel and service times. We discuss the results of the simulations in Chapter 10.

6.1 Working example

In this section, we provide a small working example of an offline solution. Throughout this chapter, we repeatedly return to this example to illustrate the concepts of the risk function and the worst-case fraction.

In Figure 6.1, we show the routes in an offline solution with two resources. For simplicity, we assume that the two resources are homogeneous with an endurance of 100, a minimal rest time of 10. The route for resource 1 consists of two trips and contains six tasks, while the route for resource 2 consists of a single trip and contains four tasks.

In Tables 6.1 and 6.2, the schedules corresponding to the routes are shown. We assume that all expected travel and service times are equal to 10. The maximal deviation fractions for travel and service times are given by $\sigma_{TT} = 0.1$ and $\sigma_{ST} = 0.25$, respectively. That is, travel and service times can be delayed by at most 10% and 25%, respectively.

In addition to the schedule, we state the time windows and latest return times LRT^{END} and LRT^{HOR} in the tables. The latest return times can be computed as explained in 5.2.1, using an endurance of 100 and a planning horizon of [0, 300]. Furthermore, we use that all worst-case travel times to the depot are equal to 11, as all expected travel times are equal to 10 and $\sigma_{TT} = 0.1$.

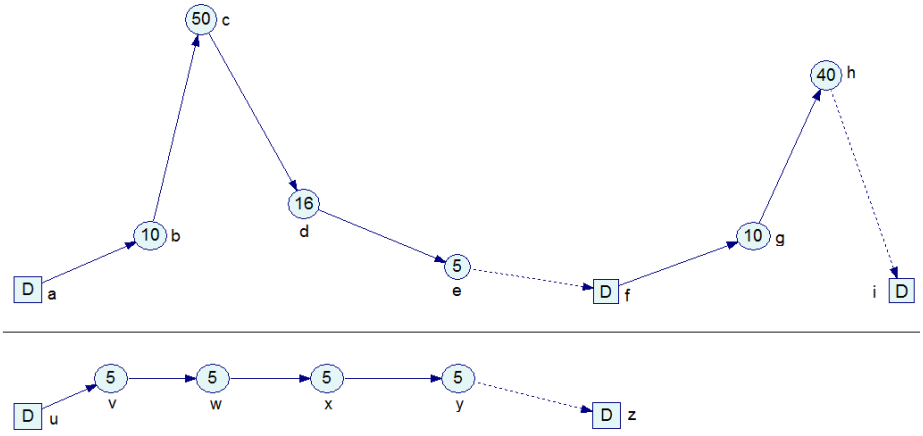

Figure 6.1: Working example: route $a - i$ for resource 1, route $u - z$ for resource 2. The squares represent depots, the circles represent tasks with their values.

Location	Arrival	Departure	Time window	LRT^{END}	LRT^{HOR}
a	-	0	[0, 300]	-	-
b	10	20	[0, 100]	89	289
c	30	40	[0, 100]	89	289
d	50	60	[0, 55]	89	289
e	70	80	[0, 70]	89	289
f	90	100	[0, 300]	-	-
g	110	120	[0, 200]	189	289
h	130	140	[0, 134]	189	289
i	150	-	[0, 300]	-	-

Table 6.1: Schedule for the route for resource 1 in Figure 6.1. LRT^{END} and LRT^{HOR} represent latest return times with respect to endurance and planning horizon, respectively.

Location	Arrival	Departure	Time window	LRT^{END}	LRT^{HOR}
u	-	0	[0, 300]	-	-
v	10	20	[0, 300]	89	289
w	30	40	[0, 300]	89	289
x	50	60	[0, 300]	89	289
y	70	80	[0, 300]	89	289
z	90	-	[0, 300]	-	-

Table 6.2: Schedule for the route for resource 2 in Figure 6.1. LRT^{END} and LRT^{HOR} represent latest return times with respect to endurance and planning horizon, respectively.

6.2 Risk function

In this section, we introduce a concept that determines for each task whether it is 'safe' to perform the task or not, based on the risk of being forced to cancel later tasks with higher value. Intuitively, a task should not be performed if this makes the risk of skipping a more important task in the 'future' too high. The risk function formalizes this concept and can be applied for any task values.

The main idea of this concept is as follows. If a route contains a task j that is succeeded by a series S of more important tasks, we perform a 'stress test' by incorporating artificial delays of travel and service times. If the ratios between the values of j and the tasks in S are higher, we incorporate more severe delays. We check the feasibility of the delayed schedule with respect to time windows, endurance, and the planning horizon. In particular, we check for each task $j' \in S$ whether we still arrive and depart in time. If the delayed schedule is

infeasible, we declare the route to be risk-infeasible. Thus, risk-feasibility can be regarded as an additional constraint to a feasible schedule, in the sense that we only allow risk-feasible routes to appear in a solution.

For instance, consider the first route in Figure 6.1. This route has a feasible schedule for the mean-case scenario, specified in Table 6.1. However, due to stochasticity of travel and service times, the execution of tasks in real-time is uncertain. In particular, the presence of task b might endanger the execution of the more important task c. That is, delays related to task b might force us to skip the more important task c in real-time.

In this example, we incorporate severe artificial delays in the part of the route until task c, since task c is significantly more important than task b. The corresponding risk check consists of determining whether we still expect to arrive at task c before its deadline, and to depart from task c before its latest return times, given the real-time scenario implied by the artificial delays. A similar risk check can be applied in a later part of the same route, in order to determine whether it is responsible to the relatively unimportant perform task e, regarding the risk of having to skip a more important task g or h. If any risk check for a route fails, the route is risk-infeasible.

Outline

In Section 6.2.1, we formalize the 'future' of a task in the idea that a task should not be performed if this makes the risk of skipping a more important task in the 'future' too high. In general, this future does not refer to the entire remaining part of the route. For instance, it does not make sense to declare the above route risk-infeasible if artificial delays related to task b would lead to a late arrival at task e, since task e is less important than task b. We restrict the future of a task to *blocks*, which contain subsequent tasks with higher value.

In Section 6.2.2, we discuss how to determine the magnitude of the artificial delays incorporated in a risk check. As we mentioned, relatively large differences in task values (i.e. higher value ratios) lead to more severe delays. In particular, we introduce a *delay function* that translates value ratios into artificial delay fractions. If a value ratio is assigned to a location, the corresponding delay fractions can be easily translated into a delayed travel time to this location and a delayed service time of this location. The assignment of value ratios to locations involves a further division of blocks into *sub-blocks*. To each sub-block, we assign a *risk function* that maps locations to value ratios.

For each sub-block, a risk check is performed. In particular, the corresponding risk function and the given delay function are used to determine a delayed schedule. In Section 6.2.3, we discuss in detail how to determine whether a sub-block remains feasible given this delayed schedule, with respect to the constraints regarding time windows, endurance, and planning horizon. We finalize this section with some remarks in Section 6.2.4.

6.2.1 Construction of blocks

Recall that our aim is to check whether it is 'safe' enough to perform a task, taking the risk of skipping more important tasks in the 'future' into account. In this section, we formalize the 'future' of a task.

Let a **block** refer to a part of the route that starts at some task j and contains a subsequent series of locations, of which the tasks all have higher value than v_j. The procedure to find all blocks of a route is as follows. We loop through all tasks and start a block at task j if the next task has a higher value than v_j. We iteratively add the next location to the block until we encounter a task j' with $v_{j'} \leq v_j$ or the final depot.

We illustrate the procedure for the routes from Figure 6.1. For the second route, observe that all values are equal. Therefore, this route does not contain any block, which means that no risk checks will be performed on this route. Hence, we can immediately conclude that this route is risk-feasible.

However, the first route contains several blocks. The first block starts at task b, because the next task c has a higher value. This block ends at task d, since the next task value v_e does not exceed v_b. After completing this block, we continue the loop at task c. However, no block starts at task c nor at task d, because of decreasing task values. The next block starts at task e, since the first task afterwards (task g) has a higher value, and ends at task h, right before the final depot. Observe that this block contains a recharging depot visit. Continuing the loop at the next task g yields another block $g - h$, which is nested in the larger block $e - h$. Thus, this route contains the blocks $b - d$, $e - h$, and $g - h$.

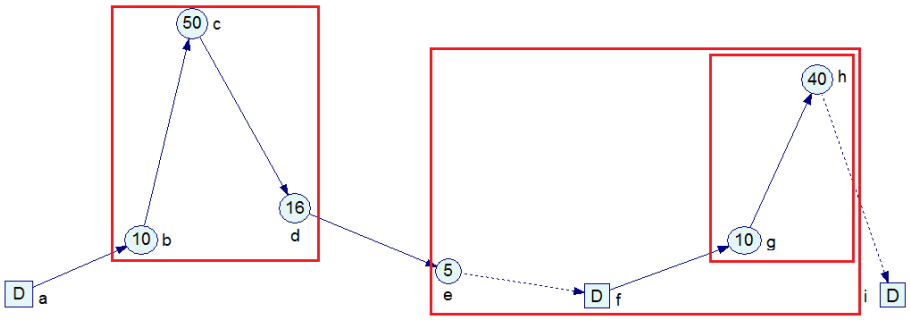

Figure 6.2: The first route from the working example contains blocks $b - d, e - h, g - h$.

For the decision whether it is 'safe' enough to perform a task, we check whether a block starts at this task. If that is the case, we check whether this block remains feasible when incorporating artificial delays. In Section 6.2.2, we discuss why and how to divide blocks further into sub-blocks. A block is feasible if and only if all of its sub-blocks are feasible. A route is risk-feasible if and only if all of its

blocks are feasible.

We stress that we ignore any problems that the delays in a block might cause *after* this block. For instance, we do not declare the route in Figure 6.2 to be risk-infeasible if delays in the block $b-d$ would definitely lead to a late arrival at task h, even though task h is more important than task b. Thus, we implicitly assume that such problems after a block can be solved by deleting the first task after the block (task e in the example), which is by construction at most equally important as the starting task of the block.

6.2.2 Delayed travel and service times

Recall that the main idea of the risk function is to perform stress tests by incorporating artificial delays. Below, we discuss in detail how to determine the severeness of these delays, based on the value ratios of the concerned tasks. In general, larger value ratios lead to more severe delays.

First, we suppose that an assignment of value ratios to locations is given and show how we translate them to delayed travel and service times. Afterwards, we discuss how to assign value ratios to locations. The most natural assignment of value ratios to locations would be to assign a value ratio v_i/v_j to each task i in a block that starts at task j. However, it turns out that this choice would be overrestrictive for certain blocks. Therefore, we propose an alternative assignment that involves sub-blocks.

Translating value ratios to delay fractions

We introduce a **delay function** $g : [1, \infty) \to (0, 1]$ that translates value ratios into corresponding *delay fractions*. A delay function should be increasing, in order to reflect that higher value ratios lead to more severe delays. For an example of a piecewise linear delay function, see Figure 6.3.

This delay function has a cutoff ratio $\bar{R} = 4$, in the sense that any value ratio that exceeds $\bar{R}$ leads to the maximal delay fraction of 1. In general, for a delay function $g(R) = \min\{(R-1)/(\bar{R}-1), 1\}$, a lower cutoff ratio $\bar{R}$ leads to a steeper delay function and the resulting delay fractions are higher. For the remainder of this chapter, we assume that the delay function from Figure 6.3 is given.

We use these delay fractions to compute delayed travel and service times. Suppose that a value ratio R is assigned to a task $l \in L$, preceded by a location $k \in L$ in a route for resource r. Then we compute an adjusted travel time t'_{klr} from k to l and an adjusted service time s'_{lr} of l as follows:

$$t'_{klr} = (1 + \sigma_{TT} \times g(R))\bar{t}_{klr} \tag{6.1}$$

$$s'_{lr} = (1 + \sigma_{ST} \times g(R))\bar{s}_{lr} \tag{6.2}$$

For a depot $l \in L$, equation (6.1) still holds, but the 'service time' of this depot remains equal to the minimal rest time of resource r. Recall that $\bar{t}_{klr}$ and $\bar{s}_{lr}$ represent the expected values for the travel and service time, respectively, while σ_{TT} and σ_{ST} denote the corresponding maximal deviations. The **delay percentages** of the original travel and service times are given by $100\sigma_{TT} \times g(R)$ and $100\sigma_{ST} \times g(R)$, respectively.

For an example for calculating delay percentages, recall from Section 6.1 that we assume $\sigma_{TT} = 0.1$ and $\sigma_{ST} = 0.25$. Suppose that a value ratio of $R = 1.6$ is assigned to some task l. For the delay function $g(R)$ from Figure 6.3, this leads to a delay fraction of $g(R) = 0.2$. Hence, the travel time to l and the service time of b are delayed by 2% and 5%, respectively.

Notice that we assumed that some value ratio R is associated to task l, without giving further details yet. The remaining part of Section 6.2.2 is devoted to answering the question which value ratios to assign to which locations.

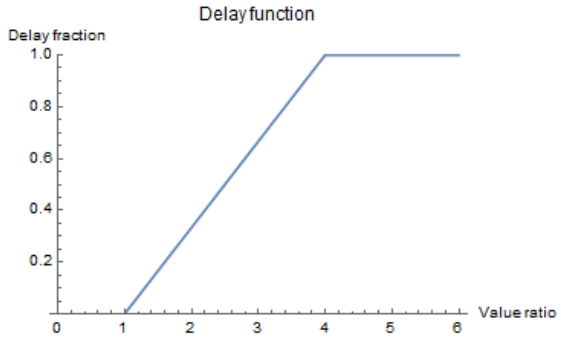

Figure 6.3: A delay function g with cutoff ratio $\bar{R} = 4$, given by $g(R) = \min\{(R-1)/3, 1\}$ for $R \geq 1$.

Assigning value ratios to locations

We introduce a **risk function** $f : L \to [1, \infty)$ that assigns a value ratio to each location from a set L. The obtained value ratios serve as input for the delay function. Below, we argue by an example that we cannot consider an entire block at once in a feasibility check for a stress test. Afterwards, we explain how to find suitable **sub-blocks** and corresponding risk functions.

Consider the first block $b - d$ from Figure 6.2. For this block, the most natural option is to compare the values of c and d to the value of the starting task b. The resulting value ratios indicate the relative importance of tasks c and d compared to task b, which reflects the aim to check whether it is responsible to perform b. In particular, this block would have a corresponding risk function defined by $f(b) = f(c) = v_c/v_b = 5$ and $f(d) = v_d/v_b = 1.6$. Observe that we

include task b in the risk function, in order to reflect that delays related to the starting task b of this block are also highly relevant in the risk check whether it is responsible to perform task b or not. Thus, a value ratio of 5 is associated to tasks b and c, and a value ratio of 1.6 is associated to task d.

However, this choice turns out to be overrestrictive for this block. Given the delay function g from Figure 6.3 and the value ratio $R = 5$ associated with tasks b and c, we can compute delay percentages, as shown in Figure 6.4. Since the value ratio $R = 5$ exceeds the cutoff ratio $\bar{R} = 4$ for this delay function g, we have maximum delay percentages. This leads to worst-case adjusted travel and service times $t'_{abr} = t'_{bcr} = 11$ and $s'_{br} = s'_{cr} = 12.5$. Similarly, we can compute $t'_{cdr} = 10.2$ and $s'_{dr} = 10.5$ using the value ratio $R = 1.6$ associated with task d. Given that we depart from the preceding depot a at time 0, the above delays lead to $arr'_b = 11, dep'_b = 23.5, arr'_c = 34.5, dep'_c = 47, arr'_d = 58, dep'_d = 70.5$. Since the arrival time at d exceeds $U_d = 53$, we would declare this route to be risk-infeasible.

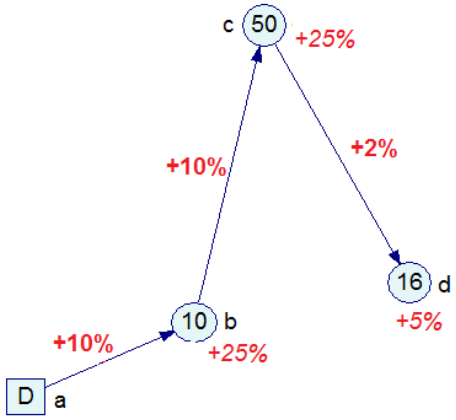

Figure 6.4: The delay percentages that correspond to a naive assignment of value ratios to locations in block $b - d$. The corresponding delay percentages for travel and service times are denoted in boldface and italic, respectively.

Observe that arr_d is partially based on the worst-case delays related to task c. However, since the value difference between b and d is relatively small, it is not fair to judge the feasibility for task d based on worst-case delays for the route up to task c. In other words, the worst-case delays are appropriate for the feasibility checks regarding the route up to task c, but overrestrictive for task d. It would be better if we only check task c under the worst-case delays up to task c, and use relaxed delays for the feasibility checks regarding task d.

For this purpose, we perform risk checks on *sub-blocks* rather than on an entire block. In this example, we divide the block $b-d$ into sub-blocks $b-c$ and $b-d$. For the sub-block $b-c$, we use the risk function $f(b) = f(c) = 5$. This leads to the delay percentages as shown in the left part of Figure 6.5 and an adjusted

partial schedule with $arr'_b = 11, dep'_b = 23.5, arr'_c = 34.5, dep'_c = 47$, which does not lead to feasibility problems.

For the sub-block $b - d$, we overrule the previously assigned value ratios to b and c, and relax them to $f(b) = f(c) = f(d) = 1.6$. This leads to the delay percentages as shown in the right part of Figure 6.5, and an adjusted partial schedule $arr'_b = 10.2, dep'_b = 20.7, arr'_c = 30.9, dep'_c = 41.4, arr'_d = 52.4, dep'_d = 62.9$. Observe that for this more appropriate risk check, task d does not violate any constraint either. Since both sub-blocks are risk-feasible, the block $b - d$ is risk-feasible.

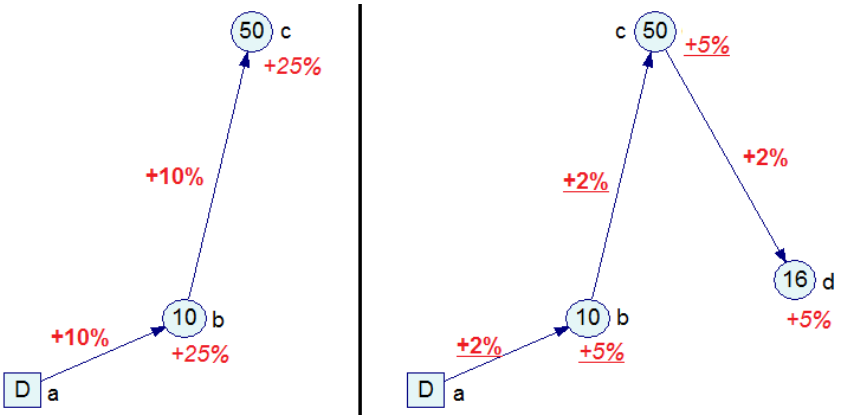

Figure 6.5: The block $b - d$ contains sub-blocks $b - c$ and $b - d$. The overruled delay percentages are underlined.

We generalize this framework in Algorithm 5. For a given block, this procedure shows how to find all of its sub-blocks and their corresponding risk functions. We stress that sub-blocks are not the same as nested blocks. For a given block, each value decrease leads to a sub-block, while each value increase leads to a nested block. We explain the algorithm below, referring to specific parts of the algorithm by line numbers.

We loop through all tasks in the block, starting at the first location after the starting task j. For each task l we encounter, we compute a value ratio $R = v_l/v_j$ (line 10). If v_l is at least as high as the value of the previous task, then we set $f(l) = R$ (line 11-12). Note that no value ratios are computed for depots and for the starting task j. To these locations, we assign the value ratio that corresponds to the next task (line 13-18 and line 22-24).

If v_l is lower than the value of the previous task, then we create a sub-block by cutting off the blcok directly after the previous task, and we assign the current function f as its risk function (line 20). Afterwards, we set $f(l) = R$ and for all locations l' before l for which we currently have $f(l') < R$, we overrule the current value ratio by setting $f(l') = R$ (line 25-27).

Algorithm 5 Procedure to find sub-blocks and their risk functions

Input: Route with locations L with a feasible mean-case schedule.
Input: Block starting at task j, with locations $L' \subset L$.
Input: Delay function $g : (1, \infty) \to (0, 1]$ that maps value ratios to delay fractions.
Output: A list ($list$) of risk functions $f : L'' \to [1, \infty)$, each of which assigns a value ratio to locations L'' in a sub-block.

Initialize: $prevTask = j, prevLoc = j, prevValue = v_j, list = \emptyset$.
for all $l \in L' \backslash \{j\}$ **do**
  if l is a depot **then**
    Set $prevLoc = l$.
  else
    Compute value ratio $R = v_l / v_j$.
    if $v_l \geq prevValue$ **then**
      Set $f(l) = R$.
      if $prevTask = j$ **then**
        Set $f(j) = R$.
      end if
      if $prevLoc$ is a depot **then**
        Set $f(prevLoc) = R$.
      end if
    else
      Add current f to $list$.
      Set $f(l) = R$.
      if $prevLoc$ is a depot **then**
        Set $f(prevLoc) = R$.
      end if
      for all $l' \in L'$ before l **do**
        Set $f(l') = \min\{f(l'), R\}$.
      end for
    end if
    Set $prevTask = l, prevLoc = l$ and $prevValue = v_l$.
    if l is final task of risk-subroute **then**
      Add current f to $list$.
    end if
  end if
end for

This update operation is crucial to avoid overrestrictive cases as in Figure 6.4. It ensures that we lower previous delays that were too severe, regarding the value ratio for the current task l. Meanwhile, creating a sub-block *before* this update operation ensures that the times windows and latest return times of the tasks contained in this sub-block are still checked based the more severe delays.

We terminate the loop if we have arrived at the final task of the block, after creating a final sub-block that equals the whole block (line 30-32).

When we apply this algorithm to the second block $e - h$ from Figure 6.2, we only obtain the entire block as a sub-block, as we do not encounter decreasing values in this block. The risk function corresponding to $e - h$ is given by $f(e) = f(f) = f(g) = 2$ and $f(h) = 8$. For the nested block $g - h$, the risk function is given by $f(g) = f(h) = 4$.

6.2.3 Feasibility check of sub-blocks

Recall that a block is risk-feasible if and only if all of its sub-blocks remain feasible when incorporating the artificial delays corresponding to their risk functions. In Section 6.2.2, we already performed some feasibility checks in examples. Below, we discuss the feasibility check of a sub-block in more detail.

Consider a sub-block that starts at some task j and has a corresponding risk function $f : L'' \to (1, \infty)$. First of all, for each location $l \in L''$, we apply equations (6.1) and (6.2) to compute delayed travel and service times, using a given delay function. Moreover, we set the *base time* equal to the planned departure time from the location before j.

After these procedures, we can proceed to the feasibility check of the sub-block. First, we perform a feasibility check with respect to time windows and the planning horizon for all sub-blocks of a route. During these feasibility checks, we keep track of possibly adjusted maximum shift values $(ms^{TW})'$ for depots, which can decrease due to the delays. We take the resulting smaller extension size of depot visits into account when performing the feasibility check on endurance. We declare a sub-block to be risk-feasible if it passes both checks.

A sub-block with location set L'' is feasible with respect to time windows and the planning horizon if and only if the delayed arrival time arr'_i satisfies $arr'_i \leq \min\{U_i, LRT_i^{HOR}\}$ for each task $i \neq j$ in L''. We omit the starting task of the sub-block j itself, since a check for j would not fit the picture of judging the risk of skipping more important tasks after j.

Before performing the endurance check, we cut off the sub-block right before the first depot we encounter, if any depot is included. This reflects that delays in some trip cannot have a negative effect on the schedule for any subsequent trip. In fact, they might even lead to a smaller duration of the next trip by a decreased waiting time in the trip. Therefore, endurance problems that appear after a

recharging depot can never be solved by removing task j, which undermines the idea of a risk check for task j.

A sub-block with location set L'', which was possibly reduced by the above procedure, is feasible with respect to endurance if and only if the adjusted departure time dep'_i satisfies $dep'_i \leq LRT_i$ for each task $i \neq j$ in L''. Again, we omit j from the check. We can possibly repair initial endurance violations by extending the stay at the previous depot, as we discussed in Section 5.2. In this context, we use the possibly decreased maximum shift values $(ms_d^{TW})'$ as an additional upper bound on the maximal extension size of a depot d.

In the special case that we still have to wait before a location j' after the starting task j, despite of the incorporated delays and before any infeasibility problems, we consider the sub-block to have passed the check. This waiting time can be due to a release time if j' is a task or an extended stay if j' is a depot. Since removing the starting task j would not decrease the starting time of j', any feasibility problems that occur after j' can never be repaired by removing j. Consequently, the part of the sub-block after j' is irrelevant in the risk check for task j.

Example

For example, consider the second block $e-h$ from Figure 6.2 again, for which we derived the risk function $f(e) = f(f) = f(g) = 2$ and $f(h) = 8$ in Section 6.2.2. In the left part of Figure 6.6, the corresponding delay percentages are shown, which lead to adjusted travel times $t'_{der} = t'_{efr} = t'_{fgr} = 10.33$ and $t'_{ghr} = 11$, and adjusted service times $s'_{er} = s'_{gr} = 10.83$ and $s'_{hr} = 12.5$. Recall that for the depot node f, the minimal rest time remains fixed. Using $dep_d = 60$ (see Table 6.1) as the base time, this leads to the following delayed partial schedule: $arr'_e = 70.33, dep'_e = 81.16, arr'_f = 91.5, dep'_f = 101.5, arr'_g = 111.83, dep'_g = 122.67, arr'_h = 133.67, dep'_h = 149.17$. It can be easily checked in Table 6.1 that none of the arrival times violate a time window deadline, and none of the departure times violates a latest return time.

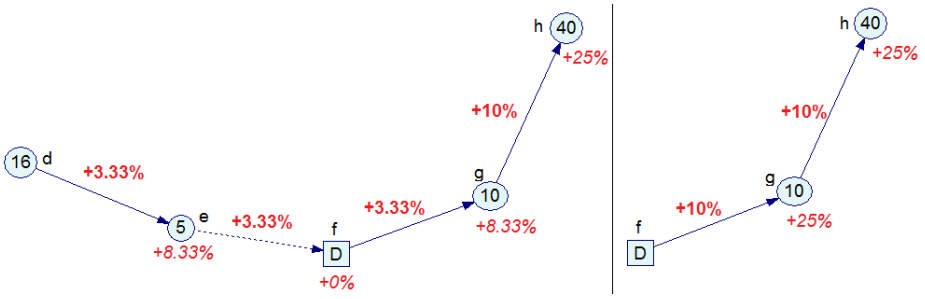

Figure 6.6: Delay percentages for the blocks $e-h$ and $g-h$, in boldface/italic for travel and service times, respectively.

Consider the last block $g - h$ from Figure 6.2, which is nested in $e - h$ and has the risk function $f(g) = f(h) = 4$. This leads to delay percentages as shown in the left part of Figure 6.6, and worst-case travel and service times of 11 and 12.5, respectively. Using $dep_f = 100$ as the base time, we obtain the following delayed partial schedule: $arr'_g = 111, dep'_g = 123.5, arr'_h = 134.5, dep'_h = 147$. Observe that the value for dep'_g is higher than in the feasibility check for block $e - h$. This shows that feasibility checks in nested blocks are not necessarily redundant. In fact, these delays even lead to a time window violation for task h, since arr'_h exceeds its deadline $U_h = 134$. Therefore, this block is risk-infeasible and hence, we declare route 1 from the working example in Figure 6.1 to be risk-infeasible.

6.2.4 Remarks

We stress that a block starting at task j can only exist if j is succeeded by a task j' has value $v_{j'} > v_j$, or if j is the final task before a depot. Therefore, more risk checks are performed for a route with increasing values (e.g. 10-20-70-80) than for a route with decreasing values (e.g. 80-70-20-10). Consequently, a route with increasing values has a lower probability of being risk-feasible. Thus, using the risk function will encourage important tasks to be performed first, which is an important and desirable side-effect, because this typically increases the probability that these important tasks are actually realized.

Observe that the risk function concept does not impose any additional feasibility requirement in routes with tasks with equal values, like route 2 from Figure 6.1. In particular, it does not assist in avoiding a tight schedule filled exclusively with important tasks. Such routes are undesirable because of the high probability of being forced to skip an important task in real-time. Instead, it would be better to distribute the important tasks better over the resources. In Section 6.3, we discuss a concept that might help in this matter and thus forms a valuable addition to the risk function.

The risk function can be incorporated as an additional constraint in the Tabu Search process, as we discuss in Section 7.2. It can be easily incorporated in a rescheduling policy for the online stage as well, as we discuss in Section 8.1.2. At each departure event, we can perform a risk check on a block starting at the upcoming task j, if such a block exists, given the current delay.

6.3 Worst-case fraction

The worst-case fraction sets a requirement on the minimal performance if the worst-case scenario unfolds in real-time, in the sense that all travel and service times take their maximum possible values.

If an unfavourable scenario unfolds in real-time, we can typically not perform all tasks scheduled in our solution. Given some passive rescheduling policy in which the only action is to delete tasks, i.e. routes are not extended by tasks that were initially unscheduled or assigned to a different resource, we can determine for each route which tasks would still be performed under a given scenario. Thus, we can compute the *worst-case fraction* of a schedule as follows:

$$\text{Worst case fraction} = \frac{\text{Total value of remaining tasks in worst case scenario}}{\text{Total value of all scheduled tasks}}$$

This value is a robustness measure for a solution that indicates how bad an initially feasible solution can become in real-time. By setting a lower bound on the worst-case fraction, denoted by $\underline{WCF}$, we aim at ensuring a certain minimal performance of a solution for any scenario.

An important remark, however, is that the worst-case scenario might not yield the lowest total value in real-time after all. For instance, consider a route from which we remove some task t with a low value in the worst-case scenario. In a more favourable scenario up to task t, we decide to include task t. However, if the realized scenario becomes very bad from arr_t onwards, we might be forced to skip a more important task after t. Thus, we obtain a lower value than in the case that we would have skipped task t in the first place. Combining the worst-case fraction with the risk function might help to avoid such situations, but it does not guarantee the performance of important tasks.

A desirable side-effect of the worst-case fraction is the encouragement to obtain a better distribution of tasks over the resources. If the schedule for a single resource is very tight in some solution, while another resource only performs a few tasks, the worst-case performance is typically bad. This is a valuable addition to our Tabu Search algorithm, since the search process currently has no incentive to prefer solutions in which tasks are distributed over the resources more evenly, since the only objective is to maximize the total task value. Moreover, the worst-case fraction can assist in obtaining a better distribution of *important* tasks as well, so that repairing a route in real-time can more often be done by deleting a task with a low value.

Example

For instance, set a worst-case requirement $\underline{WCF} = 0.8$ and consider the solution in Figure 6.1 from the working example in Section 6.1 again. Observe that route

1 contains more tasks, as well as more important tasks. As the total value of the solution is 151, we must at least obtain an online value of 120.8 in the worst-case scenario in order to meet the requirement $\underline{WCF} = 0.8$. Suppose we use a naive rescheduling policy of always heading for the next task, while we immediately head for the next location when we arrive late at a task, and we abort the performance of a task when a latest return time is attained.

Applying this rescheduling policy to the solution in the working example yields the worst-case schedules shown in Tables 6.3 and 6.4. For route 1, we violate the time windows of tasks d and h, since $arr'_d = 58 > U_d$ and $arr'_h = 137 > 134$. Therefore, we skip tasks d and h. For route 2, we are unable to complete task y before its latest return time for endurance $LRT^{END} = 89$, since we arrive at $arr_y = 80.5$ and the worst-case service time of task y is equal to 11. In order to avoid endurance violations, we abort task y at its latest return time for endurance.

Since we delete tasks d, h and y, the remaining value is 90, which does not satisfy the worst-case fraction requiremnent. Therefore, we declare this route to be WCF-infeasible and forbid this solution. It can be easily checked that this solution is WCF-feasible if we extend route 2 by a second trip that only consists of task h, which reflects that WCF-feasible solutions often have a better distribution of (important) tasks.

Location	Arrival	Departure	Time window	LRT^{END}	LRT^{HOR}
a	-	0	[0, 300]	-	-
b	11	23.5	[0, 100]	89	289
c	34.5	47	[0, 100]	89	289
d	**58**	58	[0, **55**]	89	289
e	69	81.5	[0, 70]	89	289
f	92.5	102.5	[0, 300]	-	-
g	113.5	126	[0, 200]	189	289
h	**137**	137	[0, **134**]	189	289
i	148	-	[0, 300]	-	-

Table 6.3: Worst-case schedule for route 1 in Figure 6.1.

Location	Arrival	Departure	Time window	LRT^{END}	LRT^{HOR}
u	-	0	[0, 300]	-	-
v	11	23.5	[0, 300]	89	289
w	34.5	47	[0, 300]	89	289
x	58	70.5	[0, 300]	89	289
y	80.5	**89**	[0, 300]	**89**	289
z	100	-	[0, 300]	-	-

Table 6.4: Worst-case schedule for route 2 in Figure 6.1.

Chapter 7

Generating robust solutions with Tabu Search

In this chapter, we discuss how to adjust the Tabu Search heuristic, introduced in Chapter 5, in such a way that we can generate robust solutions with this heuristic, using among others the robustness concepts from Chapter 6.

First, we propose a procedure to extend depot visits in Section 7.1. Afterwards, in Sections 7.2 and 7.3, we explain how to incorporate the risk function and the worst-case fraction in the Tabu Search heuristic, respectively. Incorporating these robustness concepts can provide guidance in finding a robust solution, whereas the extension of depot visits improves the robustness of a given feasible schedule.

7.1 Final extension of depot visits

For repairing endurance problems when constructing a feasible schedule in the Tabu Search, we use a minimal possible extension size of depot visits (see Section 5.2.2). For a final schedule, however, a minimal extension size leads to an unstable route, in the sense that even the smallest delay in the subsequent trip would lead to endurance problems. Thus, applying an additional extension of depot visits might improve the robustness of the schedule. This procedure can be regarded as a post-processing step of the search process, or as a pre-processing step before entering the online stage of the problem.

Recall from Section 5.2.2 that for a task j, a time window buffer is given by $U_j - arr_j$ and an endurance buffer is given by $LRT_j^{END} - arr_j$. An extension of a depot visit is beneficial for the endurance buffers of the tasks in the subsequent trip, provided that the extension reduces the total waiting time in the trip,

while it always deteriorates time window buffers. In Algorithm 6, we show a procedure to find an extension size that balances the buffers with respect to time windows and endurance (line 5-12). When using the risk function, we also take decreased time window and endurance buffers encountered in feasibility checks of sub-blocks into account. As a final step, we ensure that we never arrive at the first task after the depot before its release time in the worst-case scenario (line 13-14).

Due to bounding the extension size from above by the total waiting time in the trip, extending a visit to depot d never leads to a later arrival time at the next depot j. Moreover, an extension in some trip does not affect the maximum shift values for previous trips, since the extension just moves the wait to the depot instead of removing it completely. Thus, the procedure can be applied independently to all trips of a route.

Algorithm 6 Determining a final extension size of a depot visit

1: **Input:** Trip for resource r from depot d to depot j with task set $Trip_j$, starting with task a.
2: **Input:** For each task $i \in Trip_j$, endurance buffers $bufEnd_i = LRT^{END} - dep_i$ and total waiting time $tripWait_i$ in the trip up to and including i.
3: **Input:** Maximal travel time deviation fraction σ_{TT}.
4: **Output:** Additional extension size Δ for depot d

5: Set $bufTW = ms_d^{TW}$ and $\Delta = 0$.
6: **for all** tasks $i \in Trip_j$ **do**
7: Set $bufEnd'_i = bufEnd_i + \Delta$ and $tripWait'_i = tripWait_i - \Delta$
8: **if** $bufTW > bufEnd'_i$ **then**
9: Set $\Delta = \Delta + \min\{tripWait'_i, (bufTW - bufEnd'_i)/2\}$.
10: Set $bufTW = bufTW - \Delta$.
11: **end if**
12: **end for**
13: Set $minExtendBy = L_a - (1 + \sigma_{TT})\bar{t}_{dar} - dep_a$.
14: Set $\Delta = \max\{\Delta, minExtendBy\}$.

7.2 Incorporating the risk function

In this section, we discuss how to incorporate the risk function as an additional constraint for a successful move in the Tabu Search heuristic. In order to tackle the Extended Security Routing problem better, we introduce an alternative procedure to restore the risk-feasibility of a solution.

For a candidate move in the Tabu Search algorithm, a risk check on the routes involved in this move is performed after the fast feasibility check and constructing a feasible schedule. If any involved route turns out to be risk-infeasible,

we reject the candidate move, even though it is feasible with respect to time windows, endurance, and the planning horizon.

As an alternative, we can use a procedure to *restore* the risk-feasibility of a solution, rather than only accepting candidate moves that lead to risk-feasible solutions. In particular, we ignore risk-feasibility throughout the iteration when trying candidate moves. As a post-processing step after each iteration, we apply the procedure below to restore the risk-feasibility of the best solution of the iteration.

We loop through all routes that contain recurrent tasks and check their risk-feasibility. If a route is risk-infeasible, we remove the starting task of the sub-block in which we encounter this infeasibility. If the route becomes risk-feasible, possibly after removing multiple tasks, we proceed to the next route in the solution and apply the same procedure. Note that we might need to return to routes from which we have already removed tasks to ensure their risk-feasibility, due to removal of recurrent tasks in other routes. We terminate the restore procedure if all routes have become risk-feasible.

This alternative approach with the restore procedure is more suitable for the Extended Security Routing problem, due to the presence of recurrent tasks. Recall from Section 2.2 that the value of recurrent tasks depends on the number of tasks from the corresponding recurrent set that appear in the solution. Therefore, inserting a recurrent task in some route causes a value increase for the other tasks in the same recurrent set as well, even if these tasks are assigned to different resources. Similarly, a removal leads to a value decrease of tasks in the same recurrent set. Therefore, the structure of blocks and sub-blocks might change, as well as the value ratios within these (sub-)blocks. In particular, the risk checks cannot be performed separately per resource.

Moreover, when using the standard risk check, a counterintuitive behaviour regarding addition of tasks with high value can occur. For instance, consider an example in which we try to add a task j with value 100 to a route where all current tasks have value 1. The resulting route might be risk-infeasible, due to worst-case delays in the sub-block starting at the task before j, if such a task exists. When using the alternative option, we temporarily ignore this risk-infeasiblity and accept the insertion. After the iteration, we restore the risk-feasibility of the solution, which leads to the removal of one or more tasks with value 1, while the important task with value 100 remains.

7.3 Incorporating the worst-case fraction

The worst-case fraction can be easily incorporated in the Tabu Search heuristic as well. When a candidate move is feasible, we construct a feasible schedule as usual, by Algorithm 3. Afterwards, we check whether the candidate solution

satisfies the worst-case fraction requirement. In general, this procedure consists of the following steps.

1. Apply the depot visit extension procedure to all routes in the solution.
2. Apply a given rescheduling policy to the solution, in the worst-case scenario for the online stage.

Obviously, it might be useful to store some results for the worst-case scenario, instead of performing the above steps for all routes in each candidate move. Nevertheless, incorporating this worst-case fraction remains a relatively time-intensive procedure. This is particularly true when the lower bound on the worst-case fraction is high, since this leads to a large number of unsuccessful worst-case fraction checks.

Chapter 8

Rescheduling policies for the online problem

The previous chapters covered the construction of a solution in the offline stage. In Chapter 7, we discussed how to generate robust offline solutions with the Tabu Search heuristics, using robustness concepts from Chapter 6. However, due to the stochastic travel and service times, we will typically not be able to perform the schedules as planned in reality.

The online stage of the problem requires adjustments to the offline solution in real-time, in which we repeatedly make decisions based on the realized scenario up to some point in time, using rescheduling policies. Rather than maximizing the objective value in the offline stage, we aim for maximizing the obtained value in the online stage, as well as for being able to stick to the original schedules as much as possible.

In Section 8.1, we propose a passive rescheduling policy for a given route. Afterwards, we discuss the necessary adjustments to make this policy suitable when using the risk function. Furthermore, we generalize the policy to handle all routes of a solution simultaneously, which is necessary for the Extended Security Routing problem.

The above rescheduling policies are *passive* in the sense that we do not add initially unscheduled tasks in real-time, and we do not reassign scheduled tasks to different resources. These passive policies allow for a fair judgement of the performance of the robustness concepts, compared to more active policies in which the initial offline solution might be completely transformed.

However, the passive policies are focused on unfavourable scenarios. Ideally, we would be able to take advantage of favourable scenarios. For this purpose, we describe a more active policy in Section 8.2. As the focus of this thesis is on the robustness concepts, we did not implement this policy.

8.1 Passive rescheduling policies

We propose a passive rescheduling policy for a given route in 8.1.1. Afterwards, we explain how to incorporate the risk function in this policy in 8.1.2. In Section 8.1.3, we generalize the rescheduling policy in such a way that we can handle all routes of a solution simultaneously in real-time.

8.1.1 Rescheduling policy for a given route

We propose a passive rescheduling policy in which we perform feasibility checks on the future part of the route at each departure event, in order to decide whether or not to head for the upcoming task. In particular, we only head for a task if we consider that to be a responsible choice, regarding the importance of the upcoming task and the current feasibility state of the future part of the route. At each arrival event at a task, we only check whether we arrived before its deadline and start performing the task. However, we abort the execution of tasks when a latest return time with respect to endurance or the planning horizon is reached. The full rescheduling policy is shown in Algorithm 7 and is discussed below. The line numbers in this section refer to this algorithm.

The real-time route starts by leaving the depot at the planned departure time. Afterwards, we compute realized arrival and departure times based on simulated travel and service times, in a loop through all locations. If we encounter a depot j, we assign a penalty if the realized arrival time violates the planning horizon or if the total trip duration exceeds the endurance (line 13). For the waiting policy when arriving at a depot, recall from Section 7.1 that we determine a balanced departure time from each depot visit before entering the online stage. Therefore, we never leave earlier than this balanced departure time (line 14).

Before heading to a task j, we check whether it is responsible to go there (line 17-32). If we expect the future part of the route, restricted to subsequent tasks with higher value than v_j, to become infeasible when including task j in the real-time route, we decide to skip j (line 27-29). The aim of the restriction is to avoid deleting tasks too early, which might turn out to be unnecessary in case of favourable realizations. Meanwhile, we still protect the tasks with higher value. Note that we cannot repair endurance violations now, since we cannot extend a stay at a depot in history.

If the above future feasibility check is successfully passed, we head for task j (line 33). If we arrive late, we delete task j (line 34-36). Otherwise, we start performing j. If we have not completed j yet at its minimum latest return time, we interrupt the performance of j and head for the depot (line 37-41).

The quality of the real-time route can be determined by the obtained value $\sum_{i \in I^R} v_{ir}$ of the completed tasks and its total penalty pen. The number of removed tasks indicates how close we were able to stay to the original route.

Algorithm 7 The standard passive rescheduling policy

Input: Resource r with endurance E_r and minimal rest time R_r.
Input: Route with locations L and tasks $I \subset L$, starting at depot d_0.
Input: Original schedule with arrival arr and departure times dep.
Input: Latest return times LRT_t^{HOR} for all tasks $t \in I$.
Input: Simulated travel times t_{ijr}^R and service times s_{jr}^R for any $i, j \in L$.
Input: Penalty functions $horPen : \mathbb{R} \to \mathbb{R}_+$ and $endPen : \mathbb{R} \to \mathbb{R}_+$, with violation size as input and corresponding penalty as output.
Input: Previous depot d, previous location i. In the k-loop, d' and i' represent previous the depot/location for the future location k.
Output: Realized route with completed tasks I^R, total penalty pen, and arrival and departure times arr^R and dep^R.

Initialize: $d = i = d_0, I^R = I, pen = 0, dep_d^R = dep_d$.
for all locations $j \in L \backslash \{d\}$ **do**
  if j is a depot **then**
    Set $arr_j^R = dep_i^R + t_{ijr}^R$.
    Set $pen = pen + horPen(arr_j^R - U_j) + endPen(arr_j^R - dep_d^R - E_r)$.
    Set $dep_j^R = \max\{arr_j^R + R_r, dep_j\}$.
    Set $d = i = j$.
  else
    Initalize: $time = dep_i^R, i' = i, d' = d$.
    for all locations k from j onwards **do**
      Set $\overline{arr}_k = time + \bar{t}_{i'kr}$ and $\overline{dep}_k = \max\{\overline{arr}_k, L_k\} + \bar{s}_{kr}$.
      if k is a depot **then**
        Set $d' = k$ and $dep'_{d'} = \overline{dep}_k = \max\{\overline{dep}_k, dep_k\}$.
      else
        if $v_k \leq v_j$ **then**
          Break the k-loop and head for task j.
        end if
        Compute $\overline{LRT}_k^{END} = dep'_{d'} + E_r - \bar{t}_{kdr}$.
        if $\overline{arr}_k > U_k$ **or** $\overline{dep}_k > \min\{\overline{LRT}_k^{END}, LRT_k^{HOR}\}$ **then**
          Skip: set $I^R = I^R \backslash \{j\}$ and go to next j.
        end if
      end if
      Set $time = \overline{dep}_k$ and $i' = k$.
    end for
    Set $arr_j^R = dep_{ijr}^R + t_{ijr}^R$ and $i = j$.
    if $arr_j^R > U_j$ **then**
      Late: set $I^R = I^R \backslash \{j\}, dep_j^R = arr_j^R$ and go to next j.
    end if
    Set $dep'_j = \max\{arr_j^R, L_j\} + s_{jr}^R$.
    Set $LRT_j^{END} = dep_d^R + E_r - \bar{t}_{jdr}$.
    if $dep'_j > \min\{LRT_j^{END}, LRT_j^{HOR}\}$ **then**
      Abort: set $I^R = I^R \backslash \{j\}, dep_j^R = LRT_j$ and go to next j.
    end if
    Set $dep_j^R = dep'_j$ and $i = j$.
  end if
end for

Example

As an example, consider the route from Figure 8.1 and the corresponding schedule from Table 8.1. For simplicity, we assume that all travel and service times are 10 before recharging depot d, and 20 afterwards. For the real-time scenario, suppose that all travel and service times are delayed by 10%. Furthermore, suppose that an endurance of 105 is given.

We start by leaving depot a at the planned time $dep_a = 100$. At time $arr'_b = 111$, we confirm that we arrived at task b in time and start performing b. After completion of b, we check whether we expect to arrive in time at the upcoming task c and the more important next task e, assuming a mean-case scenario from time $dep'_b = 122$ onwards. We obtain $\overline{arr}_c = 132, \overline{dep}_c = 142$ and $\overline{arr}_e = 320, \overline{dep}_e = 340$, which gives no feasibility problems. Therefore, we head for task c and arrive there at time $arr'_c = 133$. Since $U_c = 132$, we are too late to start this task. Instead, we head for depot d.

At depot d, we wait for the originally planned departure time $dep_d = 300$ to leave the depot. For the upcoming task e, we expect to arrive and depart in time, since the original schedule is feasible. We arrive in time at task e at time $arr'_e = 322$ and complete task e at time $dep'_e = 344$. At this time, we check for the only remaining task f whether we expect to arrive and depart in time, given a mean-case scenario. Since $\overline{arr}_f = 364 < U_f$ and $\overline{dep}_f = 384 < \min\{LRT_f^{END}, LRT_f^{HOR}\}$, we decide to head for task f. After arrival at time $arr'_f = 366$, we start performing task f, although we do not expect to finish this task in time anymore. Indeed, at time $LRT_f^{END} = 385$, we have not completed task f yet, so we abort task f and return to the depot.

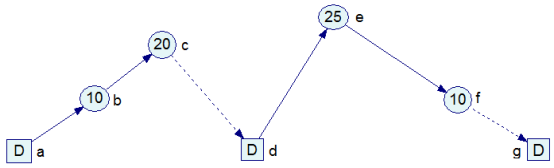

Figure 8.1: Example route for the online stage.

Location	Arrival	Departure	Time window	LRT^{END}	LRT^{HOR}
a	-	100	[0, 600]	-	-
b	110	120	[110, 120]	195	590
c	130	140	[120, 132]	195	590
d	150	300	[0, 600]	-	-
e	320	340	[320, 400]	385	580
f	360	380	[320, 400]	385	580
g	400	-	[0, 600]	-	-

Table 8.1: Original schedule for the route in Figure 8.1.

8.1.2 Adjustments when using the risk function

Recall from Section 6.2 that the main goal of the risk function is to determine beforehand whether it is safe enough to perform a task, when taking the risk of skipping a more important task in the future into account. Next to its contribution to a robust solution, this idea is useful in real-time as well.

For the rescheduling policy using the risk function, we use the core of Algorithm 7 with an adjusted future feasibility check at each departure event. Instead of checking feasibility in the mean-case scenario for the future part of the route, restricted to tasks that are more important than the upcoming task (line 17-32), we perform the following:

1. Check whether we expect to arrive at j and depart from j in time.
2. Perform a risk check on the block starting at j, if such a block exists, using the current time of the departure event as the base time.

Whenever we encounter an infeasibility in the above checks, we decide to skip j. Note that we only consider possible blocks starting at the upcoming task, given the scenario up to the current point in time. Again, this choice aims to avoid removing tasks too early and unnecessarily. The above procedure helps to avoid performing tasks at the cost of skipping more important tasks, although it does not guarantee the performance of more important tasks in general.

Suppose that the route in Figure 8.1 is risk-feasible for some delay function. As before, we complete task b at time $dep'_b = 122$. For the decision whether to head for task c or not, we check whether we expect feasibility problems regarding task c itself, as we did before. Furthermore, we perform a risk check on the block $c - e$, using $dep'_b = 122$ as the base time. We head for task c if and only if this block passes the risk check. Since no blocks start after task c, no additional adjustments to the original policy are required in this case.

8.1.3 Rescheduling policy for a given solution

As we mentioned in Section 2.2, the value of recurrent tasks depends on the number of tasks from the same recurrent set that appear in the solution. Therefore, the removal of a recurrent task in real-time leads to a lower value of tasks from the same recurrent set, both for completed and future tasks. This leads to dependencies between routes and implies that the online problem of the Extended Security Routing problem is not separable per route. Thus, we need to handle all routes simultaneously in real-time.

For this purpose, we keep track of the next departure times for each route. We terminate if no next departure times remain, i.e. if all routes have been completed. In each iteration, we consider the route with the earlier next departure time and determine the upcoming location j. If this is a depot, we apply the same procedure as before (line 12-15 of Algorithm 1). Otherwise, we

decide whether to head for task j or not. Again, we can basically use the same procedure as before, as described line 17-32 of Algorithm 7, or its risk variant described in Section 8.1.2. However, for all recurrent tasks in the route, we use the values based on the most recent information on removal of recurrent tasks in the routes.

8.2 Active rescheduling policy

In the passive rescheduling policies described above, our only possible action is to delete tasks from routes in unfavourable scenarios. However, we might be able to control the damage better if we would be able to reassign the task to a different resource, or if we could replace the deleted task by a slightly less important task. Moreover, in favourable scenarios, we might have the possibility to add initially unscheduled tasks to the route.

By using more active policies, we could probably often increase the objective value obtained in real-time. However, the other criterion to stick to the original plan would probably deteriorate. Below, we describe a policy that aims to find a compromise between aiming for a higher objective value and sticking to the original plan.

For a departure event after completing the visit to location i and before heading to a task j, we can check whether we can add an unscheduled task between i and j, or whether we can replace task j by a more important unscheduled task. Note that these actions correspond to the local moves Addition and Swap in the Tabu Search algorithm, see Section 5.1.

Observe that we only consider a single insertion position and we still do not reassign tasks to different resources. In particular, we do not consider the local moves Change and Exchange from Section 5.1. While the obtained value will typically be lower than for more active rescheduling policies, this policy can be easily implemented in practice and allows for very quick decisions. Moreover, it meets the practical wish to avoid nervousness and to stay close to the original solution in real-time.

Chapter 9

Data description

For evaluating the quality of the proposed methods and concepts, we consider different datasets. In this chapter, we describe the datasets we used for the original problem and the extended problem. Furthermore, we describe the framework we used for the simulations of travel and service times.

9.1 Instances for the original problem

For the original Security Routing problem, we use four instances, to which we refer by Instance 1 - 4, based on test instances from Vansteenwegen [2009]. These are the same instances as Instance 2 - 5 used in Breugem et al. [2015].

A very important aspect of these datasets is the strict priority distinction between tasks, in the sense that a task is more important than the set of all tasks with lower value. That is, for $\mathcal{I}' = \{i' \in \mathcal{I} \mid v_{i'} < v_i\}$, we have $v_i > \sum_{i' \in \mathcal{I}'} v_{i'}$, for each task $i \in \mathcal{I}$. In these datasets, we have priority levels $P = \{1, 2, 3\}$, which we need to convert to values w_p. Let I_p be the set of tasks with priority p. We set $w_3 = 1, w_2 = 1 + |I_3|$ and $w_1 = 1 + w_2|I_2| + w_3|I_3|$. Note that these values satisfy the required strict priority distinction, and that tasks of priority 1 are most important.

All instances contain 100 tasks and 2 depots, and the planning horizon ends at $T = 600$. Other characteristics of the instances are summarized in Table 9.1. Instances 1 and 4 share a common map with geographical locations of tasks and depots (see Figure 9.1), as well as Instances 2 and 3 (see Figure 9.2). The tasks are clustered for Instances 1 and 4, while they are scattered for Instances 2 and 3. Contrary to the geographical locations, the priorities, time windows, and service times of tasks can be different per instance.

	Instance 1	Instance 2	Instance 3	Instance 4
#tasks	39 / 30 / 31	35 / 41 / 24	32 / 41 / 27	39 / 30 / 31
values	992 / 32 / 1	1050 / 25 / 1	1176 / 28 / 1	992 / 32 / 1
max obj	39679	37799	38807	39679
avg tw	74.6	117.2	189.9	67.0
avg min dist	2.7	5.2	5.2	2.7
#resources	5	10	10	5
avg speed	21.4	25	30	21.4
avg endurance	260	260	310	260
avg rest time	58	58	42	58
avg service time	24.8	43.9	33.8	24.9

Table 9.1: Characteristics of the data instances for the original problem. For tasks, we report the number of tasks per priority, the corresponding values, the resulting maximal objective, the average size of time windows, and the average distance from a task to the nearest other task. For resources, we report the number of resources, the average speed, the average endurance, the average minimal rest time, and the average service time of resources for tasks.

Since Instances 2 and 3 have larger time windows and more resources than Instances 1 and 4, it can be expected that more tasks can be scheduled for the former instances. Furthermore, it can be expected that more tasks can be scheduled for Instance 3 than for Instance 2, due to larger averages for travel speed and endurance, and smaller averages for rest time and service time.

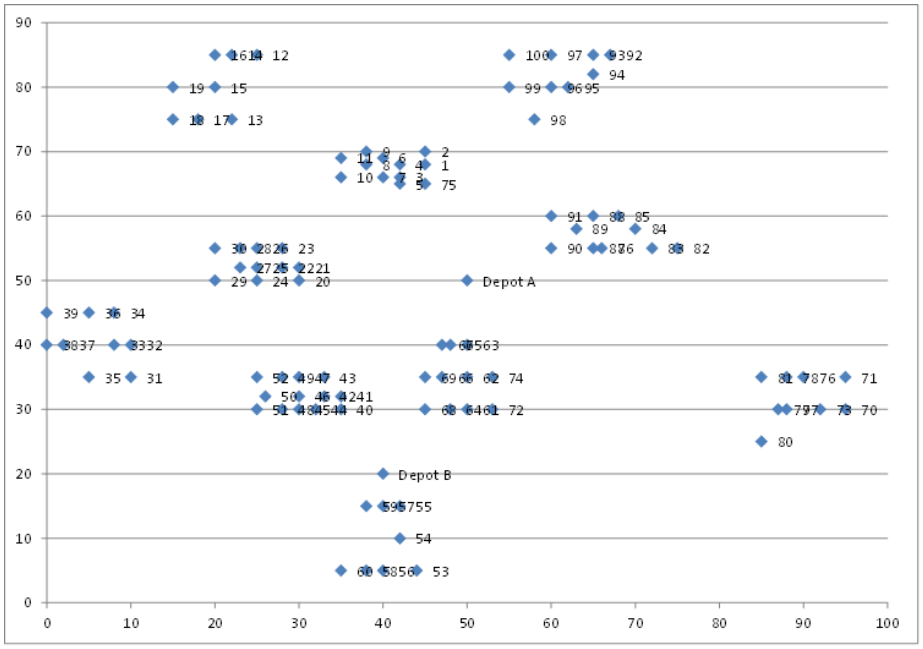

Figure 9.1: The map of tasks and depots for Instances 1 and 4.

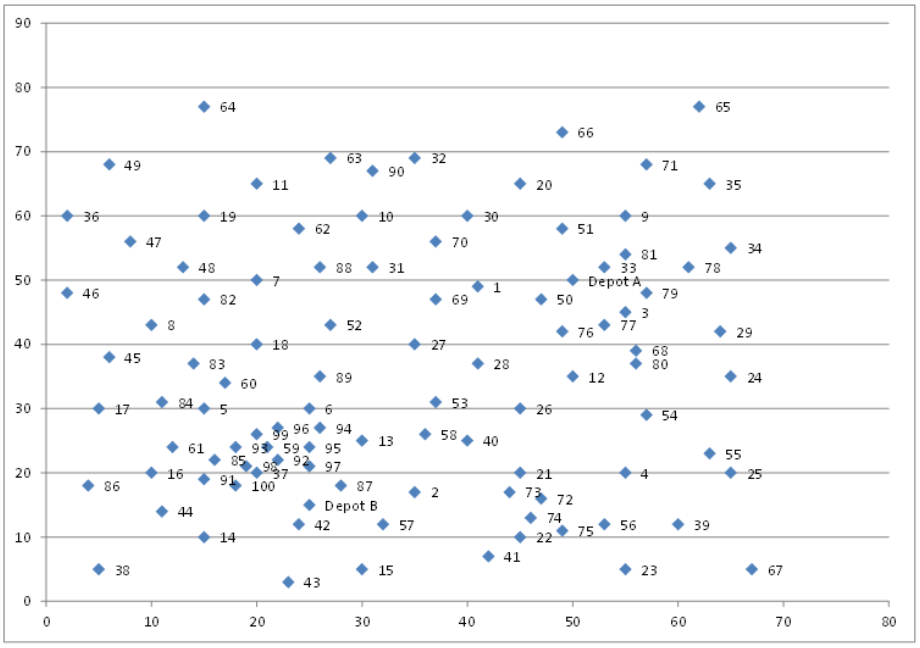

Figure 9.2: The map of tasks and depots for Instances 2 and 3.

9.2 Instances for the extended problem

For the Extended Security Routing problem, we need datasets with a more general value structure, resource-dependent values and recurrent tasks. For this purpose, we adjust and extend the value structure as follows:

1. For each task with priority 1/2/3, assign a random base value in $[11, 20]$, $[6, 10]$, or $[1, 5]$, respectively.
2. For each task, assign a random quality between 0.8 and 1 to each mean that can perform the task.

We incorporate recurrent tasks in the instances by the following procedure:

1. For each priority, randomly choose 3 tasks to become part of a recurrent set.
2. Each priority gets one recurrent set for each of the sizes 3, 5, and 7.
3. We assume that in a recurrent set, service times and qualities are equal.
4. Assign a time window $[iT/|S|, (i+0.5)T/|S|]$ to the i-th task in a recurrent set S.

Note that each of the extended instances contains 136 tasks now. The time windows of the recurrent tasks are chosen such that they are disjoint and evenly distributed over time. We refer to these extended datasets by Instances 5 - 8.

Additionally, we use a dataset that was used in an experiment in a military context, In this dataset, to which we refer as Instance 9, all 126 tasks belong to some recurrent set of either size 4 or size 5. The planning horizon ends at $T = 660$. Since we only have 3 resources and the service times of tasks are relatively short, we typically obtain routes with a lot of tasks and recharging depots, especially for the fast resources.

9.3 Simulation of travel and service times

We assume that travel and service times can deviate at most 10% and 25% from their expected values. That is, we set $\sigma_{TT} = 0.1$ and $\sigma_{ST} = 0.25$. For the online stage of both problems, we draw realizations of travel and service times from a uniform distribution over $[(1 - \sigma_{TT})\bar{t}_{ijr}, (1 + \sigma_{TT})\bar{t}_{ijr}]$ and $[(1 - \sigma_{ST})\bar{s}_{jr}, (1 + \sigma_{ST})\bar{s}_{jr}]$, respectively.

Chapter 10

Results

In this chapter, we show and analyze the results of applying our methods and concepts to the datasets described in Chapter 9. In Sections 10.1 and 10.2, we discuss the results for the original Security Routing problem and the Extended Security Routing problem, respectively.

The experiments for the Tabu Search heuristic were performed on a computer with an Intel Core i5-2400 processor with CPU running at 3.1 GHz and 4 GB of RAM-memory. The experiments for the LRCG approach were performed on a computer with an Intel Core i3-2310M processor with CPU running at 2.1 GHz and 4 GB of RAM-memory.

10.1 Results for the original problem

The majority of the results for the original Security Routing problem concerns the performance of the Tabu Search algorithm. In particular, we focus on the effect of using robustness concepts on the online performance of the solutions. Furthermore, we discuss the results of applying the LRCG method to the datasets. In particular, we compare the obtained upper bound estimators on the optimal total task value and the obtained feasible solutions to the best results for the Tabu Search algorithm.

10.1.1 Tabu Search and robustness concepts

In order to judge the effect of including the reordering mechanism and the robustness concepts on the offline solution, we generated solutions for all of the following *approaches*:

- STD: The Tabu Search algorithm from Chapter 5, but without reordering.
- REORDER: Similar to STD, but now including the reordering procedure.
- WCF90: Similar to REORDER, but with $\underline{WCF} = 0.9$.
- WCF99: Similar to WCF90, but with a more strict bound $\underline{WCF} = 0.99$.
- RISK: Similar to REORDER, but now we only accept risk-feasible routes.
- RISKRES: Similar to RISK, but with *restoring* risk-feasibility of solutions.
- RWCF90: Similar to RISK, but accompanied by $\underline{WCF} = 0.9$.
- RRWCF90: Similar to RISKRES, but accompanied by $\underline{WCF} = 0.9$.

We refer to Chapter 6 for the robustness concepts of the risk function and the worst-case fraction. We refer to Chapter 7 for the way in which these concepts were integrated in the Tabu Search algorithm, as well as for the difference between only accepting risk-feasible routes and restoring risk-feasibility of a solution after each iteration.

For the approaches WCF90 and WCF99, we require that the majority of the offline solution can be executed in the worst-case scenario. For WCF90, we allow for some more flexibility. Given the priority structure of these datasets, only a few or no tasks of priority 1 can be removed in the worst-case scenario for WCF90 and WCF99, respectively. For the four approaches using the risk function, we use a delay function that implies that each value difference leads to worst-case delays, e.g. $g(R) = \min\{(R-1)/5, 1\}$, in order to reflect the strict priority distictions in these datasets.

For all approaches, we performed 10 runs of 15 minutes in the offline stage. For each offline solution, we applied a rescheduling policy for 2000 simulations of travel and service times, of which 1000 were applied after a final depot visit extension (see Section 7.1).

Offline stage

For the offline stage, we are particularly interested in the added value of the reordering procedure of tasks within routes. Furthermore, we report the number of iterations in the Tabu Search algorithm, in order to illustrate the effect of including the reordering procedure and the robustness concepts on the speed of the algorithm. Moreover, we check whether using the robustness concepts affects the order of tasks within routes and the distribution of tasks over resources.

Table 10.1 summarizes the results obtained. Note that for Instance 3, the optimal objective value is attained for each approach. For Instance 2, all tasks of priority 1 and almost all tasks of priority 2 are assigned for each approach as well. For Instances 1 and 4, we only have 5 resources available and hence,

the number of scheduled tasks is considerably lower. This table also shows that approach WCF99 yields the most conservative solutions for all instances, which makes sense due to the very strict requirement that 99% of the scheduled task value should still be realized in the worst-case scenario.

The strategy of restoring the risk-feasibility of a solution after each iteration outperforms the standard risk-feasibility check for Instances 1 and 4, whereas the standard risk check gives slightly better results for Instance 2. Table 10.1 also shows that a more conservative approach like RISK sometimes yields a higher objective value than a basic approach like REORDER. As the risk function can be regarded as an additional feasibility constraint, this behaviour is somewhat counterintuitive. However, it appears that rejection of certain routes due to risk-infeasibility can lead the search process into promising regions of the (smaller) solution space more quickly.

The results for the approaches STD and REORDER show that the reorder procedure only slightly slows down the search process, but it does not improve the objective value significantly for these instances. Another measure for the usefulness of reordering is the success percentage, in the sense that the best objective value of the (sub)search is improved in the first iteration afterwards. For Instance 3, no reordering procedures were performed before attaining the maximal objective value. For approach REORDER and Instances 1/2/4, the average success percentages are given by $21.8\%, 10.4\%$ and 12.0%, respectively. These percentages suggest that the reordering procedure is particularly useful if there are still many unscheduled tasks remaining that can be inserted after reordering, as in Instances 1 and 4.

For investigating whether the risk function indeed leads to a preference of routes with decreasing values, we use the average number of less important tasks before a task of priority 1 or 2 as a measure. For investigating whether the worst-case fraction leads to a better distribution of scheduled tasks over the resources, we use the standard deviation of number of tasks per route as a measure. The corresponding results are shown in Table 10.2.

For Instances 1/2/4, the risk function clearly leads to a considerable decrease of the average number of less important tasks before tasks of priority 1 and 2. For Instance 3, this effect is also visible for priority 2, but not that much for priority 1. This might be due to the fact that this is an 'easy' instance in the sense that a large fraction of feasible routes is risk-feasible as well. In the more restrictive instances 1 and 4, the effect is the most significant.

On the other hand, the effect of using the worst-case fraction on the distribution of tasks over the resources is not clear for these instances. This might be partially due to the fact that a less even distribution of tasks might actually lead to more robust schedules for our heterogeneous resources. The effect is more clearly visible for the extensions of these instances, as we show in Section 10.2.

	STD	REORDER	WCF90	WCF99	RISK	RISKRES	RWCF90	RRWCF90
I1								
Obj	38405,2 (448,8)	38408,7 (452,3)	37122,9 (939,1)	33623,5 (1192,5)	38628,7 (435,9)	**38945,8 (0,4)**	37515,4 (619,9)	37828,3 (291,0)
#prio 1	38,4 (0,5)	38,4 (0,5)	37,1 (1,0)	33,5 (1,2)	38,7 (0,5)	**39,0 (0,0)**	37,5 (0,7)	37,9 (0,3)
#prio 2	9,7 (1,6)	9,8 (1,4)	9,9 (3,3)	12,1 (2,1)	7,4 (1,0)	**8,0 (0,0)**	9,8 (1,9)	7,2 (1,1)
#prio 3	2,0 (1,1)	2,3 (1,2)	2,9 (0,9)	4,3 (1,5)	1,5 (0,8)	**1,8 (0,4)**	1,8 (1,2)	1,1 (0,3)
#its	28248,4 (584,0)	25249,8 (1563,7)	8669,6 (804,5)	4003,4 (506,7)	15988,4 (173,4)	**13380,0 (484,4)**	12386,3 (412,8)	6115,7 (678,3)
I2								
Obj	37774,0 (16,0)	**37781,3 (7,2)**	37780,5 (6,2)	37753,6 (20,6)	37777,1 (7,1)	37764,6 (13,2)	37766.6 (11.6)	37757,3 (15,5)
#prio 1	35,0 (0,0)	**35,0 (0,0)**	35,0 (0,0)	35,0 (0,0)	35,0 (0,0)	35,0 (0,0)	35,0 (0,0)	35,0 (0,0)
#prio 2	40,6 (0,7)	**40,9 (0,3)**	40,9 (0,3)	39,9 (0,8)	40,9 (0,3)	40,5 (0,5)	40,5 (0,5)	40,2 (0,6)
#prio 3	9,0 (1,5)	**8,8 (1,2)**	8,0 (1,55)	6,1 (1,2)	4,6 (1,3)	2,1 (1,4)	4,1 (1,2)	2,3 (1,7)
#its	16586,5 (301,8)	**17278,0 (455,9)**	7394,2 (674,3)	2692,9 (220,7)	13744,4 (72,1)	9550,4 (344,8)	10600,4 (509,7)	4780,9 (444,5)
I3								
Obj	38807,0 (0,0)	**38807,0 (0,0)**	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)
#prio 1	32,0 (0,0)	**32,0 (0,0)**	32,0 (0,0)	32,0 (0,0)	32,0 (0,0)	32,0 (0,0)	32,0 (0,0)	32,0 (0,0)
#prio 2	41,0 (0,0)	**41,0 (0,0)**	41,0 (0,0)	41,0 (0,0)	41,0 (0,0)	41,0 (0,0)	41,0 (0,0)	41,0 (0,0)
#prio 3	27,0 (0,0)	**27,0 (0,0)**	27,0 (0,0)	27,0 (0,0)	27,0 (0,0)	27,0 (0,0)	27,0 (0,0)	27,0 (0,0)
#its	1056,4 (550,2)	**275,0 (0,0)**	434,4 (28,8)	381,0 (0,0)	626,5 (342,3)	1011,0 (464,3)	492,0 (0,0)	768,2 (626,6)
I4								
Obj	36987,8 (415,5)	36894,8 (644,4)	35001,3 (864,6)	31610,9 (459,5)	36738,1 (385,1)	**37768,7 (269,2)**	35604,4 (725,9)	35201,1 (369,0)
#prio 1	37,0 (0,5)	36,9 (0,7)	34,9 (0,9)	31,4 (0,5)	36,8 (0,4)	**37,9 (0,3)**	35,6 (0,8)	35,2 (0,4)
#prio 2	8,8 (1,1)	9,0 (1,7)	11,8 (2,4)	14,3 (1,3)	7,2 (1,0)	**5,3 (0,9)**	9,0 (2,3)	8,8 (1,3)
#prio 3	2,2 (0,4)	2,0 (0,9)	2,9 (0,9)	4,5 (1,2)	2,1 (0,9)	**2,3 (0,8)**	1,2 (1,1)	1,1 (0,3)
#its	29778,9 (2442,7)	24023,0 (1773,0)	8721,3 (487,3)	7037,8 (545,8)	16395,7 (176,6)	**14855,0 (130,0)**	12825,6 (882,9)	5968,7 (512,5)

Table 10.1: Offline results for Tabu Search: objective, number of tasks per priority, and number of iterations (speed). We report the average values and standard deviations (between brackets) over 10 runs of 15 minutes. For each instance, the results for the best approach are in boldface.

	STD	REORDER	WCF90	WCF99	RISK	RISKRES	RWCF90	RRWCF90
Instance 1								
#lower before prio 1	1,0 (0,3)	1,1 (0,4)	1,2 (0,4)	1,6 (0,4)	**0,3 (0,1)**	**0,3 (0,1)**	0,8 (0,4)	0,4 (0,1)
#lower before prio 2	0,2 (0,2)	0,2 (0,2)	0,1 (0,1)	0,4 (0,2)	**0,0 (0,1)**	**0,0 (0,0)**	**0,0 (0,0)**	**0,0 (0,0)**
St dev #tasks per route	5,3 (0,3)	5,3 (0,3)	5,2 (0,3)	5,1 (0,3)	5,0 (0,2)	5,2 (0,3)	5,1 (0,3)	**4,6 (0,2)**
Instance 2								
#lower before prio 1	1,9 (0,3)	1,8 (0,2)	2,0 (0,1)	1,7 (0,2)	1,3 (0,2)	**1,2 (0,2)**	1,3 (0,1)	**1,2 (0,2)**
#lower before prio 2	0,7 (0,2)	0,7 (0,1)	0,6 (0,1)	0,4 (0,1)	0,3 (0,1)	**0,1 (0,1)**	0,2 (0,1)	**0,1 (0,1)**
St dev #tasks per route	1,4 (0,2)	1,5 (0,2)	1,4 (0,2)	1,4 (0,2)	1,4 (0,2)	**1,3 (0,1)**	1,4 (0,3)	1,4 (0,2)
Instance 3								
#lower before prio 1	4,3 (0,5)	3,9 (0,0)	4,6 (0,2)	4,2 (0,0)	4,1 (0,2)	4,4 (0,6)	**3,8 (0,0)**	**3,8 (0,3)**
#lower before prio 2	1,2 (0,1)	1,2 (0,0)	1,3 (0,0)	1,1 (0,0)	0,9 (0,1)	0,7 (0,1)	0,7 (0,0)	**0,6 (0,2)**
St dev #tasks per route	3,2 (0,6)	3,0 (0,0)	3,9 (0,2)	3,4 (0,0)	3,2 (0,2)	3,5 (0,5)	**2,4 (0,0)**	3,2 (0,3)
Instance 4								
#lower before prio 1	1,0 (0,2)	1,1 (0,3)	1,4 (0,3)	1,9 (0,2)	0,5 (0,3)	**0,4 (0,1)**	0,8 (0,2)	0,7 (0,2)
#lower before prio 2	0,1 (0,1)	0,3 (0,2)	0,2 (0,1)	0,4 (0,2)	0,1 (0,2)	0,3 (0,1)	0,1 (0,1)	**0,0 (0,1)**
St dev #tasks per route	5,2 (0,2)	5,1 (0,2)	5,0 (0,3)	5,0 (0,2)	4,8 (0,2)	4,8 (0,2)	4,7 (0,3)	**4,6 (0,3)**

Table 10.2: Characteristics of the offline solutions: number of less important tasks before a task of some priority, and the standard deviation of the number of tasks per route. We report the average values and standard deviations (between brackets) over 10 runs of 15 minutes. For each instance, the best results are in boldface.

Online stage

For the online stage, we report both the obtained objective values in the simulations and the number of removed tasks, as a measure for the robustness of a solution, indicating its capabiility to stick to the original plan. The penalties for endurance and planning horizon were equal to zero in all experiments, thanks to the use of latest return times.

For the offline solutions constructed without using the risk function (i.e. for the first four approaches), we apply the standard rescheduling policy as described in Algorithm 7. Otherwise, we use the adjusted version with the risk check at each departure event, as described in Section 8.1.2.

When applying this risk rescheduling policy to offline solutions that were not built to satisfy these risk check, it is very likely that these solution are not even risk-feasible if the mean-case scenario would unfold in real-time. Therefore, it can be expected that a larger number of tasks is removed than for the basic rescheduling policy. This result can be observed in Tables B.1 and B.2 in Appendix B, which also shows that the online objective value always increases when incorporating the risk function in the rescheduling policy. Nevertheless, we choose to use the basic rescheduling policy for the approaches STD, REORDER, WCF90, and WCF99, since it does not make sense to start the online stage with a solution that is not even feasible in the mean-case scenario.

Tables B.1 and B.2 also show that the approach WCF99 always performs best if a worst-case scenario unfolds in real-time. The approach WCF90 is always the second best approach regarding the number of removed tasks, and for two instances also for the online objective value. However, since we assume a mean-case scenario in the offline stage when constructing the schedules, we focus on the online performance when travel and service times are simulated as described in Section 9.3, rather than on the worst-case scenario.

As a final remark for Tables B.1 and B.2, observe that the worst-case scenario does not necessarily yield the worst online objective value. In particular, for approaches WCF90 and WCF99, the average online objective value in the worst-case scenario sometimes exceeds the one in which we use simulations of travel and service times as described in Section 9.3.

Table 10.3 summarizes the results for the online stage. First of all, observe that improving the schedules of a solution by a final extension of depot visits appears to be very valuable. For all instances and approaches, the results significantly outperform the results for solutions without the final extension, both with respect to the objective values and to the number of removed tasks. When we zoom in on the reasons why tasks were removed, the results suggest that a final depot extension leads to a higher percentage of task removals due to feasibility problems with time windows. This makes sense, since an extension of a depot visit makes a schedule more compact, at the cost of decreased time window buffers.

	STD	REORDER	WCF90	WCF99	RISK	RISKRES	RWCF90	RRWCF90
Instance 1								
Offline obj	38405,2 (448,8)	38408,7 (452,3)	37122,9 (939,1)	33623,5 (1192,5)	38628,7 (435,9)	38945,8 (0,4)	37515,4 (619,9)	37828,3 (291,0)
Online obj	35210,2 (1413,6)	35617,5 (1407,9)	34522,7 (1411,2)	32867,7 (1336,9)	36244,6 (1173,7)	36279,4 (1221,7)	35852,8 (1119,1)	**36935,9 (785,7)**
Online obj no ext	34622,0 (1392,7)	34834,6 (1335,2)	34102,6 (1473,1)	32131,2 (1470,0)	35687,8 (1250,1)	36085,6 (1204,6)	35052,1 (1159,3)	36153,1 (952,5)
#tasks removed	4,2 (1,5)	3,8 (1,7)	3,9 (1,5)	2,5 (1,3)	3,7 (1,4)	3,8 (1,5)	3,3 (1,5)	**2,1 (1,2)**
#removed no ext	4,9 (1,5)	4,8 (1,7)	4,5 (1,6)	4,0 (1,4)	4,5 (1,4)	4,5 (1,5)	4,4 (1,5)	3,4 (1,3)
Instance 2								
Offline obj	37774,0 (16,0)	37781,3 (7,2)	37780,5 (6,2)	37753,6 (20,6)	37777,1 (7,1)	37764,6 (13,2)	37766.6 (11.6)	37757,3 (15,5)
Online obj	34222,5 (1559,8)	35216,5 (1303,7)	35278,0 (1432,0)	36700,7 (832,6)	36782,4 (855,8)	36495,1 (892,8)	36757,9 (903,4)	**37053,3 (722,4)**
Online obj no ext	33093,0 (1617,9)	34565,3 (1491,0)	34484,0 (1494,1)	35947,0 (1121,7)	36098,9 (972,4)	35820,5 (1078,8)	36086,5 (1181,6)	36307,4 (1056,2)
#tasks removed	7,5 (2,0)	7,2 (2,0)	7,4 (2,0)	**6,4 (1,9)**	7,0 (2,0)	7,1 (2,0)	7,0 (2,0)	6,8 (2,0)
#removed no ext	10,1 (2,2)	9,7 (2,1)	9,9 (2,2)	9,3 (2,1)	9,9 (2,2)	9,9 (2,2)	9,8 (2,1)	9,8 (2,3)
Instance 3								
Offline obj	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)	38807,0 (0,0)
Online obj	36487,3 (1456,9)	36804,6 (1115,9)	37703,5 (895,9)	38204,8 (669,8)	38560,3 (534,2)	38723,7 (229,6)	38754,7 (57,6)	**38768,9 (103,7)**
Online obj no ext	35155,1 (1772,3)	36061,3 (1219,4)	37085,6 (1020,0)	37291,9 (946,1)	37715,3 (896,2)	38482,9 (476,6)	38243,0 (627,2)	38048,0 (794,9)
#tasks removed	5,1 (1,8)	4,6 (1,5)	**3,9 (1,5)**	4,1 (1,6)	4,7 (1,8)	5,0 (1,7)	4,0 (1,5)	4,7 (1,7)
#removed no ext	8,0 (2,2)	5,8 (1,6)	7,1 (1,9)	6,9 (1,8)	7,6 (2,2)	7,8 (1,9)	7,9 (2,0)	7,7 (2,0)
Instance 4								
Offline obj	36987,8 (415,5)	36894,8 (644,4)	35001,3 (864,6)	31610,9 (459,5)	36738,1 (385,1)	37768,7 (269,2)	35604,4 (725,9)	35201,1 (369,0)
Online obj	33427,0 (1518,2)	33151,7 (1500,2)	32527,3 (1627,5)	30957,9 (854,4)	33955,6 (1341,7)	**34804,5 (1305,4)**	33990,5 (1125,8)	34266,1 (902,0)
Online obj no ext	32611,5 (1500,3)	32413,46 (1554,1)	31865,8 (1751,5)	30504,5 (944,9)	33275,0 (1325,4)	34265,7 (1349,8)	33189,3 (1262,0)	33559,4 (1040,5)
#tasks removed	4,1 (1,6)	4,2 (1,6)	3,7 (1,5)	**2,8 (1,3)**	3,5 (1,5)	3,9 (1,5)	3,1 (1,4)	2,8 (1,4)
#removed no ext	4,9 (1,6)	5,0 (1,6)	4,7 (1,5)	4,1 (1,4)	4,4 (1,5)	4,7 (1,5)	4,3 (1,5)	4,2 (1,5)

Table 10.3: Online results for Tabu Search: online objective value and the number of removed tasks, with and without a final extension of depot visits. For comparison purposes, we repeat the offline objective value. We report the averages and standard deviations (between brackets) over 1000 simulations per offline solution. For each instance, the highest online objective values and the lowest number of removed tasks are in boldface.

As all tasks can be scheduled in Instance 3, the robustness of a solution is the most interesting result. The effect of including the robustness concepts on the online performance can be observed very clearly for this instance. Requiring a worst-case fraction of $\underline{WCF} = 0.9$ already improves the basic online objective value significantly (compare approaches WCF90 and REORDER), while using the more strict requirement $\underline{WCF} = 0.99$ (approach WCF99) yields even higher objective values. Using the risk function (approaches RISK, RISKRES, RWCF90, RRWCF90) improves the online objective value even further.

The results suggest that using a worst-case fraction is especially useful when most of the tasks can be scheduled, as the approaches WCF90 and WCF99 perform quite poorly in Instances 1 and 4, while it significantly improves the results obtained by the basic approaches (STD and REORDER) in Instances 2 and 3. For the former instances, the offline objective value for WCF99 already drops below the obtained online objective value for the basic approaches. Although fewer and less important tasks have to be removed in real-time, the online objective value often is significantly smaller than the ones for the basic approaches in these instances.

In general, the risk function and the corresponding rescheduling policy prove their usefulness, as the corresponding approaches (RISK, RISKRES, RWCF90, RRWCF90) outperform all other approaches for all instances, regarding the online objective value. In particular, using the risk function leads to a considerable improvement of the online results compared to the basic approaches (STD and REORDER) regarding both online objective value and the number of removed tasks, regardless of the instance.

Combining the risk function with a worst-case fraction (approaches RWCF90 and RRWCF90) always leads to a decreased number of removed tasks compared to the plain risk approaches (RISK and RISKRES), while the effect on the online objective value varies per instance and approach. Considering the online performance over all instances, the approach RRWCF90 yields the best results, which suggests that combining the two concepts might actually pay off.

10.1.2 LRCG results

For the LRCG method, we investigate the effect of our adjustments to the subgradient algorithm (see Section 4.2.2). We are particularly interested in the strength of the obtained upper bound estimators for the optimal total task value and in the quality of the obtained feasible solutions.

For the subgradient algorithm, we used the following parameters related to the step size: $\alpha = 2.0, decrFactor = 0.5, p = 20, p' = 100, nrItsAvg = 4$, and $minImprove = 0.02$. Furthermore, we used $\mathbf{\Lambda}^0 = \mathbf{0}, \delta = 0.01, \varepsilon = 0.001$, and $j_{max} = 1500$.

For the pricing problem, recall the dominance rules from Section 4.3.1. As in

Breugem et al. [2015], we report the results when using rule 1 and a maximum running time of 30 seconds per pricing problem. We also report results when using rule 2 additionally, with a maximum running time of 60 seconds per pricing problem. We stress that both dominance rules are of a heuristic nature.

Upper bound estimators

Figure 10.1 shows the convergence of the Lagrangian Dual solution throughout the column generation algorithm for all instances. Instance 3 is particularly interesting, since we have seen in the Tabu Search results that all tasks can be scheduled. Therefore, the dual solution should ideally converge to 0. In Breugem et al. [2015], a value of -3.01 was achieved for this instance. Within 10 column generation iterations, a value of -0.003 has already been attained with the adjusted subgradient algorithm. Afterwards, the dual solution converges even more to 0.

Table 10.4 shows the obtained Lagrangian Dual solutions $\hat{\Theta}_N$ in the final iteration N of the LRCG algorithm. As we discussed in Section 4.4, $\hat{\Theta}_N$ can be used as a lower bound estimator (UBE) of the objective value of the ILP in Section 4.1, while the converse $(\sum_{i \in \mathcal{I}} v_i) - \hat{\Theta}_N$ can be used an upper bound estimator for the optimal total task value.

For dominance rule 1, the new UBEs are stronger bounds than the ones in Breugem et al. [2015] for Instances 1 and 4, while they are comparable for Instances 2 and 3. Comparing these UBEs to the maximum values we encountered for these instances when applying Tabu Search, we observe that we tend to underestimate the maximum possible objective value. This is due to the heuristic nature of dominance rule 1, which leads to an early termination of the LRCG algorithm even though there are still favourable columns remaining.

For dominance rule 2, the UBEs become stronger, since rule 2 is less strict than rule 1 in the sense that more partial paths remain non-dominated. In particular, the UBEs now exceed the maximum values obtained by Tabu Search. The UBEs might still underestimate the optimal total task value since dominance rule 2 is a heuristic rule as well. However, an overestimation is possible as well now, since the Lagrangian Dual problem is a relaxation of the original problem and its solution is approximated by the subgradient algorithm.

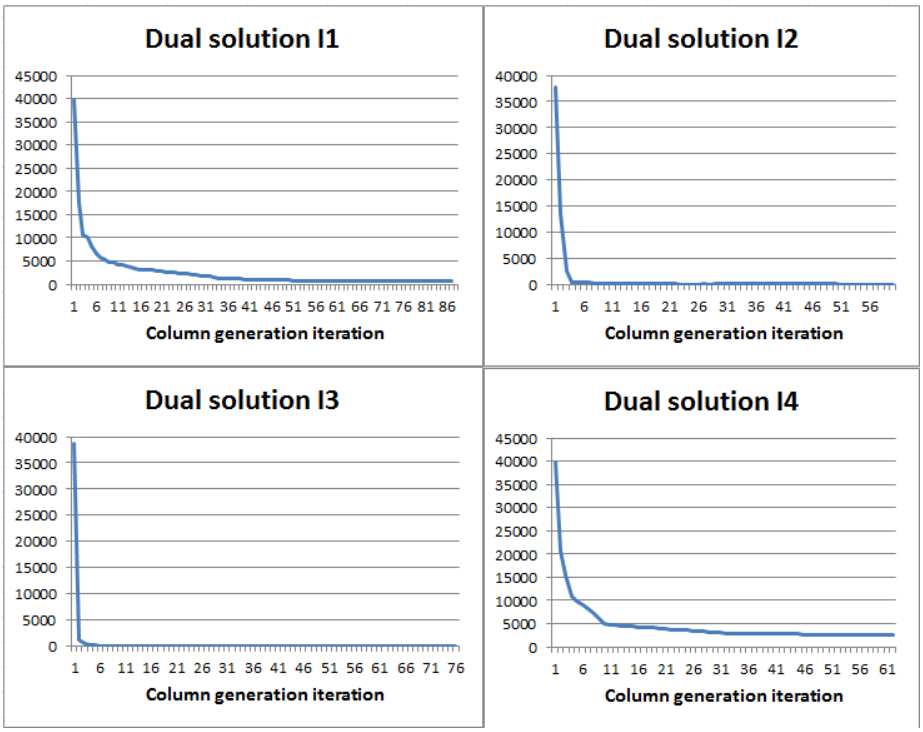

Figure 10.1: The development of the Lagrangian Dual solutions.

		Instance 1	Instance 2	Instance 3	Instance 4
Rule 1	Obj Breugem et al. [2015]	33826	34457	37640	26918
	Feasible obj value	36737	37454	38716	35777
	UBE Breugem et al. [2015]	38829	37789	38807	36293
	UBE	38931	37784	38807	37142
Rule 2	Feasible obj value	38753	37584	38765	35841
	UBE	39012	37793	38807	38550
	Highest obtained obj TS	39008	37786	38807	37923
	Maximal objective	39679	37799	38807	39679

Table 10.4: LRCG results: objective values of feasible solutions and upper bound estimators (UBEs) on the optimal total task value. The UBEs correspond to the Lagrangian Dual solutions from the final LRCG iteration. For comparison, we also show the UBEs and objective values of feasible solutions obtained in Breugem et al. [2015] with rule 1, the highest obtained objective value by the Tabu Search (TS) heuristic, and the maximal possible objective value if all tasks could be scheduled. We use a maximum running time of 30 / 60 seconds for each pricing problem when using dominance rule 1 / 2, respectively.

Feasible solutions

Table 10.4 also shows the objective values of the best feasible solutions obtained by the greedy procedure in Section 4.5. For dominance rule 1, the obtained feasible solutions significantly improve the ones in Breugem et al. [2015] for all instances, especially for Instance 4. Due to the adjusted subgradient algorithm, the Lagrangian Dual problem results in different Lagrangian multipliers $\mathbf{\Lambda}$, which in turn leads to different generated columns in the pricing problem. Thus, in addition to an improvement of the upper bound estimators, the adjusted subgradient algorithm leads to better feasible solutions as well.

For dominance rule 2, the quality of the feasible solutions is improved further for all instances. Due to using a less strict dominance rule and a higher maximal running time per pricing problem, we are better able to find the shortest path in the pricing problem. Thus, we generate better columns and obtain a feasible solution with a higher objective value. However, the LRCG approach is still outperformed by the Tabu Search heuristic for all instances, although the differences in objective values are typically relatively small now.

10.2 Results for the extended problem

For the extended problem, we use the datasets with resource-dependent values and recurrent tasks as described in Section 9.2. The evaluation criteria basically remain the same for both stages. An interesting new aspect is the assignment of recurrent tasks. Due to the non-linear convex profit function, we would expect that recurrent sets are often performed either completely or not at all, especially in the offline stage. Furthermore, with the more general structure of task values, it is interesting to observe the behaviour of the risk function under different delay functions.

For the Extended Security Routing problem, we can only use the risk function in the *restore* variant, since the value structure in a route might change due to moves involving recurrent tasks for other routes, as shown in Section 7.2. We distinguish two piecewise linear delay functions, with cutoff ratios of 3 and 6. In addition to the used approaches REORDER, WCF90, and WCF99, we use the following approaches:

- NOADDRECS: Similar to REORDER, but without adding recurrent tasks in the perturbation procedure.
- RISKRES6: Similar to RISKRES in the original problem, given a delay function $g(R) = \min\{(R-1)/5, 1\}$.
- RISKRES3: Similar to RISKRES6, but with $g(R) = \min\{(R-1)/5, 1\}$.
- RR3WCF90: Similar to RISKRES3, but accompanied by $\underline{WCF} = 0.9$.

For RISKRES3, the delay function is steeper than for RISKRES6. Thus, we probably obtain more conservative solutions with RISKRES3, as a steeper delay function leads to more severe delays in the risk checks.

Offline stage

The results for the offline stage are summarized in Table 10.5. We observe that recurrent sets are indeed very often either scheduled completely or not at all. Most exceptions occur for approach WCF99. This might be partially due to a low number of reorder and perturbation procedures for this approach, due to a lack of running time.

Regarding the robustness concepts, we observe that the worst-case fraction often leads to a better distribution of tasks over the routes, but also to a significantly lower objective value, especially for approach WCF99. Using the risk function only leads to a relatively small decrease in the offline objective values. When we compare approaches RISKRES6 and RISKRES3, we observe that a steeper delay function (with cutoff ratio 3) typically leads to a lower offline objective value, due to the more strict requirements in the risk checks.

Table 10.5 also shows that the number of iterations for Instance 9 is very low. This is due to the characteristics of the instance that lead to routes of up to 60 tasks and 6 recharging depots for a single resource, which makes it very time-intensive to determine alternative routes, especially in a reorder procedure. Recall that all tasks in this instance are recurrent and observe that almost all tasks are scheduled for most approaches.

For Instance 7, almost all recurrent tasks are scheduled for all approaches. For the other instances, the solutions typically include the recurrent sets for original priority 1. If sets of lower priority are included, these sets often have the smallest size 3. In Instance 8, however, the recurrent set of size 3 of original priority 1 was not scheduled for any approach, while a less important recurrent set of size 7 for original priority 2 was often scheduled. This can be explained by the topography of the instance, which shows that the small set of priority 1 is far from both depots, while the large set of priority 2 is relatively close to both depots. In particular, the average distances to a depot are given by 52.2 and 18.1, respectively.

The results suggest that incorporating the step to add recurrent tasks in the perturbation procedure does not necessarily increase the percentage of scheduled recurrent tasks or the objective value. Typically, the results for NOADDRECS are very close to the ones for REORDER. This means that the common scheduled recurrent tasks find their way into the solution, regardless of any help in the perturbation procedure. For the recurrent tasks that are left unscheduled by REORDER, it might just not be worthwhile to include them in the solution. In Instance 6, however, the recurrent set of size 7 with original priority 2 is included in the solution more often for NOADDRECS.

	NOADDRECS	REORDER	WCF90	WCF99	RISKRES6	RISKRES3	RR3WCF90
Instance 5							
Offline obj	707,2 (6,0)	**709,2 (4,4)**	637,0 (17,3)	521,6 (5,0)	697,0 (4,8)	679,6 (11,4)	623,7 (22,7)
#tasks	53,9 (0,8)	54,2 (0,8)	53,4 (1,6)	49,8 (0,6)	51,0 (1,0)	49,1 (1,4)	47,3 (1,1)
%rec tasks	28,4 (2,8)	28,4 (-)	21,6 (-)	17,3 (-)	30,0 (5,1)	26,2 (-)	15,7 (-)
%rec sets all/0	98,9 (3,3)	97,8 (-)	94,4 (-)	46,7 (-)	**100,0 (0,0)**	**100,0 (0,0)**	97,8 (-)
St dev #tasks	5,7 (0,3)	5,6 (0,1)	5,4 (0,3)	5,1 (0,0)	5,2 (0,2)	5,2 (0,3)	**4,9 (0,2)**
#iterations	62504,4 (1047,1)	62490,4 (1970,8)	10110,3 (1315,3)	1320,0 (72,0)	34719,1 (1643,5)	26010,1 (889,1)	4176,8 (443,0)
Instance 6							
Offline obj	**936,5 (5,1)**	932,9 (2,8)	832,3 (0,0)	637,0 (0,0)	919,6 (14,8)	905,9 (3,5)	783,0 (0,0)
#tasks	86,0 (0,9)	85,2 (1,0)	80,3 (0,9)	59,0 (0,0)	82,3 (1,1)	80,4 (0,7)	74,0 (0,0)
%rec tasks	52,9 (6,3)	46,0 (-)	37,3 (-)	17,8 (0,0)	37,3 (-)	39,3 (-)	13,3 (0,0)
%rec sets all/0	95,6 (7,3)	93,3 (-)	67,7 (-)	44,4 (0,0)	92,2 (-)	**100,0 (0,0)**	**100,0 (0,0)**
St dev #tasks	1,1 (0,1)	1,0 (0,2)	1,0 (0,1)	1,9 (0,0)	1,2 (0,2)	1,2 (0,2)	**0,9 (0,0)**
#iterations	17070,4 (1220,2)	15609,5 (1647,2)	2258,3 (52,7)	602,0 (0,0)	15471,7 (765,2)	11059,7 (436,7)	1405,0 (0,0)
Instance 7							
Offline obj	1189,2 (1,2)	**1190,5 (2,3)**	1176,3 (2,5)	655,2 (0,0)	1173,4 (2,8)	1168,7 (1,1)	1082,3 (17,6)
#tasks	123,8 (1,4)	124,4 (1,4)	122,5 (0,5)	56,0 (0,0)	116,0 (1,1)	113,8 (0,4)	106,2 (1,0)
%rec tasks	84,9 (1,3)	85,1 (1,4)	84,4 (0,0)	4,4 (0,0)	72,0 (2,7)	73,3 (0,0)	49,6 (-)
%rec sets all/0	98,9 (3,3)	97,8 (4,4)	**100,0 (0,0)**	77,8 (0,0)	**100,0 (0,0)**	**100,0 (0,0)**	88,9 (-)
St dev #tasks	3,9 (0,2)	3,8 (0,3)	4,1 (0,1)	**1,9 (0,0)**	3,8 (0,2)	3,4 (0,1)	3,0 (0,4)
#iterations	13708,6 (1216,0)	14224,2 (1122,5)	3287,9 (53,8)	166,0 (0,0)	3269,9 (94,7)	1886,1 (83,5)	638,8 (98,4)
Instance 8							
Offline obj	**747,5 (5,5)**	747,4 (6,8)	681,2 (27,4)	607,3 (2,0)	735,5 (1,9)	731,3 (5,9)	706,1 (9,0)
#tasks	56,9 (0,7)	56,9 (0,7)	55,0 (2,1)	52,5 (0,5)	52,3 (0,8)	51,6 (0,7)	51,0 (1,2)
%rec tasks	43,8 (1,0)	44,0 (0,9)	32,0 (3,9)	40,9 (1,8)	31,3 (7,1)	26,7 (0,0)	28,2 (1,7)
%rec sets all/0	92,2 (5,1)	91,1 (4,4)	81,1 (12,2)	40,0 (7,4)	**100,0 (0,0)**	**100,0 (0,0)**	92,2 (8,7)
St dev #tasks	5,3 (0,3)	5,3 (0,2)	5,3 (0,3)	5,0 (0,1)	4,7 (0,1)	**4,5 (0,1)**	4,9 (0,2)
#iterations	60367,6 (1290,1)	59827,6 (1475,5)	10669,1 (1524,4)	1195,0 (0,0)	30953,7 (884,8)	25338,1 (481,9)	5579,2 (430,3)
Instance 9							
Offline obj	**124,2 (1,5)**	123,7 (1,5)	121,8 (1,0)	94,3 (1,5)	122,4 (2,7)	122,9 (0,9)	104,7 (6,4)
#tasks	125,0 (0,9)	124,6 (0,9)	122,7 (1,1)	99,3 (1,4)	122,8 (2,1)	123,5 (1,5)	106,9 (7,2)
%rec sets all/0	97,9 (1,7)	97,5 (1,6)	97,1 (1,4)	84,6 (2,8)	**98,9 (2,3)**	98,2 (1,8)	93,6 (4,2)
St dev #tasks	16,8 (0,6)	15,2 (2,0)	18,5 (2,8)	**10,9 (0,3)**	13,6 (3,8)	12,3 (2,6)	12,4 (3,2)
#iterations	1279,0 (232,0)	1348,9 (288,8)	839,3 (85,5)	362,0 (0,0)	939,8 (101,0)	1070,2 (197,0)	864,8 (63,1)

Table 10.5: Offline results for Tabu Search: objective value, number of scheduled tasks, percentage of recurrent tasks scheduled, percentage of recurrent sets either fully scheduled or not at all, standard deviation of number of tasks per route, and number of iterations. We report the averages and standard deviations (between brackets, if available) over 10 runs of 15 minutes. For each instance, the highest offline objective value, the highest percentage of full or empty recurrent sets, and the most even distribution of tasks are in boldface.

Online stage

For the online stage of the Extended Security Routing problem, we only consider simulations after a final extension of depot visits. As before, the criteria for online performance are the online objective value and the number of removed tasks. The results are summarized in Table 10.6.

This table shows that the online solutions for WCF99 are close to their original offline solutions again. However, in all instances, the offline objective values already drop significantly below the online objective values of all other approaches. The less strict approach WCF90 performs better: it yields the best online performance for both criteria in the Divi instance, while it yields the highest online objective value for Instance 7.

As for the original Security Routing problem, using the risk function (approaches RISKRES6 and RISKRES3) is the best way to obtain online solutions of high quality. Although the offline objective values for these approaches are slightly smaller than for the basic approach REORDER, the online objective values are typically larger, while the number of removed tasks is always lower.

Whereas the offline objective value was almost always higher for RISKRES6 than for the more strict RISKRES3, the comparison for the online results strongly depends per instance. Thus, the more strict risk requirements sometimes turn the lower offline objective value into a better real-time performance, which might be caused by a better protection of important tasks in real-time.

For these instances, combining the risk function and the worst-case fraction is not very successful: it leads to considerable deterioriations of the online objective value compared to the approaches RISKRES6 and RISKRES3, although the number of removed tasks decreases as well. Furthermore, it only improves the online performance compared to WCF90 for Instances 5 and 8.

	NOADDRECS	REORDER	WCF90	WCF99	RISKRES6	RISKRES3	RR3WCF90
Instance 5							
Offline obj	692,4 (12,2)	709,2 (4,4)	637,0 (17,3)	521,6 (5,0)	697,0 (4,8)	679,6 (11,4)	623,7 (22,7)
Online obj	615,6 (34,4)	626,8 (33,5)	606,5 (24,9)	512,8 (11,1)	**634,3 (33,0)**	626,7 (33,7)	609,3 (27,9)
#tasks removed	5,0 (1,6)	5,6 (1,6)	3,1 (1,5)	1,8 (1,0)	3,3 (1,6)	2,9 (1,4)	**1,4 (1,1)**
Instance 6							
Offline obj	936,5 (5,1)	932,9 (2,8)	832,3 (0,0)	637,0 (0,0)	919,6 (14,8)	905,9 (3,5)	783,0 (0,0)
Online obj	809,0 (35,7)	819,8 (32,9)	777,7 (22,9)	611,9 (16,6)	823,0 (36,1)	**825,3 (33,1)**	745,7 (13,1)
#tasks removed	10,0 (2,3)	9,4 (2,3)	6,5 (2,0)	**2,6 (1,2)**	7,4 (2,1)	6,5 (2,1)	5,2 (1,6)
Instance 7							
Offline obj	1189,2 (1,2)	1190,5 (2,3)	1176,3 (2,5)	655,2 (0,0)	1173,4 (2,8)	1168,7 (1,1)	1082,3 (17,6)
Online obj	1097,4 (35,8)	1102,6 (33,7)	1123,2 (25,6)	651,7 (3,9)	1101,9 (36,5)	**1122,4 (24,1)**	1059,3 (21,0)
#tasks removed	8,7 (2,2)	9,0 (2,2)	7,0 (1,9)	**1,1 (0,7)**	5,0 (1,9)	4,6 (1,7)	3,9 (1,5)
Instance 8							
Offline obj	747,5 (5,5)	747,4 (6,8)	681,2 (27,4)	607,3 (2,0)	735,5 (1,9)	731,3 (5,9)	706,1 (9,0)
Online obj	673,1 (39,3)	680,4 (35,3)	649,3 (32,7)	590,65 (18,3)	**735,5 (1,9)**	677,4 (32,3)	690,4 (15,8)
#tasks removed	4,6 (1,7)	4,2 (1,6)	3,3 (1,3)	**1,7 (1,0)**	3,5 (1,5)	3,4 (1,5)	1,8 (1,1)
Instance 9							
Offline obj	124,2 (1,5)	123,7 (1,5)	121,8 (1,0)	94,3 (1,5)	122.4 (2.7)	122,9 (0,9)	104.7 (6.4)
Online obj	116,0 (4,4)	116,1 (4,8)	119,4 (2,8)	93,4 (2,1)	115,8 (4,5)	**118,1 (3,6)**	102,0 (6,7)
#tasks removed	3,9 (2,3)	3,8 (2,5)	1,1 (1,2)	**0,7 (0,9)**	3,0 (2,2)	2,3 (1,7)	1,4 (1,2)

Table 10.6: Online results for Tabu Search: offline objective value (for comparison), online objective value, and number of removed tasks. We report the averages and standard deviations (between brackets) over 1000 simulations for each offline solution. For each instance, the highest online objective value and the lowest number of removed tasks are in boldface.

Chapter 11

Conclusions and further research

In this final chapter, we first summarize the thesis. Afterwards, we point to some directions for further research.

11.1 Conclusions

This thesis addresses the Security Routing problem, which is an extensive version of the Team Orienteering Problem, including multi-trips, endurance constraints, and heterogeneous resources. In the Extended Security Routing problem, resource-dependent task values and recurrent tasks with non-linear profits were added. Moreover, due to stochastic travel and service times, we have to deal with an offline stage of constructing initial (robust) solutions and an online stage of adjusting the schedules in real-time.

For the original Security Routing problem, we proposed a heuristic based on the combination of Lagrangian Relaxation and Column Generation, and a Tabu Search heuristic that contains a procedure of reordering tasks within routes. The Tabu Search heuristic is suitable for the Extended Security Routing problem as well, and generates schedules that have a low probability of violating the constraints on endurance and the planning horizon, by the use of latest return times.

We introduced the robustness concepts of the risk function and the worst-case fraction. We incorporated these concepts in our Tabu Search heuristic, in order to construct robust solutions in the offline stage. Moreover, the robustness of individual schedules was improved by extending depot visits. We generated solutions for the offline stage with and without incorporating the robustness

concepts in the Tabu Search heuristic. We used simulations of travel and service times in order to measure the robustness of the solutions.

For the online stage, we proposed rescheduling policies that check whether it is responsible to head for the upcoming task, regarding the current delay and the values of the subsequent tasks. In addition to its contributions to a robust solution for the offline stage, the introduced concept of the risk function can be incorporated in the rescheduling policy in a natural way.

For the Extended Security Routing problem, we discussed a procedure to handle all routes simultaneously in real-time, which is necessary because of the non-linear profits for recurrent tasks. The quality of realized online solutions is evaluated by the obtained objective value and the number of removed tasks, which indicates how close the realized solution was to the initial solution.

We applied the LRCG heuristic to the original Security Routing problem. The results suggest that our adjustments to the subgradient algorithm are successful. The upper bound estimators for the optimal total task value are significantly stronger than the ones in Breugem et al. [2015] for two instances, and comparable for the two other instances. Moreover, the quality of the generated feasible solutions was improved significantly for all instances. Using a less strict dominance rule leads to further improvements of the results.

When applying the Tabu Search heuristic to original Security Routing problem, the results for the offline stage suggest that including the reorder procedure in the Tabu Search heuristic does not improve the objective value, although the reorder procedure is applied successfully around $10\% - 20\%$ of the times. For the risk function, the desirable side-effect to prefer routes with decreasing values is clearly visible, as opposed to the desired side-effect of a more even distribution of tasks over resources when using the worst-case fraction.

The online results show that the procedure to extend depot visits is a successful and important part of the construction of robust solutions. It appears that using the worst-case fraction is especially useful for instances in which most of the tasks can be scheduled. Regardless of the instance, using the risk function leads to a significant improvement of the online results compared to the basic heuristic, in both online objective value and the closeness to the original plan. Furthermore, combining the two concepts appears to be a promising option.

For the Extended Security Routing problem, the offline results show that recurrent sets are very often either completely incorporated in the offline schedule or not at all, which is a natural behaviour considering the non-linear convex profits of recurrent sets. The incorporated step in the perturbation procedure to add recurrent tasks, in order to help them appear in the solution, does not seem to contribute much to the offline objective value. As for the original problem, using a strict requirement on the worst-case fraction appears to be questionable, since this can lead to a large decrease of the offline objective value.

For the online stage, using the risk function again proved to be the best way to obtain stable and high quality online results. Using more strict risk requirements, modeled by a steeper delay function, leads to a lower number of removed tasks for all instances, whereas the effect on the online objective value varies per instance. Contrary to the original problem, the results obtained by using the risk function are never improved when incorporating the worst-case fraction as well.

11.2 Further research

Several different objective criteria might be included in the Tabu Search, in addition to maximizing the task values. For instance, the worst-case fraction might contribute to a more even distribution of tasks over the resources, but within the subset of solutions that satisfy the requirement implied by the worst-case fraction, there is currently no incentive to prefer solutions in which tasks are distributed more evenly. If the problem would be extended further by including the danger of performing a task, a corresponding second objective criterion would allow the user to choose between risky or risk-averse solutions. Furthermore, some measure of free space in a solution might be designed and used as a second objective criterion, possibly combined with including the reordering procedure as a local move in the Tabu Search algorithm.

In the current definition of the latest return times (see Section 5.2.1), we assume worst-case travel times to the depot, which successfully avoids endurance and planning horizon violations. It might be interesting to consider a less strict travel time that is based on the value of the task we are performing, in order to balance the risk of violating endurance or planning horizon constraints and the cost of excluding or aborting an important task. Similarly, we could apply more flexible rules regarding heading for the upcoming task in real-time. Currently, we skip the upcoming task when the probability of arriving in time is below 50%. A more flexible and value-dependent threshold might be suitable here as well.

An active real-time policy could be implemented to improve the online results, especially when the objective value in real-time is considered to be more important than sticking to the original plan. A suggestion for such a policy was made in Section 8.2. Furthermore, in the case that the first task after a depot visit is removed, a decision to stay longer at the depot than initially planned might improve the current rescheduling policies.

In order to improve the robustness of a given schedule, we applied a final depot visit extension that balances the size of buffers with respect to time windows and endurance buffers. However, it might be worthwhile to investigate a weighted procedure that takes task values and the position of the task in the route into account, as delays can become larger in a later stage of the route. Furthermore,

in a very tight trip with respect to endurance, we might want to relax the upper bound of the total waiting time in the trip on the extension size.

Currently, the feasible solutions generated by the LRCG heuristic are outperformed by the ones generated in the Tabu Search heuristic. A common issue for LRCG approaches is that the generated routes have a high value for the resources individually, but share a relatively large common set of tasks, which only need to be performed at most once. In order to circumvent this problem, *column fixing* could be used (see e.g. Caprara et al. [1999], Potthoff et al. [2010]). The main idea of this technique is to repeatedly fix different parts of the solution and to generate columns that complement the fixed part well.

Appendix A

Terminology and variables

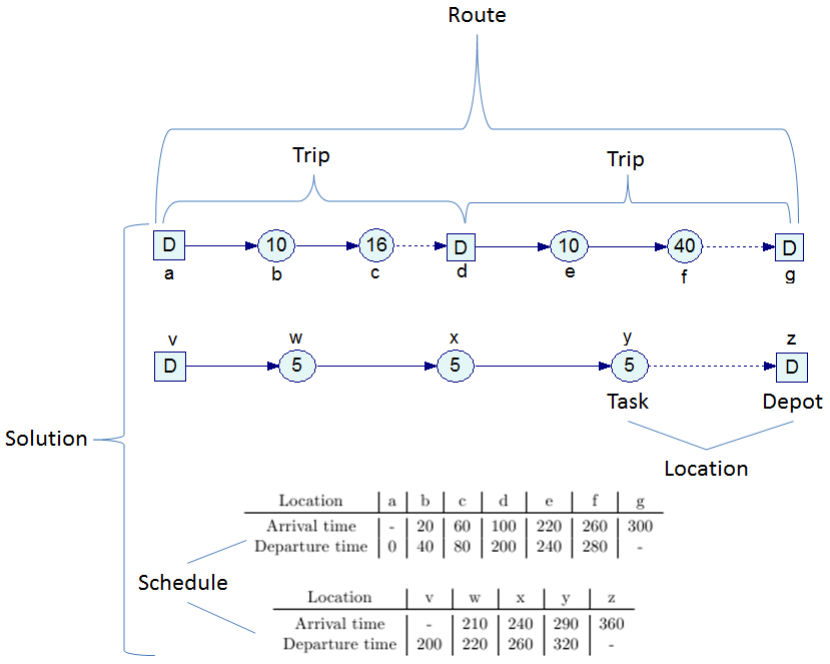

Figure A.1: A solution that consists of two routes and their schedules. Route 1 consists of two trips, while route 2 consists of a single trip. The locations are represented by squares (depots) and circles (tasks with their values).

Variable	Section	Description
$\mathcal{I}$	2.1	Set of tasks
$\mathcal{R}$	2.1	Set of resources
L_i	2.1	Release time of task i
U_i	2.1	Deadline of task i
v_i	2.1	Base value for task i
v_{ir}	2.2	Value for task i, when performed by resource r
E_r	2.1	Endurance of resource r
R_r	2.1	Minimal rest time for resource r
T	2.1	End of the planning horizon
$\bar{s}_{ir}$	2.1	Expected service time of task i by resource r
$\bar{t}_{ijr}$	2.1	Expected travel time from location i to location j for resource r
σ_{TT}	2.1	Maximal deviation fraction for travel times
σ_{ST}	2.1	Maximal deviation fraction for service times
K_r	4.1	Set of all feasible routes for resource r
K_r^n	4.2	Set of generated routes for resource r in the n-th LRCG iteration
K^n	4.2	Set of all generated routes in the n-th LRCG iteration
$\mathbf{\Lambda} = (\lambda_i)_{i \in \mathcal{I}}$	4.2	Lagrangian multipliers
$\zeta_{kr}(\mathbf{\Lambda})$	4.2	Reduced costs for route k for resource r, given Lagrangian multipliers $\mathbf{\Lambda}$
$\Theta_n(\mathbf{\Lambda})$	4.2	Optimal objective value for the Lagrangian subproblem in the n-th LRCG iteration, given Lagrangian multipliers $\mathbf{\Lambda}$
Θ_n^*	4.2	Optimal objective value for the Lagrangian Dual problem in the n-th LRCG iteration
$\hat{\Theta}_n$	4.2	Approximation of Θ_n^*, obtained by the subgradient algorithm.
arr_j	4.3, 5.2	Arrival time at location j
$wait_j$	4.3, 5.2	Waiting time before location j
$start_j$	4.3, 5.2	Starting time at location j
dep_j	5.2	Departure time from location j
LRT_i^{HOR}	5.2	Latest return time from task i, w.r.t. the planning horizon
LRT_i^{END}	5.2	Latest return time from task i, w.r.t. endurance
ms_i^{TW}	5.2	Maximum shift of the starting time of i, w.r.t. time windows
ms_i^{END}	5.2	Maximum shift of the starting time of i, w.r.t. endurance
$g : [1, \infty) \to (0, 1]$	6.2	Delay function that maps value ratios to delay fractions
$f : L \to [1, \infty)$	6.2	Risk function that maps locations to value ratios
$\underline{WCF}$	6.3	Lower bound on the worst-case fraction of a solution

Table A.1: For a selection of variables, we state the notation, the section in which they were introduced, and a description.

Appendix B

Additional results

	REORDER	WCF90	WCF99	RISK
Instance 1				
Offline obj	38637.3 (415.5)	37336.3 (437.5)	35106.7 (51.0)	38945.3 (0.9)
Online obj basic policy	34820.7 (1436.2)	35733.0 (1113.7)	34527.8 (676.5)	36436.2 (1159.2)
Online obj risk policy	35599.6 (1273.4)	36329.4 (1013.4)	34912.2 (306.5)	36397.7 (1185.7)
Online obj worst-case	27330.0 (459.8)	34208.7 (455.1)	34859.7 (30.2)	31552.7 (453.1)
#tasks removed basic	4.6 (1.5)	3.1 (1.4)	2.6 (1.2)	4.3 (1.5)
#tasks removed risk	5.9 (1.5)	4.2 (1.4)	3.6 (1.3)	3.9 (1.4)
#tasks removed WC	14.0 (0.8)	10.0 (1.6)	9.3 (0.5)	11.7 (0.5)
Instance 2				
Offline obj	37775.0 (10.6)	37764.7 (23.1)	37699.3 (13.8)	37778.3 (0.5)
Online obj basic policy	33638.9 (1613.8)	35610.2 (1070.1)	35947.9 (997.6)	37283.6 (650.7)
Online obj risk policy	35735.2 (1190.7)	36868.9 (803.4)	37198.1 (577.4)	37379.1 (541.2)
Online obj worst-case	26257.3 (1764.6)	34230.3 (20.0)	37345.7 (12.6)	34584.7 (1956.1)
#tasks removed basic	7.7 (2.0)	6.9 (1.9)	6.2 (1.8)	5.7 (1.8)
#tasks removed risk	13.6 (2.3)	9.2 (2.0)	9.1 (1.9)	6.0 (1.8)
#tasks removed WC	24.3 (0.9)	19.3 (1.3)	17.7 (1.3)	21.0 (0.0)

Table B.1: Online results for a selection of approaches of the Tabu Search heuristic for Instances 1 and 2. The offline objective value is reported for comparison. We report the online results when using the basic rescheduling policy, when incorporating the risk function in the policy, and for the worst-case scenario. For the former two policies, we report the averages and standard deviations (between brackets) over 1000 simulations for each of the 3 offline solutions, generated in at most 10 minutes. For the worst-case results, we report the averages and standard deviations (between brackets) over the online performance in the worst-case scenario over the 3 offline solutions. For the worst-case results, we use the basic rescheduling policy, except for approach RISK.

	REORDER	WCF90	WCF99	RISK
Instance 3				
Offline obj	38807.0 (0.0)	38807.0 (0.0)	38807.0 (0.0)	38807.0 (0.0)
Online obj basic policy	36850.2 (1157.0)	37705.1 (907.3)	38232.8 (583.7)	38340.4 (707.4)
Online obj risk policy	38471.8 (524.9)	38703.3 (228.1)	38726.7 (128.1)	38475.9 (622.5)
Online obj worst-case	28304.3 (553.4)	34990.0 (0.0)	38472.3 (13.2)	36574.0 (2010.6)
#tasks removed basic	4.8 (1.6)	4.6 (1.5)	4.4 (1.6)	3.8 (1.6)
#tasks removed risk	8.3 (1.8)	6.5 (1.6)	7.1 (1.7)	4.9 (1.8)
#tasks removed WC	23.0 (0.0)	22.0 (0.0)	19.7 (0.5)	22.7 (1.3)
Instance 4				
Offline obj	37313.7 (409.0)	33476.7 (452.8)	31171.0 (836.1)	37560.0 (422.9)
Online obj basic policy	33978.8 (1418.4)	32128.0 (1143.3)	30235.4 (1110.5)	34876.3 (1413.4)
Online obj risk policy	34766.8 (1214.2)	32639.4 (898.3)	30906.3 (932.9)	34923.3 (1357.7)
Online obj worst-case	26006.3 (761.8)	30348.0 (467.4)	30934.0 (823.1)	26914.0 (1359.1)
#tasks removed basic	4.2 (1.5)	3.1 (1.6)	2.7 (1.3)	3.3 (1.4)
#tasks removed risk	5.5 (1.6)	4.8 (2.0)	4.0 (1.6)	3.6 (1.5)
#tasks removed WC	14.0 (1.4)	11.0 (1.4)	9.7 (1.7)	13.3 (1.9)

Table B.2: Online results for a selection of approaches of the Tabu Search heuristic for Instances 3 and 4. The offline objective value is reported for comparison. We report the online results when using the basic rescheduling policy, when incorporating the risk function in the policy, and for the worst-case scenario. For the former two policies, we report the averages and standard deviations (between brackets) over 1000 simulations for each of the 3 offline solutions, generated in at most 10 minutes. For the worst-case results, we report the averages and standard deviations (between brackets) over the online performance in the worst-case scenario over the 3 offline solutions. For the worst-case results, we use the basic rescheduling policy, except for approach RISK.

Bibliography

E. J. W. Abbink, L. Albino, T. A. B. Dollevoet, D. Huisman, J. Roussado, and R. L. Saldanha. Solving Large Scale Crew Scheduling Problems in Practice. *Public Transport*, 3(2):149–164, 2011.

C. Archetti, M. G. Speranza, and D. Vigo. Vehicle Routing Problems with Profits. *Vehicle Routing: Problems, Methods, and Applications*, 18:273, 2014.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.

N. Bianchessi, J.-F. Cordeau, J. Desrosiers, G. Laporte, and V. Raymond. A heuristic for the multi-satellite, multi-orbit and multi-user management of Earth observation satellites. *European Journal of Operational Research*, 177 (2):750–762, 2007.

J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Science & Business Media, 2011.

T. Breugem, G. van Ginkel, M. L. van Meerkerk, and R. Sewnarain. Security Routing with Uncertainty. Report for Seminar Logistic Case Studies, Erasmus University Rotterdam, 2015.

J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan. Rich Vehicle Routing Problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2): 32, 2014.

A. Caprara, M. Fischetti, and P. Toth. A Heuristic Method for the Set Covering Problem. *Operations Research*, 47(5):730–743, 1999.

I.-M. Chao, B. L. Golden, and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

J.-F. Cordeau and G. Laporte. Maximizing the Value of an Earth Observation Satellite Orbit. *Journal of the Operational Research Society*, 56(8):962–968, 2005.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column Generation*, volume 5. Springer Science & Business Media, 2006.

G. Erdogan and G. Laporte. The Orienteering Problem with Variable Profits. *Networks*, 61(2):104–116, 2013.

L. Evers, K. Glorie, S. van der Ster, A. Barros, and H. Monsuur. The Orienteering Problem under Uncertainty Stochastic Programming and Robust Optimization compared. Technical report, Econometric Institute Research Papers, 2012.

L. Evers, T. A. B. Dollevoet, A. I. Barros, and H. Monsuur. Robust UAV Mission Planning. *Annals of Operations Research*, 222(1):293–315, 2014a.

L. Evers, K. Glorie, S. Van Der Ster, A. I. Barros, and H. Monsuur. A two-stage approach to the orienteering problem with stochastic weights. *Computers & Operations Research*, 43:248–260, 2014b.

M. L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, 1981.

E. F. Flushing, L. M. Gambardella, and G. A. Di Caro. Collaborative Missions with Heterogeneous Teams: Mathematical Model and Solution Approach. *IDSIA, Lugano (Switzerland), Tech. Rep*, pages 02–14, 2014.

F. Glover and M. Laguna. *Tabu Search*. Springer, 2013.

B. L. Golden, L. Levy, and R. Vohra. The Orienteering Problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

I. E. Grossmann. Review of Nonlinear Mixed-Integer and Disjunctive Programming Techniques. *Optimization and Engineering*, 3(3):227–252, 2002.

D. Huisman, R. Jans, M. Peeters, and A. P. M. Wagelmans. *Combining Column Generation and Lagrangian Relaxation*. Springer, 2005.

T. İlhan, S. M. R. Iravani, and M. S. Daskin. The Orienteering Problem with Stochastic Profits. *Iie Transactions*, 40(4):406–421, 2008.

L. Ke, Z. Xu, Z. Feng, K. Shang, and X. Qian. Proportion-based robust optimization and team orienteering problem with interval data. *European Journal of Operational Research*, 226(1):19–31, 2013.

L. Kroon and M. Fischetti. Crew Scheduling for Netherlands Railways - Destination: Customer. In *Computer-aided scheduling of public transport*, pages 181–201. Springer, 2001.

E. J. Kuipers. An algorithm for selecting and timetabling requests for an Earth Observation Satellite. *Bulletin de la Société Française de Recherche Opérationnelle et dAide à la Décision*, pages 7–10, 2003.

P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing*. Springer, 1987.

J. K. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

C. Liebchen, M. Lübbecke, R. Möhring, and S. Stiller. The Concept of Recoverable Robustness, Linear Programming Recovery, and Railway Applications. In *Robust and Online Large-Scale Optimization*, pages 1–27. Springer, 2009.

S. Pirkwieser and G. R. Raidl. A Column Generation Approach for the Periodic Vehicle Routing Problem with Time Windows. In *Proceedings of the International Network Optimization Conference*, volume 2009, 2009.

D. Potthoff, D. Huisman, and G. Desaulniers. Column generation with dynamic duty selection for railway crew rescheduling. *Transportation Science*, 44(4): 493–505, 2010.

ROADEF. Management of the mission of Earth observation satellites - Challenge description. `http://challenge.roadef.org/2003/files/formal$\_$250902.pdf`, September 2002.

M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

H. Tang, E. Miller-Hooks, and R. Tomastik. Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, 43(5):591–609, 2007.

T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, pages 797–809, 1984.

P. Vansteenwegen. The Team Orienteering Problem with Time Windows, Test Instances, c-r-rc-100-100. `http://www.mech.kuleuven.be/en/cib/op`, 2009.

P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden. Metaheuristics for Tourist Trip Planning. In *Metaheuristics in the Service Industry*, pages 15–31. Springer, 2009.

P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden. The orienteering problem: A Survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

L. P. Veelenturf, D. Potthoff, D. Huisman, L. G. Kroon, G. Maróti, and A. P. M. Wagelmans. A Quasi-Robust Optimization Approach for Crew Rescheduling. *Transportation Science*, 2014.

X. Wang, B. L. Golden, and E. A. Wasil. Using a Genetic Algorithm to Solve the Generalized Orienteering Problem. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 263–274. Springer, 2008.

J. Xu and S. Y. Chiu. Effective Heuristic Procedures for a Field Technician Scheduling Problem. *Journal of Heuristics*, 7(5):495–509, 2001.