Matheuristics for Robust Optimization

Application to Real-World Problems

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera Italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Nihat Engin Toklu

under the supervision of
Prof. Luca Maria Gambardella
Prof. Roberto Montemanni
Prof. Jürgen Schmidhuber

November 2014

Dissertation Committee

Prof. Evanthia Papadopoulou
University of Lugano

Prof. Laura Pozzi
University of Lugano

Prof. Christian Artigues
LAAS (Laboratory for Analysis and Architecture of Systems),
CNRS (National Center for Scientific Research),
Toulouse University

Prof. Marino Widmer
University of Fribourg

Dissertation accepted on 24 November 2014

Research Advisor

Prof. Luca Maria Gambardella

Co-Advisor

Prof. Roberto Montemanni

Academic Advisor

Prof. Jürgen Schmidhuber

PhD Program Director

Prof. Igor Pivkin

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Nihat Engin Toklu
Lugano, 24 November 2014

To my family

"We are getting way behind schedule"

Battlecruiser unit from StarCraft
(Blizzard Entertainment, 1998)

Abstract

In the field of optimization, the perspective that the problem data are subject to uncertainty is gaining more and more interest. The uncertainty in an optimization problem represents the measurement errors during the phase of collecting data, or unforeseen changes in the environment while implementing the optimal solution in practice. When the uncertainty is ignored, an optimal solution according to the mathematical model can turn out to be far from optimal, or even infeasible in reality.

Robust optimization is an umbrella term for mathematical modelling methodologies focused on finding solutions that are reliable against the data perturbations caused by the uncertainty. Among the relatively more recent robust optimization methodologies, an important concept studied is the degree of conservativeness, which can be explained as the amount of targeted reliability against the uncertainty while looking for a solution. Because the reliability and solution cost usually end up being conflicting objectives, it is important for the decision maker to be able to configure the conservativeness degree, so that the desired balance between the cost and reliability can be obtained, and the most practical solution can be found for the problem at hand.

The robust optimization methodologies are typically proposed within the framework of mathematical programming (i.e. linear programming, integer programming). Thanks to the nature of mathematical programming, these methodologies can find the exact optimum, according to the various solution evaluation perspectives they have. However, dependence on mathematical programming might also mean that such methodologies will require too much memory from the computer, and also too much execution time, when large-scale optimization problems are considered. A common strategy to avoid the big memory and execution time requirements of mathematical programming is to use metaheuristic optimization algorithms for solving large problem instances.

In this research, we propose an approach for solving medium-to-large-sized robust optimization problem instances. The methodology we propose is a matheuristic (i.e. a hybridization of mathematical programming and meta-

heuristic). In the matheuristic approach we propose, the mathematical programming part handles the uncertainty, and the metaheuristic part handles the exploration of the solution space. Since the exploration of the solution space is entrusted onto the metaheuristic search, we can obtain practical near-optimal solutions while avoiding the big memory and time requirements that might be brought by pure mathematical programming methods. The mathematical programming part is used for making the metaheuristic favor the solutions which have more protections against the uncertainty. Another important characteristic of the methodology we propose is concurrency with information exchange: we concurrently execute multiple processes of the matheuristic algorithm, each process taking the uncertainty into account with a different degree of conservativeness. During the execution, these processes exchange their best solutions. So, if a process is stuck on a bad solution, it can realize that there is a better solution available thanks to the information exchange, and it can get unstuck. In the end, the solutions of these processes are collected into a solution pool. This solution pool provides the decision maker with alternative solutions with different costs and conservativeness degrees. Having a solution pool available at the end, the decision maker can make the most practical choice according to the problem at hand.

In this thesis, we first discuss our studies in the field of robust optimization: a heuristic approach for solving a minimum power multicasting problem in wireless actuator networks under actuator distance uncertainty, and a linear programming approach for solving an aggregate blending problem in the construction industry, where the amounts of components found in aggregates are subject to uncertainty. These studies demonstrate the usage of mathematical programming for handling the uncertainty. We then discuss our studies in the field of matheuristics: a matheuristic approach for solving a large-scale energy management problem, and then a matheuristic approach for solving large instances of minimum power multicasting problem. In these studies, the usage of metaheuristics for handling the large problem instances is emphasized. In our study of solving minimum power multicasting problem, we also incorporate the mechanism of information exchange between different solvers. Later, we discuss the main matheuristic approach that we propose in this thesis. We first apply our matheuristic approach on a well-known combinatorial optimization problem: capacitated vehicle routing problem, by using an ant colony optimization as the metaheuristic part. Finally, we discuss the generality of the methodology that we propose: we suggest that it can be used as a general framework on various combinatorial optimization problems, by choosing the most appropriate metaheuristic algorithm according to the nature of the problem.

Acknowledgements

I would like to express my gratitude to my research advisors Luca Maria Gambardella and Roberto Montemanni, for helping and supporting me during my studies, and for allowing me to learn from their experience. Thanks to their guidance, I was able to come this far.

I would also like to say thanks to my academic advisor Jürgen Schmidhuber, and to my dissertation committee members Christian Artigues, Evanthia Papadopoulou, Laura Pozzi, and Marino Widmer, for their valuable comments and suggestions.

I am grateful to Davide Anghinolfi, Cristiano Nattero, and Massimo Paolucci for their big efforts on our large scale energy management study, which has become an important stepping stone in my thesis.

Finally, I would like to say *sağolun* to my family, for their endless support during all my endeavours in my life!

Contents

Figures

Tables

Chapter 1

Introduction

Let us consider the field of operations research, and the optimization problems studied within this field, related to transportation, supply chains, scheduling, etc. In operations research, such problems are mathematically modelled, and then these models are solved by mathematical programming (collective name for techniques like linear programming, integer programming, etc.), or by heuristic algorithms which find near-optimal solutions in short amounts of times.

A traditional approach in optimization is to model the problems deterministically, without the consideration of uncertainty. In deterministic optimization problems, all problem data (all the coefficients of the objective function and the constraints) are represented as fixed numbers. The reality, however, is not deterministic. Random behavior of the nature and/or the measurement errors during the collection of problem data causes the coefficients in the reality to differ from the nominal coefficient values (i.e. coefficient values assumed in the mathematical model). The difference between the nominal value of a coefficient and its value in reality is called the perturbation on that coefficient (Nemirovski [2009]).

The perturbations on coefficients can have very undesirable effects. For example, let us consider a cost minimization problem and let us consider that there are perturbations on the objective function coefficients. Because of these perturbations, a solution which seems optimal according to the nominal coefficient values can turn out to be far from optimal and very costly in reality. In addition, when we consider that there are perturbations on the constraint coefficients, the feasibility of the solution is in danger as a solution which is feasible according to the nominal coefficient values can be infeasible according to the coefficient values in reality.

To deal with the perturbations on the problem data, two schools of thoughts

emerged in the last decades: stochastic programming (Birge and Louveaux [1997]) and robust optimization (Kouvelis and Yu [1997]; Ben-Tal and Nemirovski [2000]; Bertsimas and Sim [2004a]). In the school of stochastic programming, the problems are modelled by using the information on the behavior of the uncertainty (i.e. the probability distribution information of the uncertain data), and solutions which will be satisfactory with high probabilities are sought. There might be cases, however, in which there is not enough information to define the uncertainty by probability distributions. In such cases, it can be easier to define lower and upper boundaries, or discrete collections of possible values for data perturbations. In this way of defining the uncertainty, it is not necessary to include probability distribution information in the mathematical model, it is enough just to say that the data are somehow perturbed complying with the boundaries/collections. For dealing with this kind of uncertainty, robust optimization can be used. We can think of robust optimization as an umbrella term, as there are multiple methodologies treating the uncertainty similarly. In the robust optimization school, mathematical models are prepared in a "cautious" way (with protective assumptions that some or all coefficients might end up having critical values) so that the solution will be satisfactory, always or most of the time.

Robust optimization methodologies are usually based on mathematical programming techniques like linear programming, mixed integer linear programming, etc. While these mathematical programming techniques are frequently used and are able to solve many popular problems, there are cases where heuristic methods are preferred. The reason for this preference is usually because mathematical programming techniques consume too much time and computer memory for solving a problem. To avoid big time and/or memory requirements of mathematical programming, one can use metaheuristic algorithms. A metaheuristic algorithm can be explained as a high-level master strategy, which can be used as a guideline for developing a heuristic search method to find near-optimal solutions for the optimization problem at hand (Yang [2011]). Genetic algorithms (Holland [1975]; Goldberg [1989]), simulated annealing (Kirkpatrick et al. [1983]) and ant colony optimization (Dorigo et al. [1991]; Dorigo [1992]) are examples for metaheuristic algorithms. Despite the fact that the metaheuristic algorithms do not guarantee to find the optimal solution, they can be preferred for their lower computational time and memory requirements, in comparison to the mathematical programming approaches. In the field of metaheuristics, a recent development is *matheuristic* algorithms (Raidl and Puchinger [2008]; Maniezzo et al. [2009]). A matheuristic algorithm is based on the ideas of metaheuristics, but also incorporates mathematical programming components

for solving a portion of the problem. In other words, a matheuristic is a hybridization of metaheuristics and mathematical programming.

An important concept in robust optimization is the degree of conservativeness, which means how much the wanted solution has to be protected against the uncertainty. This degree of conservativeness concept is previously discussed in Ben-Tal and Nemirovski [1999, 2000]. Let us first consider the type of uncertainty which affects the cost of a solution (because of uncertain coefficients in the cost function). With this type of uncertainty, in a solution space, there might be solutions which have very low potential costs, but also are very risky, in the sense that their costs could jump to much higher values in some critical scenarios. A more conservative solution could have higher minimum cost, but would be safer, as its cost would not suffer a significant increase in critical scenarios. Let us now consider the type of uncertainty which affects the feasibility of a solution (because of uncertain coefficients in the constraints). With this second type of uncertainty, there might be solutions which have very low costs, but also which can turn out to be infeasible in many critical scenarios. On the other hand, a more conservative solution could have a higher cost, but would be feasible in many more scenarios. It is useful for a decision maker to analyze solutions with different potential costs and different conservativeness degrees, so that she/he can see the effects of the uncertainty on the problem, and pick the solution which seems the most practical one.

In our research, the purpose is to build a matheuristic robust optimization framework, for finding practical uncertainty-aware heuristic solutions for larger problem instances. The characteristics of our framework are as follows:

- *Matheuristics.* The framework involves matheuristic hybridization: we embed the mathematical formulations of robust optimization into metaheuristic optimization algorithms. This allows us to do heuristic optimization avoiding the big time/memory requirements of pure mathematical programming approaches, while still evaluating each solution under uncertainty by using the ideas of robust optimization.

- *Solution pools.* Within this framework, the purpose is to generate solution pools, not single solutions. A solution pool is a collection of solutions with varying conservativeness degrees, and costs. Therefore, these solutions are alternatives to each other. For the decision maker, having a solution pool instead of a single solution can be much more useful, because, she/he can analyze various alternative solutions, see the trade-off between solution cost and robustness by looking at how the cost and robustness change

from solution to solution, and, in the end, she/he can pick the most practical solution. In our framework, solution pools are generated by executing multiple matheuristic optimization processes concurrently, with each optimization process focused on a different conservativeness degree. These processes also exchange solutions with each other so that no optimization process is stuck on a dominated solution.

Our research consists of multiple studies. These studies can be grouped as follows:

- *Robust optimization studies.* We have studied an aggregate blending problem, and minimum power multicasting problem in wireless actuator networks. While studying these problems, the techniques we have used do not involve matheuristics, but they do involve robust optimization, therefore, they are related to the overall goal of our research.

- *Matheuristic optimization studies.* Under this title, we have first studied the minimum power multicasting problem. The goal was not robustness, but the effectiveness in solving the large-scale instances heuristically, by executing a metaheuristic solver and a mathematical programming solver in parallel with the help of a solution sharing mechanism between the two solvers. In more details, when a solver is stuck on a local minimum, it can get unstuck by importing a better solution from the other solver. This is a matheuristic technique, as a mathematical programming solver and a metaheuristic solver are combined. Also, the solution sharing mechanism between the solvers is re-used in our matheuristic robust optimization framework. Secondly, we have studied a large-scale energy management problem. In the large-scale energy management problem, we have used a matheuristic technique, where an outer simulated annealing algorithm uses an embedded mathematical programming model to evaluate the performance of a solution, by also considering the uncertainty. In the way the uncertainty handled here, this problem is set apart from robust optimization, as a solution's average performance over the scenarios is measured, not the performance in the worst-case scenario. However, as a matheuristic approach and uncertainty are involved, this study can be considered as quite related, and a stepping stone to our final matheuristic robust optimization framework.

- *Study on matheuristic robust optimization.* This is the study which involves the main methodology proposed in this thesis. In this study, we

first apply the ideas of our matheuristic robust optimization for solving a well-known combinatorial optimization problem: vehicle routing problem (VRP). In more details, we have studied the two extensions of capacitated VRP (CVRP): capacitated vehicle routing problem with uncertain travel costs (CVRPU), and capacitated vehicle routing problem with time window constraints and uncertain travel times (CVRPTWU). Complying with the characteristics of our matheuristic robust optimization framework that we will discuss in the end, CVRPU and CVRPTWU are solved by executing multiple ant colony optimization (ACO) algorithms concurrently, with each ant colony focused on a different conservativeness degree, and in the end, solution pools are generated. After working on these CVRP variations, we also discuss the generality of this methodology, and how it can be treated as a framework for solving various combinatorial optimization problems under uncertainty.

The structure of this thesis is as follows. First, in chapter 2, we give a description of robust optimization and discuss the state of the art. Later, in chapter 3, we present our studies related to the field of robust optimization. Chapter 4 discusses our studies related to the field of matheuristics. In chapter 5, we present our matheuristic robust optimization studies for solving CVRP problems subject to uncertainty. In chapter 6, we discuss the generality of our matheuristic robust optimization methodology and we suggest that it can be considered as a framework for solving various combinatorial optimization problems with uncertain data. Finally, in chapter 7, we draw our conclusions.

Chapter 2

Classic Robust Optimization

2.1 Basic Description

Robust optimization approaches assume that the coefficients of a mathematical model can not be known exactly. So, depending on the particular robust optimization approach, a robust counterpart of a mathematical model is generated to explicitly consider the uncertainty within the optimization process.

Let us consider the following simple 0-1 linear program:

$$\begin{cases} \text{minimize} & \sum_{j \in J} c_j x_j & & \\ \text{subject to} & \sum_{j \in J} a_{ij} x_j \leq b_i & \forall i \in I & \\ & x_j \in \{0,1\} & \forall j \in J & \end{cases} \tag{2.1}$$

In the example 0-1 linear programming model (2.1), x_j is the j-th binary decision variable, c_j is the j-th coefficient of the objective function, a_{ij} is the j-th left-hand side coefficient of the i-th constraint, b_i is the right-hand side constant of the i-th constraint, J is the set of the indices for the coefficients (columns of the model) and I is the set of the indices for the constraints (rows of the model). Also, for simplicity, let us assume that all the coefficients (a_{ij}, b_i, c_j) are greater than or equal to 0.

The uncertainty is usually considered in two alternative forms: *interval form* and the *discrete scenario form*. They can be explained in further details as follows.

Uncertainty in the interval form. Let us consider the uncertainty representation in the interval form. If we assume that the objective coefficients are subject

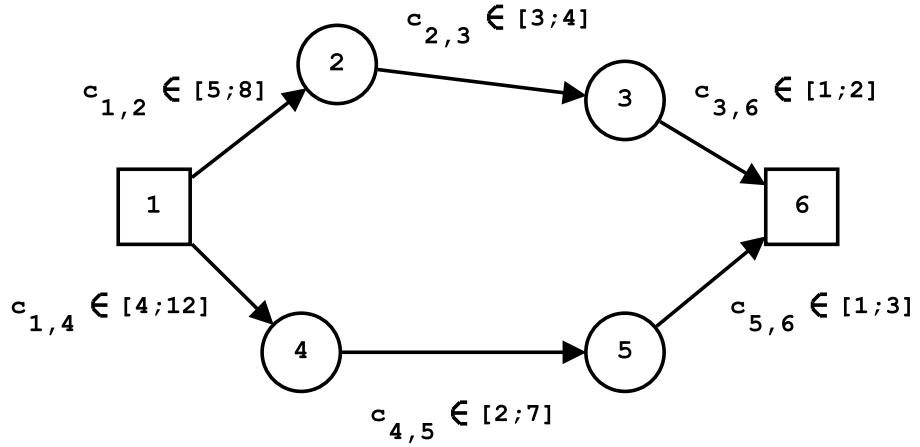


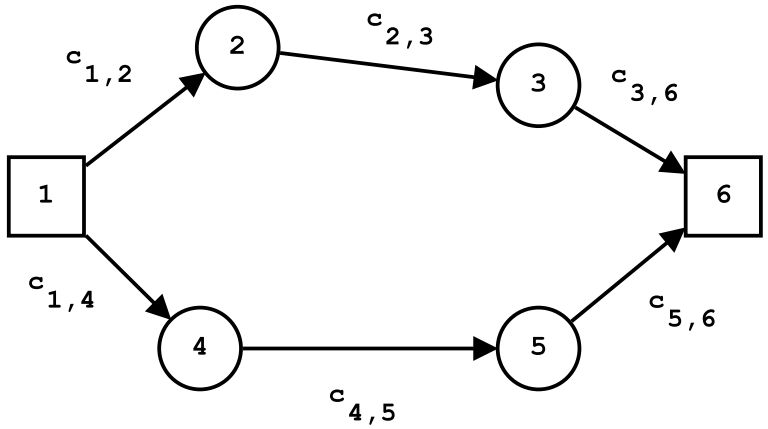
Figure 2.1. A shortest path problem instance, where the uncertainty is expressed in the interval form.

to uncertainty, we say that c_j are not exactly known numbers anymore, but they are $c_j \in [\underline{c}_j; \overline{c}_j]$. Similarly, for a_{ij} values under uncertainty, we say $a_{ij} \in [\underline{a}_{ij}; \overline{a}_{ij}]$. We assume in this case that a_{ij} and/or c_j values can turn out to be any value in the reality out of their intervals, and they are independent from each other. We also assume that within these intervals, we do not know anything related to the probability distributions: we do not know which values within these intervals are more likely. Finally, when representing the uncertainty via intervals, we define a scenario as a collection of assumptions, where a specific value is assumed by each uncertain coefficient out of its interval.

Uncertainty in the discrete scenario form. Let us consider the uncertainty representation in the discrete scenario form. This form of representation is used when we have a finite set of most likely scenarios available to us, where each scenario contains assumed values for uncertain coefficients.

Figures 2.1 and 2.2 summarize these two forms of uncertainty representation, by taking a small shortest path problem as an example.

The rest of this chapter will list robust optimization methodologies to give a short summary of the state of the art.



Scenario	$c_{1,2}$	$c_{2,3}$	$c_{3,6}$	$c_{1,4}$	$c_{4,5}$	$c_{5,6}$
1	6	3	1	6	5	2
2	5	4	2	5	2	1
3	7	4	2	6	2	3

Figure 2.2. A shortest path problem instance, where the uncertainty is expressed in the discrete scenarios form.

2.2 Exact Robust Optimization Methodologies

2.2.1 The Soyster Approach

In Soyster [1973], the author considered uncertainty in the interval form and proposed a methodology where the problem is optimized according to the most pessimistic scenario.

Let us think of the example (2.1) and let us assume that we have uncertainty on all coefficients. Considering that the uncertainty of interval form imposes $c_j \in [\underline{c}_j; \overline{c}_j]$ and $a_{ij} \in [\underline{a}_{ij}; \overline{a}_{ij}]$, when we apply the Soyster approach, we get:

$$\begin{cases} \text{minimize} & \sum_{j \in J} \overline{c}_j x_j & & \\ \text{subject to} & \sum_{j \in J} \overline{a}_{ij} x_j \leq b_i & \forall i \in I & \\ & x_j \in \{0,1\} & \forall j \in J & \end{cases} \tag{2.2}$$

that is, the minimization is done according to the maximum possible values of the coefficients.

Note that, even if in this example we assume that the worst-case scenario is the scenario where all the uncertain coefficients are maximized, in the case

of some other problems, it might not be straightforward to know what is the worst-case scenario. Therefore, a very general definition of the Soyster approach would be the approach of minimizing the solution cost according to the scenario where the cost coefficients (c_j) are perturbed in such a way that the total solution cost is maximized, and the constraint coefficients (a_{ij}) are perturbed in such a way that the constraint left-hand sides are as close as possible to violating the constraints.

The limitation of the Soyster approach is that it is not possible to configure the level of conservatism. By considering the most pessimistic scenario, the Soyster approach will always go with full conservatism.

2.2.2 Absolute Robustness Approach

The absolute robustness approach is discussed in Kouvelis and Yu [1997]. The authors consider the uncertainty in the discrete scenario form, affecting the objective function coefficients. According to this approach, considering that we have a set of scenarios S, and that the value of an objective function coefficient c_j within a scenario $s \in S$ is expressed by c_j^s, each solution is evaluated according to the scenario in which its cost is maximized. In other words, like in Soyster approach, each solution is evaluated according to its worst-case scenario. Therefore, the model (2.1) turns into:

$$\begin{cases} \text{minimize} & \max\left\{ \sum_{j \in J} c_j^s x_j \;\middle|\; s \in S \right\} & \\ \text{subject to} & \sum_{j \in J} a_{ij} x_j \leq b_i & \forall i \in I \\ & x_j \in \{0,1\} & \forall j \in J \end{cases} \tag{2.3}$$

Having a similar perspective with the Soyster approach, this absolute robustness approach shares the same drawback: the conservativeness degree can not be configured, and always the most robust solution is sought.

2.2.3 The Robust Deviation Approach

In Kouvelis and Yu [1997], in addition to the absolute robustness approach, an approach called robust deviation approach (also known as the minimax regret approach) is discussed. The authors consider the uncertainty in the form of discrete scenarios; however, the applications of this approach on problems with

uncertainty of interval form have been done as well (see, for example, Karaşan et al. [2001]; Montemanni and Gambardella [2005]; Montemanni [2007]).

The robust deviation approach is based on the concept of "regret". Let us think of the example problem (2.1) and let us assume that the objective coefficients c_j are subject to uncertainty. Also, let us define sol_1 as a solution that the decision maker accepted for the example problem (2.1) before knowing the exact values for c_j. Even if sol_1 is feasible, if the decision maker knew the exact values for c_j in advance, she/he would go for a cheaper solution sol_2. The difference between the solution costs of sol_1 and sol_2 is called the regret. The robust deviation approach is focused on finding the solution where the maximum possible regret is minimized.

Again, like in the Soyster approach and the absolute robustness approach, the robust deviation approach is limited in the sense that the decision maker is not able to configure the level of conservatism.

2.2.4 The Ellipsoid Approach

In Ben-Tal et al. (Ben-Tal and Nemirovski [1997, 1999, 2000]) and also independently in El Ghaoui et al. (El Ghaoui and Lebret [1997]; El Ghaoui et al. [1998]), the authors consider a robust optimization methodology where the conservatism can be configured. The idea can be explained as defining an ellipsoidal uncertainty set where the considered perturbations are contained. In other words, all the perturbations that we want to be protected against are included in the uncertainty set. The maximum total amount of perturbation from the uncertainty set is added into the objective function and/or into the left-hand side of the uncertain constraint for having robust solutions.

Let us assume that p_j values are perturbations on objective coefficients or the coefficients of a row i, depending on the problem. We can formulate the uncertainty set as:

$$U = \left\{ p_j \,\middle|\, \sqrt{\sum_{j \in J} (p_j)^2 (x_j)^2} \leq \Omega \right\} \tag{2.4}$$

In (2.4), by increasing Ω, the perturbations considered by the uncertainty set U can be increased. So, Ω is the conservativeness parameter.

The ellipsoid approach was applied to linear programming problems with uncertainty of the interval form on the coefficient constraints in Ben-Tal and Nemirovski [2000]. Also, in Bertsimas and Sim [2004b], the authors discuss the application of this approach into 0-1 linear programming problems with uncertainty on the objective function coefficients.

The ellipsoidal shape of the protection area imposes non-linear additions to an originally linear mathematical model. These non-linear additions turn the problem into conic quadratic problems.

As a linearized and simplified version of the ellipsoid approach, the polyhedral uncertainty set approach can be considered. In this simplified approach, the uncertainty set imposes that the total amount of perturbations are less than Θ (i.e. $U = \{p_j \mid \sum_{j \in J} p_j \leq \Theta\}$). However, in Ben-Tal and Nemirovski [1999], in favor of using the ellipsoid approach, the authors argue that "in some important cases, there are 'statistical' reasons which give rise to ellipsoidal uncertainty" and that large-scale conic quadratic problems can be solved by recent interior points optimization methods.

2.2.5 The Bertsimas-Sim Approach

In Bertsimas and Sim [2003, 2004a], the authors propose a robust optimization methodology for which the degree of conservatism can be configured. In addition to this, unlike the ellipsoid approach, the Bertsimas-Sim approach can be linearized assuming that the original problem is linear.

According to this approach, the uncertainty is bounded by the concept of uncertainty budget. For the objective function, the uncertainty budget is Γ_0. For the constraint i, the uncertainty budget is Γ_i. When a full perturbation happens on a coefficient (i.e. when the coefficient's value in reality is perturbed fully towards its worst case, making it equal to its maximum possible value), its related Γ (Γ_0 or Γ_i, depending on where the coefficient belongs) is decreased by 1. When, say, a half perturbation happens on a coefficient (i.e. when the coefficient is perturbed halfway towards its worst-case value, making it equal to the value at the middle between its best-case value and its worst-case value), its related Γ is decreased by 0.5. After the uncertainty budget is finished, we assume that no more perturbations will happen on the rest of the coefficients, therefore, we leave them at their best-case values. The uncertainty budgets represent the amount of perturbations allowed. By setting values for these uncertainty budgets, the decision maker can configure the conservatism.

When we apply the Bertsimas-Sim approach to (2.1), we get:

$$\begin{cases} \text{minimize} & \sum_{j\in J} \underline{c}_j x_j \\ & +\max\left\{ (\overline{c}_j - \underline{c}_j) x_j \;\middle|\; \sum_{j\in J} c_j^+ \le \Gamma_0;\ 0 \le c_j^+ \le 1\ \forall j \in J \right\} \\ \text{subject to} & \sum_{j\in J} \underline{a}_{ij} x_j \\ & +\max\left\{ (\overline{a}_{ij} - \underline{a}_{ij}) x_j \;\middle|\; \sum_{j\in J} a_{ij}^+ \le \Gamma_i;\ 0 \le a_{ij}^+ \le 1\ \forall j \in J \right\} \\ & \le b_i \qquad \forall i \in I \\ & x_j \in \{0,1\} \qquad \forall j \in J \end{cases} \tag{2.5}$$

where c_j^+ and a_{ij}^+ represent the perturbations on the coefficients c_j and a_{ij}, respectively. Notice that, unlike the ellipsoid approach, (2.5) does not contain non-linear operators like square and square root. Because of this, (2.5) can be expressed as a linear program.

The formulation (2.5) corresponds to the way of thinking which says that the Γ_0 number of coefficients which will maximize the cost, and the Γ_i number of coefficients which will (maybe critically) drive each constraint i towards the feasibility boundary denoted by b_i, are assumed to be in their highest values. To sum up, the most "disturbing" coefficients are assumed to be perturbed. This pessimistic way of thinking makes the model favor robust solutions.

2.3 Metaheuristic Robust Optimization Methodologies

In addition to the field of mathematical programming, uncertainty and robust optimization have also been studied in the field of heuristics. We now mention some examples within this field.

2.3.1 Robust Optimization with Simulated Annealing

In Bertsimas and Nohadani [2010], the authors propose a metaheuristic robust optimization technique. In this study, the authors consider problems in which the decision variables are continuous, and the uncertainty is on the decision variables.

In this approach, a simulated annealing algorithm is executed, with a given conservativeness degree, Δ. Given a solution x, and a minimization problem

P, let us now define DETEVAL(x) as the objective function of the deterministic counterpart of P, and ROBEVAL(x) as the objective function according to the uncertainty-aware counterpart of P. Let us also define $Around(x, \Delta)$ as a set of solutions around the solution x within the radius Δ. Now, in terms of DETEVAL, we can define the robust objective function as ROBEVAL(x) as:

$$\text{ROBEVAL}(x) = \max\{\text{DETEVAL}(x') \mid x' \in Around(x, \Delta)\}$$

Minimizing the result of ROBEVAL corresponds to following a min-max approach: considering that various outcomes can be obtained from a solution due to the uncertainty, we are trying to minimize the cost of the worst outcome.

2.3.2 Evolutionary Multiobjective Optimization Algorithms Dealing with Uncertainty

From the field of evolutionary algorithms, evolutionary multiobjective reliability-based optimization emerges. In this field, usually, a multiobjective evolutionary algorithm is developed, for handling a problem under uncertainty. The first objective is the minimization of the solution cost, and the second objective is the maximization of the *reliability* (or robustness, if the second objective models the uncertainty by following the school of robust optimization). The main advantage of this approach is that, the evolutionary algorithm, at the end of its execution, provides a solution pool, allowing the decision maker to see the trade-off between solution cost and reliability by analyzing various solutions. Among the frequently used multiobjective optimization algorithms for this kind of optimization, there are nondominated sorting genetic algorithm 2 (NSGA-II; see Deb et al. [2002]), and strength pareto evolutionary algorithm 2 (SPEA2; see Zitzler et al. [2001]). Some examples of evolutionary multiobjective reliability-based optimization studies are Li et al. [2005]; Gunawan and Azarm [2005]; Deb et al. [2009].

2.4 Matheuristic Framework for Robust Optimization

In section 2.2, we have discussed popular exact methods to retrieve robust solutions. These approaches, especially when we consider non-trivially sized instances of combinatorial optimization problems, might require large amounts of execution time and memory, because of many binary decision variables. The framework we ultimately propose in this thesis focuses on execution speed and

lower memory requirements, and provides a heuristic solution pool after a single concurrent execution.

For quickly obtaining heuristic solution pools for combinatorial optimization problems under uncertainty, the ideas of evolutionary multiobjective reliability-based optimization mentioned in section 2.3.2 can be applied as well: an evolutionary multiobjective optimization algorithm can be developed, in which the objectives are the cost evaluations according to various conservativeness degrees, and/or robustness in terms of satisfying the constraints. The importance of the techniques we propose within our framework is that they can be built upon an existing metaheuristic. Let us say we have an uncertainty-unaware combinatorial optimization problem P, and there is a well-known metaheuristic in the literature for solving P. In this case, for solving the uncertainty-aware counterpart of P, one can quickly apply the techniques proposed within our framework (concurrent generation of solution pools with information exchange), without having to make fundamental modifications on the underlying metaheuristic algorithm. Therefore, for obtaining solution pools for the uncertainty-aware counterpart of P, our approach can be more practical than implementing an evolutionary algorithm from scratch. Further discussions on this are made in chapter 6.

Chapter 3

Robust Optimization Studies

In this chapter, we discuss our studies (Montemanni et al. [2012]; Toklu and Montemanni [2011a,b, 2012b]) which are related to robust optimization. These studies are not part of our matheuristic framework, because no metaheuristic was involved.

3.1 Aggregate Blending Problem

We now present our aggregate blending problem study. In this study, a linear-programming-based approach (to be explained in section 3.1.2) was developed by R. Montemanni and N.E. Toklu. The results of this work show that this approach is competitive, for which the details will be discussed in section 3.1.3.

In the construction industry, the aggregate blending problem can be frequently faced. According to this problem, we have different fractions available. The goal is to mix these fractions into a final blend, the crucial decisions being how much of each fraction should exist within the final blend. Each fraction itself contains various amounts of ingredients, and we have to make sure that, in the final blend, each ingredient will be in the desired amounts, so that the final blend is strong and reliable.

In the aggregate blending problem, the amount of the ingredients in each sample of fraction is actually subject to uncertainty. Ignoring this uncertainty might lead to wrong assumptions about the quantities of the ingredients in the final blend. As a result of having wrong quantities, the quality of the final blend might be compromised. Because of this, a robust aggregate blending solution is a solution which stays away as much as possible from the infeasibility borders.

Examples of aggregate blending studies can be found in Lee [1973]; Tubacanon et al. [1980] for asphaltic concrete mixes and in Ritter and Shaffer [1961];

Neumann [1964] for sub-base granular material for highways. In addition to the construction industry, the blending problem can be encountered in other domains like in food, chemical, pharmaceutical and petrochemical industries (see Toklu [2005]; Venter [2010]).

In the literature, various methods were proposed for solving the aggregate blending problem: graphical technique, trial-and-error technique, linear programming, least-square optimization, and so on (see Lee [1973] and Toklu [2002] for a detailed review of the classical methods). Studies which consider the uncertainty were also reported in Sargent [1960], Easa and Can [1985b] and Easa [1985]. In Easa and Can [1985b], Easa [1985], and Toklu [2005], the aggregate blending problem was treated as a weighted multiobjective problem, the objectives being the minimization of the cost, and the maximization of the robustness. In Easa and Can [1985b] and Easa [1985], the authors used quadratic programming. In Toklu [2005] the authors used genetic algorithm. In our study Montemanni et al. [2012], we show that this multiobjective nature of the problem can be modeled by linear programming, and we then report our competitive results.

3.1.1 Formulation of the Aggregate Blending Problem

Let us now look at the classical formulation of the aggregate blending problem without the consideration of the uncertainty. First, we define the following problem data:

- $I \in \{1, 2, ...\}$: the set of ingredients (also $|I|$ is the number of ingredients).
- $J \in \{1, 2, ...\}$: the set of fractions (also $|J|$ is the number of fractions).
- $G_{ij} \in [0; 1]$: How much of fraction j consists of the ingredient i, represented as a ratio (in terms of volume or weight, depending on the particular problem at hand).
- $C_j(fa)$: Cost function associated with fraction $j \in J$, where the argument fa represents the fraction amount.
- $[s_i; r_i]$: The interval of desirable amount of ingredient $i \in I$ in the final blend.

Our decision variables are:

- x_j: The amount of fraction $j \in J$ in the final blend.

We are now ready to define how a valid aggregate blending solution looks like. The total amount of fraction amount ratios should be 1:

$$\sum_{j \in J} x_j = 1 \tag{3.1}$$

The amount of each ingredient i in the final blend must be within an interval:

$$s_i \leq \left(\sum_{j \in J} G_{ij} x_j \right) \leq r_i \quad \forall i \in I \tag{3.2}$$

The amount of fraction x_j for each $j \in J$ must be a positive real number:

$$x_j \geq 0 \quad \forall j \in J \tag{3.3}$$

Given these constraints, we can now summarize the classical aggregate blending problem, AGGBLEND, in which the objective is the minimization of the total cost, as follows:

$$\text{AGGBLEND} \begin{cases} \text{minimize} & \sum_{j \in J} C_j(x_j) \\ \text{subject to} & (3.1), (3.2), (3.3) \end{cases}$$

The model AGGBLEND, assuming that the cost function corresponds to a simple multiplication with a cost coefficient (i.e. assuming $C_j(x_j) = c_j x_j$), can be solved by using a linear programming solver (see IBM CPLEX [2014], Gurobi [2014], GNU Project [2014]).

When the cost function C_j is non-linear, we can approximate to each curve of C_j by using piecewise a linear cost function. This way of handling the non-linear cost functions is equivalent to how the "non-constant cost functions of aggregates" were handled in Easa and Can [1985a].

To express the piecewise linear cost function, let us now make the following definitions:

- $S(j)$: Number of linear pieces in the cost function of fraction j.
- α_{jk}: The cost at the beginning of the k-th linear piece of fraction j.
- a_{jk}: The amount ratio of fraction j which marks the beginning of its k-th linear piece. In other words, the k-th linear piece of fraction j is active when the amount of the fraction j is between a_{jk} and $a_{j,k+1}$.

In figure 3.1, an example of a piecewise linear cost function is given.

This piecewise linear nature of the problem can be solved by a mixed integer linear programming model. For the model, let us define two groups of decision variables:

$$y_{jk} = \begin{cases} 1 & \text{if the amount ratio of the fraction } j \text{ is between } \alpha_{jk} \text{ and } \alpha_{j,k+1} \\ 0 & \text{otherwise} \end{cases}$$

The other group of decision variables is u_{jk}, which stores how much of fraction j is used within the linear piece k. Given these definitions above, we are now ready express our model AGGBLENDPL, which considers the piecewise linear cost function, as:

AGGBLENDPL

$$\begin{cases} \text{minimize} & \displaystyle\sum_{j\in J}\sum_{k=1}^{S(j)} \left(a_{jk}y_{jk} + \frac{(a_{j,k+1}-a_{jk})}{(\alpha_{j,k+1}-\alpha_{jk})} u_{jk} \right) & (3.4) \\ \text{subject to} & \displaystyle\sum_{k=1}^{S(j)} y_{jk} = 1 \qquad \forall j \in J & (3.5) \\ & y_{jk} \geq u_{jk} \qquad \forall j \in J;\ \forall k = 1,2,\ldots,S(j) & (3.6) \\ & u_{jk} \leq \alpha_{j,k+1} - \alpha_{jk} \qquad \forall k = 1,2,\ldots,S(j) & (3.7) \\ & \displaystyle\sum_{j\in J}\sum_{k=1}^{S(j)} \left(\alpha_{jk}y_{jk} + u_{jk}\right) = 1 & (3.8) \\ & \displaystyle s_i \leq \sum_{j\in J} \left(G_{ij} \sum_{k=1}^{S(j)} \left(\alpha_{jk}y_{jk} + u_{jk}\right) \right) \leq r_i \qquad \forall i \in I & (3.9) \\ & y_{jk} \in \{0,1\} \qquad \forall j \in J;\ \forall k = 1,2,\ldots,S(j) & (3.10) \\ & u_{jk} \geq 0 \qquad \forall k = 1,2,\ldots,S(j) & (3.11) \end{cases}$$

where the constraints (3.5) impose that only one linear piece has to be selected. The constraints (3.6) say that the u_{jk} variables can have values greater than 0, only when the k-th linear piece is active (i.e. only when $y_{jk} = 1$). The constraints (3.7) define the upper bounds for the u_{jk} variables. The constraints (3.8) and (3.9) are the piecewise-linear-counterparts of (3.1) and (3.2), respectively. The constraints (3.10) and (3.11) define the domains of the variables y_{jk} and u_{jk}. Finally, the objective (3.4) is the piecewise-linear-counterpart of the objective of AGGBLEND, imposing the minimization of the total cost.

To sum up, when the cost function is purely linear, we can use the linear programming model AGGBLEND. When the cost function is nonlinear, we approximate that nonlinearity by expressing the cost function by using a piecewise

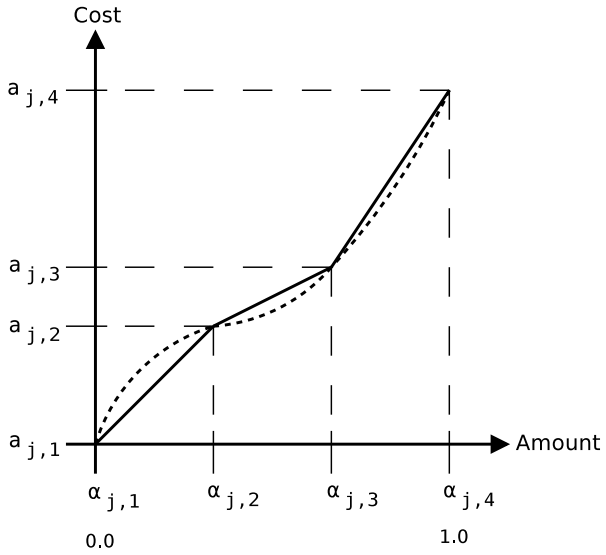


Figure 3.1. An example of piecewise linear cost function, approximating a nonlinear cost function (shown by the dotted curve). In this example, we have three linear pieces (i.e. $S(j) = 3$).

linear approach, and then we use the mixed integer linear programming model AGGBLENDPL. In section 3.1.2, we discuss the robust counterparts of these models.

3.1.2 Robust Linear Programming Formulation

In the reality, the amount of ingredient i in fraction j (i.e. G_{ij}), varies from sample to sample. Ignoring this fact might lead to a final blend with low qualities. Therefore, we need robust models which provides solutions protected against the uncertainty.

In the literature, probabilistic approaches exist (Tubacanon et al. [1980]; Lee and Olson [1983]) for finding solutions which are protected against the uncertainty. These approaches assume that the probability distributions of G_{ij}s are known. This assumption, however, brings the main drawback of stochastic optimization approaches: collecting the probability distribution information can prove to be very difficult for the decision maker. In fact, it might be as difficult as knowing the G_{ij} values. Therefore, a robust approach which does not require any probability distribution assumptions might be much more practical.

Previously, the robustness in aggregate blending was considered in Neumann [1964], where the authors minimized the mean deviation, ensuring that the so-

lution is as far as possible from the borders of infeasibility. In Easa and Can [1985a], the authors proposed a quadratic programming approach which can show the decision maker a trade-off between cost and robustness. In our study, we present a smaller robust model which can be solved by using linear programming, and which can, like the approach of Easa and Can [1985a], show the trade-off between cost and robustness.

Detailed comparisons with the studies in the literature will be given in section 3.1.3.

To define our robust model, let us first define w_i as the safest value within the interval $[s_i; r_i]$. This means, the most robust solution would be a solution in which the amount of each ingredient i is w_i. In many cases, intuitively, w_i would be set as the value in the middle of the interval $[s_i; r_i]$ (i.e. $r_i + ((r_i - s_i)/2)$). In the literature, various objectives were formulated to minimize the distance from w_i. The most popular formulations are:

$$D_1 = \sum_{i \in I} \left| w_i - \sum_{j \in J} G_{ij} x_j \right|$$

$$D_2 = \sum_{i \in I} \left(w_i - \sum_{j \in J} G_{ij} x_j \right)^2$$

$$D_3 = \max_{i \in I} \left\{ \left| w_i - \sum_{j \in J} G_{ij} x_j \right| \right\}$$

Both D_1 and D_2 measure the uncertainty-vulnerability by the total amount of (squared, in the case of D_2) deviation from $w_i \, \forall i \in I$. In D_3, the maximum among all the deviations from $w_i \, \forall i \in I$ is taken as the uncertainty-vulnerability measure. Taking the maximum deviation, instead of the total deviation, prevents the model from giving a solution in which the total deviation ends up being low but actually one of the ingredient amounts is critically close to the border of infeasibility. In our study, being inspired by D_3, we have introduced:

$$D_4 = \max_{i \in I} \left\{ \left| \frac{w_i - \sum_{j \in J} G_{ij} x_j}{\left(\frac{r_i - s_i}{2} \right)} \right| \right\}$$

Like in D_3, the maximum deviation is used as the measurement in D_4. However, in D_4, the amount of a deviation is calculated relative to the size of its $[s_i; r_i]$ interval, so that extra care is taken with critically small intervals.

We are now ready to define our robust model as the minimization of D_4, given a budget. For configuring the budget, we use a parameter δ, from which the cost upper bound is calculated by $C^*_{\text{AGGBLEND}} \cdot (1+\delta)$, where C^*_{AGGBLEND} is the optimal cost found by the uncertainty-unaware model AGGBLEND. Given these definitions, we formulate our robust model as follows:

$$
\text{RAGGBLEND} \begin{cases}
\text{minimize} & z & & (3.12)\\
\text{subject to} & \sum_{j\in J} C_j(x_j) \le C^*_{\text{AGGBLEND}}(1+\delta) & & (3.13)\\
& \sum_{j\in J} x_j = 1 & & (3.14)\\
& s_i \le \sum_{j\in J} G_{ij}x_j \le r_i & \forall i \in I & (3.15)\\
& \sum_{j\in J} G_{ij}x_j \ge w_i - \frac{r_i - s_i}{2} z & \forall i \in I & (3.16)\\
& \sum_{j\in J} G_{ij}x_j \le w_i + \frac{r_i - s_i}{2} z & \forall i \in I & (3.17)\\
& x_j \ge 0 & \forall j \in J & (3.18)
\end{cases}
$$

where the objective (3.12) imposes the minimization of the variable z, which stores the value of D_4. The constraint (3.13) imposes the cost upper bound configured by the parameter δ. The constraint (3.14) says that the total amount of the fractions must be 1. The constraint (3.15) is the same as (3.2), saying that the $[s_i; r_i]$ interval must be respected. The constraints (3.16) and (3.17) shrink the interval according to the value of z, and impose that the amount of each ingredient i stays within its related shrunk interval. The shrinking of interval is as follows: when $z = 1$, an interval $[s_i; r_i]$ is not shrunk at all, staying the same. When, $z < 1$, the interval $[s_i; r_i]$ is shrunk around w_i values, its size becoming the original size multiplied by z_i, therefore becoming: $\left[w_i - \frac{r_i - s_i}{2} z \;;\; w_i + \frac{r_i - s_i}{2} z \right]$. To sum up, the model is focused on shrinking the $[s_i; r_i]$ intervals as much as possible and find a feasible solution for these shrunk intervals without violating the budget. As these intervals get shrunk more and more, the ingredient amounts in the solution become closer to w_i values, and farther from the borders of $[s_i; r_i]$, therefore increasing the robustness.

Note that all the constraints are linear in RAGGBLEND. This means, adding robustness considerations into the original model AGGBLEND does not increase the time complexity. In case where we have a nonlinear cost function to which it is possible to approximate by using a piecewise linear cost function, the same

robustness approach can be applied on AGGBLENDPL, and the robust counterpart of AGGBLENDPL model would stay as a mixed integer linear programming model. In practice, an overhead in terms of time exists because of the fact that the robust models need the cost of the optimal solution of the uncertainty-unaware model (in the budget constraint (3.13)). This requirement, however, can be avoided, if the decision maker has an absolute constant budget value available in advance.

3.1.3 Experimental Results

We now compare the results of our approach with the results of some previous studies appeared in the literature.

3.1.3.1 Comparison against Ritter and Shaffer [1961]

In Ritter and Shaffer [1961], the first usage of linear programming for aggregate blending is introduced. Uncertainty was not considered, and a technique for approximating nonlinear cost functions was not proposed. The model is similar to AGGBLEND, with an additional contraint for plasticity index, which was also inserted into our models for the sake of comparison.

The authors of Ritter and Shaffer [1961] use an example of with 37 fractions and 9 ingredients. They solve this example to optimality, and they also discuss some alternative solutions. These alternative solutions, however, seem to break some constraints, probably because of numerical issues introduced by their solving tools. In this comparison, we use their feasible and optimal solution. The comparison is presented in table 3.1, where the optimal solution of Ritter and Shaffer [1961], the optimal solution of AGGBLEND, and more robust alternative solutions of RAGGBLEND are evaluated in terms of cost and D_4. We have to stress that, we excluded the ingredients 4 and 7 from the evaluation of robustness, because each feasible solution has to have the amount of these ingredients at the border of infeasibility, therefore including these ingredients in the robustness evaluation would always result in $D_4 = 1$. In the table, we can see that the optimal solutions of Ritter and Shaffer [1961] and AGGBLEND are almost the same, the difference being probably introduced by numerical issues. In these optimal solutions, we can observe that the cost is very low, however, also so is the robustness: $D_4 = 1$ means that the amount of at least one of the ingredients is touching the infeasibility border. We can see that the solutions of RAGGBLEND can increase the robustness (i.e. can decrease D_4), by sacrificing in terms of solution cost, the amount of this sacrifice being dependent on δ.

Table 3.1. Comparison of our approach against the study Ritter and Shaffer [1961]

Solution	Cost	D_4
Ritter and Shaffer [1961]	0.0847	0.9999
AGGBLEND	0.0831	1.0000
RAGGBLEND: $\delta = 0.30$	0.1081	0.6103
RAGGBLEND: $\delta = 0.60$	0.1330	0.4227
RAGGBLEND: $\delta = +\infty$	0.1552	0.3170

3.1.3.2 Comparison against Neumann [1964]

In Neumann [1964], a quadratic programming model was proposed, in which the minimization of the solution cost was not considered. The focus was on maximizing the robustness. Differently from our approach where D_4 is used, in Neumann [1964], D_2 was used as the measurement of robustness. The authors used an example with 3 fractions and 8 ingredients. The comparison of our results against theirs are given in table 3.2. In the table, it can be seen that, compared to AGGBLEND and RAGGBLEND, the model of Neumann [1964] can indeed find solutions with much lesser D_2 values. However, when we measure the robustness of these solutions in terms of D_4, these solutions seem to be touching the border of infeasibility (i.e. $D_4 \approx 1$). On the other hand, the solution of RAGGBLEND, seems to be the most robust one in terms of D_4, even if it has higher D_2 value. This means that, the understanding of robustness of D_2 and the understanding of robustness of D_4 are not the same, driving the solver into different solutions.

At this point, we would like to stress the main point of D_4: especially when facing $[s_i; r_i]$ intervals of different sizes, smaller intervals would be more critical than the others, and it would be more important to measure the robustness by considering the closeness to the safety curve relative to the interval sizes, resulting in paying extra attention for those critical intervals.

3.1.3.3 Comparison against Tubacanon et al. [1980]

In Tubacanon et al. [1980], the authors present an approach in which the purpose is the minimization of the cost, but the uncertainty is also considered: when a solution violates the $[s_i; r_i]$ intervals probabilistically, a penalty is added onto the cost. Because the probabilities are handled directly, the model is not linear. Therefore, the authors propose to approximate this model by using an algorith-

Table 3.2. Comparison of our approach against the study Neumann [1964]

Solution	D_2	D_4
Neumann [1964]: solution 1	101.85	1.0802
Neumann [1964]: solution $2^{(a)}$	70.30	1.0456
AGGBLEND	215.89	0.9524
RAGGBLEND:$\delta = +\infty^{(b)}$	174.43	0.8736

(a) This solution is not feasible according to a modern quadratic programming solver (IBM CPLEX [2014]). The problem could be caused by numerical issues.
(b) Since the minimization of the cost is not a consideration in the study of Neumann [1964] (i.e. the budget is infinite), we set the budget parameter of our approach as $+\infty$.

mic approach in which a linear programming model is solved iteratively. The stopping criterion of this iterative approach does not tell how far the current solution is from the optimality, and there is not a defined upper bound for number of iterations. This implies that the approach is not a polynomial-time algorithm.

The authors test their approach on an example with 4 fractions and 9 ingredients. The results are presented in table 3.3, where each solution is evaluated in terms of the penalty system they propose, and also in terms of D_4. In the example, it can be seen that our approach is able to improve upon the solution of Tubacanon et al. [1980], in terms of cost (when AGGBLEND is used, without considering uncertainty at all), in terms of robustness (when RAGGBLEND with $\delta = +\infty$ is used), and in terms of minimization of Penalty+Cost (when RAGGBLEND with $\delta = 0.15$ is used). It can also be seen that the minimization of D_4 is compatible with the robustness criterion of Tubacanon et al. [1980], as the solution with minimum D_4 yields zero penalty.

3.1.3.4 Comparison against Lee and Olson [1983]

In Lee and Olson [1983], a probabilistic model is proposed. For each G_{ij}, a normal probability distribution is associated. Their proposed model involves a hierarchical objective function: at each level of hierarchy, a sub-objective is optimized, and a penalty is paid if a constraint is not satisfied. The hierarchy of these sub-objectives are ordered as follows (from the most important to the least important):

1. Satisfy the $[s_i; r_i]$ intervals, ignoring the uncertainty at this level.

Table 3.3. Comparison of our approach against the study Tubacanon et al. [1980]

Solution	Cost	Penalty	Penalty+Cost	D_4
Tubacanon et al. [1980]	90.19	8.80	98.99	0.5818
AGGBLEND	**81.06**	940.84	1021.90	1.0000
RAGGBLEND : $\delta = 0.10$	89.23	15.66	104.89	0.7144
RAGGBLEND : $\delta = 0.15$	93.31	0.66	**93.97**	0.5716
RAGGBLEND : $\delta = 0.20$	97.44	0.01	97.45	0.4309
RAGGBLEND : $\delta = 0.25$	101.38	**0.00**	101.38	0.3739
RAGGBLEND : $\delta = +\infty$	103.43	**0.00**	103.43	**0.3428**

2. Satisfy the budget constraint (analogous to the budget constraint of RAGGBLEND model).

3. Minimize the probability of violating the $[s_i; r_i]$ intervals, taking into account the uncertainty. Dealing with probability distributions, this step adds nonlinearity to the model.

4. Minimize the total cost.

5. Satisfy the constraint which says that a given amount of a certain fraction has to be used.

The authors propose the usage of their approach in two modes. The first of these modes is the *deterministic* mode, in which the 3rd sub-objective is disabled. Since the nonlinearity of the model comes from the 3rd sub-objective, the deterministic mode reduces the model into a linear approach, solvable in polynomial-time. The second mode is *probabilistic* mode, in which the 3rd sub-objective is enabled, making the model nonlinear.

Seeing that the fifth sub-objective is the least important one in the hierarchy, and that the results presented in Lee and Olson [1983] do not seem to be satisfying it, we have excluded it from the comparison.

In Lee and Olson [1983], the authors use an example with 8 ingredients and 9 fractions. They generate two solutions: one deterministic solution, and one probabilistic solution. In table 3.4, these solutions are compared to our solutions, in terms of cost, the robustness criterion of Lee and Olson [1983] called *Critical Constraint Satisfaction Probability* (CCSP), and D_4. In the results, the first observation we can make is that the deterministic solution of Lee and Olson

Table 3.4. Comparison of our approach against the study Lee and Olson [1983]

Solution	Cost	CCSP	D_4
Lee and Olson [1983]: deterministic	1.035	0.3366	1.0000
Lee and Olson [1983]: probabilistic	1.328	0.6074$^{(a)}$	0.7720
AGGBLEND	1.038	0.3314	1.0000
RAGGBLEND : $\delta = 0.05$	1.090	0.4728	0.5956
RAGGBLEND : $\delta = 0.10$	1.142	0.5512	0.5508
RAGGBLEND : $\delta = +\infty$	1.150	0.5455	0.5472

(a) Erroneously reported as 0.8453 in Lee and Olson [1983]

[1983] and the solution of AGGBLEND are very similar. The second observation we can make is that the probabilistic solution of Lee and Olson [1983] is the one with the highest CCSP value, but, in terms of D_4, it is the RAGGBLEND model which has found the most robust solution. When the probability distributions are completely known, one could go for the approach of Lee and Olson [1983], to make sure that the probability of satisfying the critical constraints is high. However, here, the classical decision issue between a stochastic optimization approach (which is the approach of Lee and Olson [1983], considering that it relies on probability distribution information) and a robust optimization approach shows itself: it might be very difficult to find reliable information about the probability distributions of the G_{ij} values, and using the approach of Lee and Olson [1983] without truly knowing the probability distributions might result in solutions which are much less reliable than anticipated. For obtaining robust solutions without knowing anything except the boundaries of $[s_i; r_i]$ intervals, it could be much more practical to use our RAGGBLEND approach.

3.1.3.5 Comparison against Easa and Can [1985a]

Easa and Can [1985a] propose a mathematical programming approach very similar to our AGGBLEND and RAGGBLEND approaches, the difference being that, in Easa and Can [1985a], the authors minimize D_2 (instead of D_4 which is used by RAGGBLEND), and use quadratic programming to handle the square operation within D_2 (instead of linear programming used by RAGGBLEND to handle D_4). The authors also show that their quadratic programming formulation is solvable in polynomial time. Like RAGGBLEND, they provide a budget constraint, so that the trade-off between cost and robustness can be explored by the decision maker.

The authors use an instance with 8 ingredients and 3 fractions. By using

Table 3.5. Comparison of our approach against the study Easa and Can [1985a]

Solution	Cost	D_2	D_4
Easa and Can [1985a] solution $1^{(a)}$	13.72	$\mathbf{62.5} \cdot 10^4$	0.7649
Easa and Can [1985a] solution 2	13.50	$63.5 \cdot 10^4$	0.7648
Easa and Can [1985a] solution 3	13.25	$64.8 \cdot 10^4$	0.7654
Easa and Can [1985a] solution 4	13.00	$66.2 \cdot 10^4$	0.7682
Easa and Can [1985a] solution 5	12.85	$67.1 \cdot 10^4$	0.7702
Easa and Can [1985a] solution $6^{(a)}$	12.75	$67.9 \cdot 10^4$	0.7618
Easa and Can [1985a] solution $7^{(a)}$	12.50	$87.2 \cdot 10^4$	**0.6464**
Easa and Can [1985a] solution 8	12.30	$125.4 \cdot 10^4$	0.9036
Easa and Can [1985a] solution 9	**12.27**	$139.1 \cdot 10^4$	0.9825
AggBlend	**12.27**	$139.1 \cdot 10^4$	0.9825
RAggBlend: $\delta = 0.05$	12.88	$89.5 \cdot 10^4$	0.6329
RAggBlend: $\delta = 0.10$	13.49	$\mathbf{85.6} \cdot 10^4$	0.6274
RAggBlend: $\delta = +\infty$	13.50	$86.4 \cdot 10^4$	**0.6273**

The best solutions in terms of cost, D_2, and D_4 for each approach are written in bold.

(a) This solution is not feasible, according to Gurobi solver (Gurobi [2014]). This solely depends on the advances in the numerical precisions of the solvers, and not on a mistake of the authors of Easa and Can [1985a].

various budget levels, they produce various solutions. In table 3.5, our results are compared to theirs. The first observation is that, when the uncertainty is ignored and the purpose is set as the minimization of the cost, both approaches can find the optimum solution. The second observation is that the approach of Easa and Can [1985a] is able to find the solution with the minimum D_2. When it comes to D_4, however, the best solutions come from our approach. Therefore, our comments on the study of Neumann [1964] about the difference between D_2 and D_4 apply here too.

3.1.3.6 Comparison against Easa and Can [1985b]

In Easa and Can [1985b], the authors propose an extension to their previous study (Easa and Can [1985a]), where both the cost and D_2 are formulated within the objective function, each multiplied by a weight coefficient. By configuring these weight coefficients, the decision maker can explore the trade-off

Table 3.6. Comparison of our approach against the study Easa and Can [1985b]

Solution	Cost	D_2	D_4
Easa and Can [1985b]: solution 1	**12.65**	71.12	1.0000
Easa and Can [1985b]: solution 2	12.65	71.12	1.0000
Easa and Can [1985b]: solution 3	12.68	49.01	0.9937
Easa and Can [1985b]: solution 4	12.70	35.91	0.9977
Easa and Can [1985b]: solution 5	12.72	29.12	1.0000
Easa and Can [1985b]: solution 6	12.73	27.13	1.0000
Easa and Can [1985b]: solution 7	12.73	26.39	1.0000
Easa and Can [1985b]: solution 8	12.73	26.08	1.0000
Easa and Can [1985b]: solution 9	12.81	21.89	0.8070
Easa and Can [1985b]: solution 10	12.95	17.54	0.4684
Easa and Can [1985b]: solution 11	13.05	**16.50**	**0.2511**
AGGBLEND	**12.65**	70.25	1.0000
RAGGBLEND:$\delta = 0.01$	12.77	41.09	0.7530
RAGGBLEND:$\delta = 0.02$	12.90	23.19	0.5058
RAGGBLEND:$\delta = 0.03$	13.03	16.56	0.2587
RAGGBLEND:$\delta = +\infty$	13.04	**16.52**	**0.2461**

The best solutions in terms of cost, D_2, and D_4 for each approach are written in bold.

between the cost and robustness.

In their study, the authors use an example with 7 ingredients and 3 fractions. By using different weights, the authors generate various solutions. The comparison of our results on this example against their results are shown in table 3.6. In the results, again, it can be seen that while the approach of Easa and Can [1985b] minimizes the D_2, it is the RAGGBLEND model which finds the most robust solution in terms of D_4. Therefore, our comments in Neumann [1964] and in Easa and Can [1985a] apply here too.

3.1.3.7 Comparison against Toklu [2005]

In Toklu [2005] a multiobjective genetic algorithm is proposed, where the multiobjective nature of the model is handled by adding the cost and robustness into a single objective, by using weight coefficients. For measuring the robustness, the authors use D_1, $\sqrt{D_2}$, and D_3 formulations.

The authors use an example with a linear cost function, consisting of 10

Table 3.7. Comparison of our approach against the study Toklu [2005], considering the examples with linear costs

Solution	Cost	D_1	$\sqrt{D_2}$	D_3	D_4
Toklu [2005]:					
D_1	16.7170	**9.4207**	3.9113	5.6000	0.6405
$\sqrt{D_2}$	16.6723	9.6270	**3.5983**	2.2630	0.6276
D_3	16.7680	11.4534	4.0567	**2.0298**	0.6339
0.75 D_1+0.25 Cost	14.9846	9.7405	4.3105	2.6704	0.6165
0.50 D_1+0.50 Cost	11.8899	12.1303	5.3276	2.8477	**0.6119**
0.25 D_1+0.75 Cost	7.7629	17.9950	8.0723	5.7925	0.8274
Cost	**7.2173**	30.6231	12.6856	7.9999	1.0000
AggBlend	**7.2173**	30.6238	12.6858	8.0002	1.0000
RAggBlend:δ=0.05	7.5781	24.5230	10.0415	5.9763	0.7471
RAggBlend:δ=0.20	8.6607	21.0990	8.6244	4.9765	0.6221
RAggBlend:δ=0.35	9.7433	19.1429	7.9595	4.6895	0.5861
RAggBlend:δ=$+\infty$	16.2979	**18.0596**	**7.6141**	**4.5149**	**0.5644**

The best solutions in terms of cost, D_1, $\sqrt{D_2}$, D_3, and D_4 for each approach are written in bold.

ingredients and 4 fractions. On this example, they obtain various solutions by configuring their genetic algorithm approach into minimizing D_1, $\sqrt{D_2}$, D_3, $(0.75D_1 \cdot 0.25Cost)$, $(0.50D_1 \cdot 0.50Cost)$, $(0.25D_1 \cdot 0.75Cost)$, and, finally, the solution cost. The comparison between their solutions and our solutions are given in table 3.7. In addition, to demonstrate that the genetic algorithm can handle nonlinear cost functions directly, the authors use a version of the previous example with a nonlinear cost function, and obtain various solutions on it by using various configurations. The comparison between their solutions for the nonlinear example, and our solutions which were obtained by approximating the nonlinearity by using piecewise linear cost function, are given in table 3.8. In both tables, it can be seen that both approaches were able to find the optimal solution when the goal was the minimization of the cost. We can also see that the genetic algorithm approach was able to find the best solutions in terms of D_1, $\sqrt{D_2}$, and D_3. Finally, once again, when we consider D_4, which is the only D_n robustness criterion relative to the interval sizes, we see that RAggBlend finds the best solutions.

Table 3.8. Comparison of our approach against the study Toklu [2005], considering the examples with non-linear costs

Solution	Cost	D_1	$\sqrt{D_2}$	D_3	D_4
Toklu [2005]:					
D_1	14.5952	**9.4207**	3.9113	5.6000	0.6405
$\sqrt{D_2}$	14.4811	9.6270	**3.5983**	**2.2630**	0.6276
D_3	14.4368	11.4534	4.0567	2.0298	0.6339
0.75 D_1 + 0.25 Cost	14.4313	9.4511	3.8033	2.3102	0.6075
0.50 D_1 + 0.50 Cost	11.3894	12.1206	5.3210	2.8470	**0.6042**
0.25 D_1 + 0.75 Cost	7.1004	17.9959	8.0736	5.7945	0.8273
Cost	**6.3643**	30.6233	12.6857	8.0000	1.0000
AggBlend	**6.3643**	30.6238	12.6858	8.0001	1.0000
RAggBlend:$\delta = 0.05$	6.6825	26.1468	10.8384	6.6335	0.8291
RAggBlend:$\delta = 0.20$	7.6372	21.9262	8.8863	5.0864	0.6358
RAggBlend:$\delta = 0.35$	8.5918	20.2518	8.3607	4.8643	0.6080
RAggBlend:$\delta = +\infty$	10.2247	**18.2983**	**7.5159**	**4.5149**	**0.5644**

The best solutions in terms of cost, D_1, $\sqrt{D_2}$, D_3, and D_4 for each approach are written in bold.

3.1.4 Protection against uncertain input data: an empirical study

The aim of this section is to validate the robust model RAGGBLEND from an empirical point of view. The solutions provided by RAGGBLEND for different values of the budget δ are assessed for different levels of uncertainty. Each level is identified by a parameter α. For each value of α, scenarios are generated with noise affecting each passing percentage G_{ij}; the modified value for G_{ij} is generated at random in $Norm[Gij; \alpha(G_{ij})^2]$: the normal distribution with meaning G_{ij} and variance $(\alpha G_{ij})^2$. Values of α up to 3% are considered. In the literature, it has been estimated that values of α on the order of 2% are typical in practice (Easa and Can [1985b]). Practical situations are therefore covered by the experiments described here.

For each uncertainty value α considered, one million scenarios were generated as described previously starting from the instance with linear costs taken from Toklu [2005]. The solutions of models AGGBLEND and RAGGBLEND (with different budgets δ) were generated according to the original G_{ij} values and checked for feasibility on each of the scenarios. The proportion of scenarios on which the provided solution was feasible is plotted in figure 3.2 for each model. Uncertainty levels (α) are plotted on the x-axis. The budget parameter α ranges between 0.0 (model AGGBLEND) and 0.6. Higher budget values yielded no improvement in the quality of the solution.

Figure 3.2 suggests that model RAGGBLEND provides important protection against uncertainty even when just a small budget $\delta = 1\%$ is set. However, small budget values lead to a reduction in the solution quality for high uncertainty levels α. Larger budget values are required to have good protection for all the uncertainty levels considered. In general, satisfactory protection is achieved with $\delta = 10\%$. Larger budget values lead to better protection. The robust model improves upon the solution obtained by model AGGBLEND in the proportion of satisfied scenarios up to approximately 0.55 for $\alpha = 0.02$ (a typical value in practice, according to Easa and Can [1985b]).

3.1.5 Validation of the optimization criterion D_4

A marginal innovation introduced in this study is the optimization criterion D_4. While introducing the robust model RAGGBLEND, we argued that D_4 is a promising criterion from a theoretical point of view. In this section we validate it against previously appeared criteria from an empirical point of view. We consider two variations of the robust model RAGGBLEND, in which criteria D_1 and D_3 are implemented instead of D_4. Criterion D_2 was not tested since its implementation

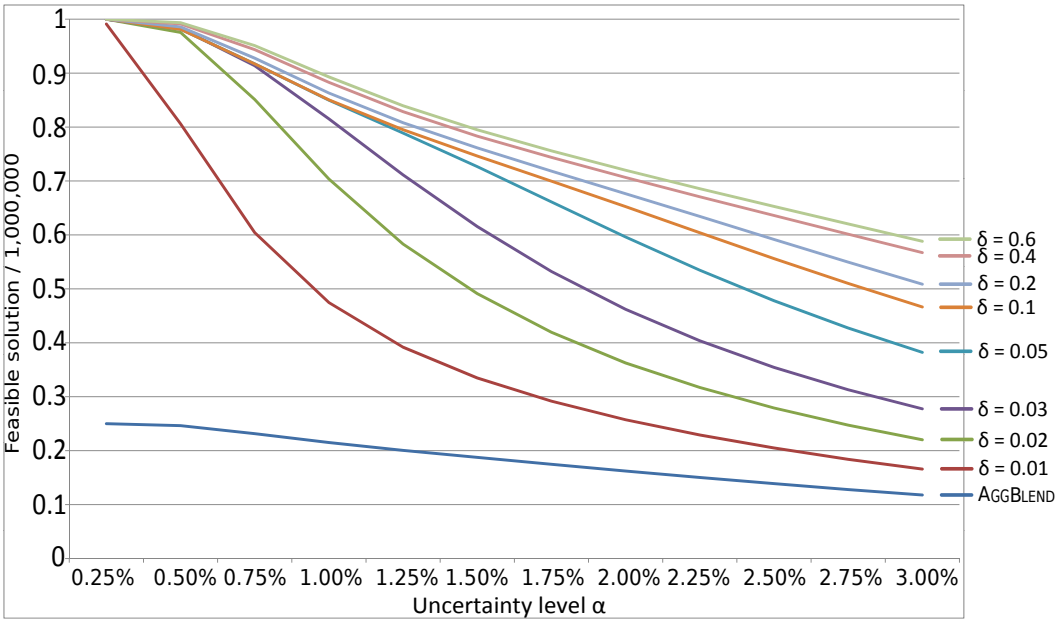


Figure 3.2. Protection provided by models AggBlend and RAggBlend with different budgets δ. The curve of AggBlend is labeled as "AggBlend". The curves of RAggBlend are labeled by their δ values.

would produce a non-linear model, which falls outside the scope of the present work.

In order to plug the criterion D_1 into our model, the following modifications to RAggBlend have to be done. The resulting model will be referred to as RAggBlend$_{D_1}$. A new set of free variables has to be introduced: z_i will contain the value $|w_i - \sum_{j=1}^{m} G_{ij}x_j|$ for each $i \in I$. Variable z is not used anymore. The objective function (3.12) is substituted by the following one:

$$\text{minimize} \sum_{i \in I} z_i \tag{3.19}$$

while the constraints (3.16) and (3.17) are substituted by:

$$\sum_{j \in J} G_{ij}x_j \geq w_i - z \qquad \forall i \in I \tag{3.20}$$

$$\sum_{j \in J} G_{ij}x_j \leq w_i + z \qquad \forall i \in I \tag{3.21}$$

The same experimental setups described before are adopted, and the results obtained by the original model RAggBlend and by its modifications RAggBlend$_{D_1}$ and RAggBlend$_{D_3}$ are reported in figure 3.3. Ratios

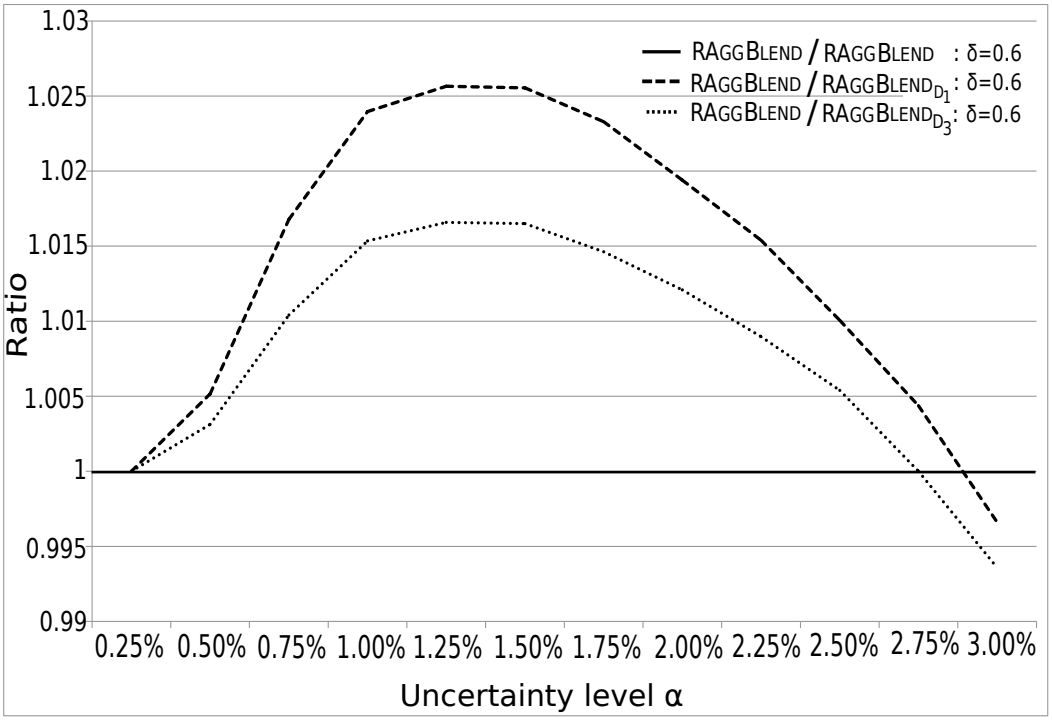


Figure 3.3. Protection provided by different optimization criteria. Budget $\delta = 0.6$.

$\frac{\text{RAggBlend}}{\text{RAggBlend}_{D_1}}$ and $\frac{\text{RAggBlend}}{\text{RAggBlend}_{D_3}}$ are plotted for budget $\delta = 0.6$ and increasing values of the uncertainty α. The chart suggests that RAggBlend provides a superior protection against uncertainty than RAggBlend_{D_3} and (especially) RAggBlend_{D_1} for most of the values of α considered. The exception is represented by the highest values of α considered: in these cases the different models tend to perform the same, with RAggBlend being even worse than the others for $\alpha \to 0.3$. This intuitively happens because the unpredictable random factor tends to be dominant under these settings, and criticalities (at the basis of D_4) are lost. A hierarchic objective function, with D_4 as the prominent criterion and D_3 as the secondary one, would probably be more indicated in such a context. Notice however that - according to Easa and Can [1985b] - these values of α are unlikely in practice. It is finally worth to mention that different budget values of δ generate charts similar to that proposed in figure 3.3.

3.1.6 Summary

A robust optimization approach for aggregate blending problem was studied. Differently from other approaches in the literature, we measure the robustness relatively to the size of the interval associated with the uncertain variable. This allows our approach to take extra care with the critical intervals.

Considering the instance sizes of aggregate blending problem in the literature (usually less than 10 ingredients, and less than 10 fractions), and the fact that the main decision variables are continuous, this study is a good example of using mathematical programming for solving a robust optimization problem. However, when it comes to bigger problems with combinatorial nature (i.e. with binary decision variables), using a mathematical programming approach would not be practical, as it would require too much execution time and memory. The main focus of the present thesis is the handling of such big combinatorial optimization problems by using heuristic algorithms, which will be explained in chapters 5 and 6.

3.2 Minimum Power Multicasting Problem

The minimum power multicasting problem (MPMP), and the minimum power broadcasting problem (MPBP) are faced when wireless networks are to be established in human-unfriendly environments, or in places where the existing communication infrastructure is damaged by natural disasters, etc. (Leggieri et al. [2008]). In MPMP, our assumption is that we have devices that we call *terminals*, each terminal being able to receive and transmit data wirelessly by using their antennas (Rappaport [1996]). Thanks to their ability to receive and transmit, these terminals can act as routers too, redirecting the data they receive to others. We also consider that these terminals are mobile devices, meaning that they require batteries to keep functioning. Our overall goal in MPMP is to find a routing scheme by configuring the coverage area of each terminal, such that a *source terminal* can send data to a set of *destination terminals*. While achieving this goal, in important fact to consider is that, as the coverage area of a terminal is increased, its power consumption increases exponentially with the distance. More power consumption means more cost spent on replacements of batteries, meaning that coverage areas return to us as costs in the end. Therefore, in MPMP, we have to find a routing scheme such that the total cost caused by the coverage areas is minimized (Wieselthier et al. [2000, 2001]; Leggieri et al. [2008]). The MPBP is a special case of MPMP, in which all the terminals except

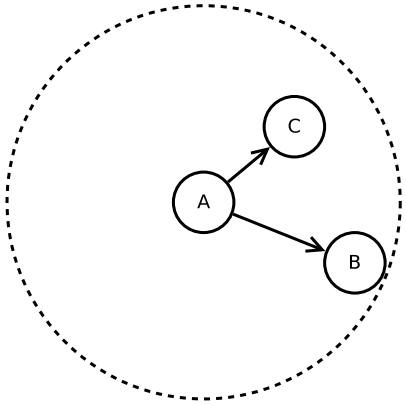


Figure 3.4. The wireless multicast advantage.
In this example, terminal A has extended its coverage area to be able to transmit to terminal B. At this point, terminal A can also transmit to terminal C in addition to terminal B, because terminal C has been included within the coverage area of terminal C.

the source terminal are destination terminals. In the rest of this chapter, we will collectively refer to MPMP and to MPBP as MPMP. For MPMP, various mathematical programming approaches and heuristic techniques were proposed. A survey can be found in Guo and Yang [2007].

In MPMP, an important characteristic that should be mentioned is the *wireless multicast advantage* (Wieselthier et al. [2002]). According to this characteristic, when a terminal A is linked to terminal B, it extends its coverage area enough to transmit to terminal B, and while doing this, it can also transmit to a terminal C, if the terminal C is within this coverage area. This characteristic is visualized in figure 3.4.

To sum up the high-level definition of MPMP, see the illustration in figure 3.5, where a tiny instance and two example solutions are visualized.

In the classical MPMP, the data of the problem is the power requirement for a terminal i to reach (i.e. to include in its coverage area) terminal j. In this study, previously discussed in Toklu and Montemanni [2011a,b, 2012b], we consider a variation of MPMP in which the power requirement values are subject to uncertainty. This uncertainty represents the errors made while measuring/calculating the distances between the terminals, unfriendly weather conditions and unknown obstacles affecting the quality of the transmissions. We call this variation of the problem as MPMP with uncertain power requirements (MPMPU). Note that, in MPMPU we study here, differently from classical robust optimiza-

tion studies, we make assumptions about probability distributions: we assume that each uncertain power requirement value belongs to an associated interval, in which the probability distribution is uniform. Because of these assumptions, MPMPU is actually closer to stochastic optimization field.

In our study, to handle MPMPU, we propose a 3-step heuristic approach based on the mathematical programming previously discussed in Montemanni and Mahdabi [2011]. In our approach, the following steps are taken:

- *Step 1*: An initial scenario is assumed, and a mathematical programming formulation is executed to find the optimal routing scheme according to the initial scenario.
- *Step 2*: The routing scheme found in step 1 is analyzed, and estimations are made about the parts of the routing scheme which makes the network most vulnerable to the uncertainty
- *Step 3*: Coverage areas of the critical parts (i.e. parts which are vulnerable to uncertainty) of the routing scheme are increased, and the coverage areas of the non-critical parts of the routing scheme are decreased, in such a way that the total power consumption of the network stays the same. In other words, the power consumption of the network is slightly shifted towards its critical parts.

3.2.1 Problem Definition

The MPMP can be defined in terms of a graph $G = (V,A)$, where V is the set of terminals, and $A = \{(i,j)|i,j \in V\}$ is the set of arcs. Within the set V, we have a source terminal s. The set of destination terminals is represented by $D \subseteq (V \backslash \{s\})$. Each arc $(i,j) \in A$ represents a connection between the terminals i and j. The power requirement value r_{ij} represents the power consumption needed by terminal $i \in V$ to establish the connection $(i,j) \in A$. The power requirement for a terminal to connect to itself is defined as 0 (i.e. $r_{ii} = 0$). The decisions we make in this problem are in terms of power consumptions, represented by ρ_i. As we increase ρ_i, we increase its coverage area. If $\rho_i \geq r_{ij}$, we say that the terminal i *reaches* terminal j (i.e. terminal i includes terminal j within its coverage area). Given these defintions, our objective is to minimize the total cost $\sum_{i \in V} \rho_i$, while making sure that each destination terminal $i \in D$ can receive data from the terminal s, directly or via the help of the intermediate routing terminals. It is important to note the following two points: (i) it is not necessary to reach the non-destination terminals (i.e. terminals excluded

from the set D), but reaching these non-destination terminals and using them as routers can prove to be useful; and (*ii*) both destination terminals and non-destination terminals can be used as routers.

In MPMPU, we say that we do not know the power requirement values exactly, so, we express the uncertain data by intervals. Therefore, we say: $r_{ij} \in [\underline{r}_{ij}; \overline{r}_{ij}] \quad \forall (i,j) \in A, i \neq j$, and for self-loop arcs we say: $r_{ii} = \underline{r}_{ii} = \overline{r}_{ii} = 0 \quad \forall i \in V$. Within the $[\underline{r}_{ij}; \overline{r}_{ij}]$ intervals, we assume that the probability distributions are uniform. We can now define a *scenario* z as a deterministic MPMP instance in which each power requirement value $r[z]_{ij}$ is randomly picked from the interval $[\underline{r}_{ij}; \overline{r}_{ij}]$. Because of the uncertainty, in MPMPU, we have to think about the reliability of the network. Given that we have a set of scenarios expressed by Z, we measure the reliability in two ways:

- *Total number of disconnections*: in a scenario $z \in Z$, the number of disconnections is the number of terminals which were not able to receive the data from the source because of the uncertainty. The total number of disconnections, is the number of disconnections summed over all the scenarios in Z.

- *Total number of flawed scenarios*: A flawed scenario is a scenario in which there is at least one disconnection. So, this measurement is found by the number of flawed scenarios in Z.

In MPMPU, in addition to the objective of minimizing the cost, we are also looking to increase the reliability of the network by minimizing the total number of flawed scenarios, and the total number of disconnections.

3.2.2 Mathematical Programming Formulation for the Deterministic MPMP

Let us now look at the formulation of the deterministic MPMP, without the consideration of the uncertainty, meaning that r_{ij} values are exactly known constant numbers.

First, we start by defining an array v^i for each terminal i. In v^i, all the terminals j are ordered non-decreasingly according to r_{ij} values. The k-th element of v^i is represented by v^i_k. The first element of v^i is always terminal i itself (i.e. $v^i_1 = i$). An example which illustrates the v^i arrays can be seen in figure 3.6.

We now define $x_{ik} \in \{0,1\}$ variables. If x_{ik} is set as 1, it is decided that a terminal i should extend its coverage area just enough to reach its k-th closest neighbour (i.e. v^i_k); otherwise x_{ik} becomes 0. To impose the flow of connectivity

from terminal i to terminal j, we also define a variable y_{ij} for each arc (i,j). By using these variables, we can formulate MPMP as follows:

$$
\text{MPMP}\begin{cases}
\text{minimize} & \displaystyle\sum_{i\in V}\sum_{k=1}^{|V|} r_{i,v_k^i} & & (3.22)\\
\text{subject to} & \displaystyle\sum_{k=1}^{|V|} x_{ik} = 1 & \forall i \in V & (3.23)\\
& \displaystyle(|V|-1)\sum_{k=k':v_{k'}^i=j}^{|V|} x_{ik} \geq y_{ij} & \forall (i,j)\in A & (3.24)\\
& \displaystyle\sum_{j\in V} y_{ji} - \sum_{j\in V} y_{ij} = \begin{cases} -|D| & \text{if } i = s \\ 1 & \text{if } i \in D \\ 0 & \text{otherwise} \end{cases} & \forall i \in V & (3.25)\\
& 0 \leq y_{ij} \leq |V| & \forall (i,j)\in A & (3.26)\\
& x_{ik} \in \{0,1\} & \forall i \in V,\ \forall k \in \{1,2,...,|V|\} & (3.27)
\end{cases}
$$

In the model of MPMP, the constraints (3.23) say that only one x_{ik} has to be set as 1 for each terminal i. The constraints (3.24) provide the connection between x_{ik} and y_{ij} variables. The constraints (3.25) make sure that the flow represented by the y_{ij} variables construct an arborescence starting at the source terminal s, and ending at the destination terminals. Finally, the domains of y_{ij} and x_{ik} variables, are specified by the constraints (3.26) and (3.27) respectively.

After getting the optimal result from the mathematical model of MPMP, one can easily find the power consumption decisions for terminals as:

$$\rho_i = \sum_{k=1}^{|V|} r_{i,v_i^k} x_{ik}^*$$

where x^* represent the x values of the optimal solution.

3.2.3 Approaches to handle the uncertainty within MPMPU

While MPMP can be solved to optimality by using the model presented in section 3.2.2, we need to change our approach for handling the uncertainty within MPMPU.

3.2.3.1 A straightforward approach

Let us first discuss a classical, straightforward approach. According to the straightforward approach, we depend on a protection parameter denoted by $\alpha \in [0;1]$. This α protection parameter can be seen as the conservativeness degree configuration mechanism of this straightforward approach. In this approach, we create a special scenario in which all the power requirements are equal to $\underline{r}_{ij}+(\overline{r}_{ij}-\underline{r}_{ij})$. With $\alpha = 0$, the coverage area requirements are assumed to be at their best-case values. As α value increases, the coverage area requirements are assumed to be at more pessimistic values. With $\alpha = 1$, we assume the worst-case scenario, and the model becomes equivalent to the Soyster approach discussed in section 2.2.1.

3.2.3.2 The 3-step approach

Now, we discuss our 3-step approach, which is an extension to the straightforward approach, and which has shown improved results in our experiments over randomly generated instances. Like the straightforward approach, the 3-step approach depends on the parameter α. In addition to α, we now define the parameter β, with the condition $0 \leq \beta \leq \alpha$. The parameter β represents the minimum protection that should be provided in the entire network.

Step one. The straightforward approach is executed, which provides the initial decisions about the coverage areas. The initial decision about the coverage area of a terminal $i \in V$ is expressed by $\hat{\rho}_i$. These initial decisions are subject to change in the next steps. Also, we define λ_i as the most distant terminal reached by the terminal i according to the special scenario of α. Therefore, we say:

$$\lambda_i = j \text{ such that } r^{\alpha}_{ij} = \hat{\rho}_i$$

Finally, the cost of the solution, which is to stay the same during the next steps, is defined as:

$$C = \sum_{i \in V} \hat{\rho}_i$$

Step two. In this step, we estimate which parts of the network are most vulnerable against the uncertainty.

Let us now define the concept of *dependency*. Under a scenario in which we assume the power requirement value for each $(i,j) \in A$ is $r^{\beta}_{ij} = \underline{r}_{ij} + \beta(\overline{r}_{ij} - \underline{r}_{ij})$, if a terminal k' receives data from terminal k, we say that terminal k' *depends* on terminal k. Also, if another terminal k'' receives data from terminal k', we

say that the terminal k'' *depends* on terminals k and k'. The visualization of the dependency concept can be seen in figure 3.7.

Now, we define the concept of *critical dependency*. Under the scenario in which we assume the power requirement value for each $(i,j) \in A$ is r_{ij}^{β}, if a terminal k' depends on a terminal k, but the connection (k,k') can not be established when we consider the worst-case scenario (in which the power requirement value becomes $\overline{r}_{ij}$ for each $(i,j) \in A$), we say that the terminal k' *critically depends* on terminal k. We also say that terminal k *weakly reaches* terminal k'. If, on the other hand, the connection (k,k') can be established even in the worst-case scenario, we say that the terminal k' *non-critically depends* on terminal k, and that the terminal k *strongly reaches* terminal k'.

Let us consider two terminals i and j, and formulate the query functions indicating whether the terminal i reaches terminal j strongly or weakly, as follows:

$$\textsc{StronglyReaches}(i,j) = \begin{cases} 1 & \text{if } \hat{\rho}_i \geq \overline{r}_{ij} \\ 0 & \text{otherwise} \end{cases}$$

$$\textsc{WeaklyReaches}(i,j) = \begin{cases} 1 & \text{if } \hat{\rho}_i \geq r_{ij}^{\beta} \\ 0 & \text{otherwise} \end{cases}$$

Now, let us define a function which indicates the quality of reaching from terminal i to terminal j. If a terminal i strongly reaches terminal j, then we say that the quality is 1, which is the maximum quality. If a terminal i reaches terminal j such that its power consumption ρ_i is just enough to satisfy the power requirement r_{ij}^{β}, we say that the quality is at minimum, therefore, 0. If terminal i can not reach terminal j, there is no quality to assign. The function which returns the quality of reaching from terminal i to terminal j is formulated as follows:

$$\textsc{Quality}(i,j) = \begin{cases} 1 & \text{if } \hat{\rho}_i \geq \overline{r}_{ij} \\ \dfrac{\left(\dfrac{\hat{\rho}_i - \underline{r}_{ij}}{\overline{r}_{ij} = \underline{r}_{ij}} \right) - \beta}{1 - \beta} & \text{if } \underline{r}_{ij} \leq \hat{\rho}_i < \overline{r}_{ij} \end{cases}$$

Now, we are ready to define the algorithm which estimates the criticalities of the terminals of the network. The algorithm we propose here depends on

a recursive procedure. This procedure starts by visiting the source terminal s. At each visited terminal, the procedure scans for all the terminals which can be strongly or weakly reached. At each visit, the procedure updates a list called $critical\_dependencies$. As the name suggests, this list stores the terminals on which the currently visited terminal critically depends. At each new visit, all the terminals within $critical\_dependencies$ receive criticality points. The terminals with more criticality points are heuristically identified to be the terminals which make the connectivity of the network more vulnerable against the uncertainty. The amount of increase in the criticality points is calculated according to the quality of reaching to the currently visited terminal. In more details, lesser quality means more criticality. At the end, the result of the execution of this algorithm is the criticality value I_i for each terminal i. The details of this procedure is given in algorithm 1.

Algorithm 1 The algorithm for estimating the criticalities of the terminals

initialize $critical\_dependencies = \langle\rangle$
initialize $I_i = 0 \quad \forall i \in V$
call EstimateCriticalities$(s, 1)$

function EstimateCriticalities(visited terminal i, quality q)
    mark terminal i as reached
    $score \leftarrow \begin{cases} |V| & \text{if } i \in D \\ 1 & \text{otherwise} \end{cases}$
    $I_d \leftarrow I_d + (score \cdot (1 - q)) \quad \forall d \in critical\_dependencies$
    for $\forall j \in V$ **do**
        if node j was not marked as reached already **then**
            if StronglyReaches(i, j) **then**
                call EstimateCriticalities$(j,$ Quality$(i, j))$
            else if WeaklyReaches(i, j) **then**
                push i into $critical\_dependencies$
                call EstimateCriticalities$(j,$ Quality$(i, j))$
                pop i from $critical\_dependencies$
            end if
        end if
    end for
end function

Scanning for other terminals for each visited terminal, this implementation

presented in algorithm 1 runs in $O(|V|^2)$ time. The criticality values, I_i, generated at this step, are to be used by the third step, in which the coverage areas of the terminals are adjusted.

Step three. In this step, the final decisions about the power consumption values (and therefore the coverage areas) are made. In more details, the power consumption values of the terminals with higher critical values are increased, and the power concumption values of the terminals with lower critical values are decreased, in such a way that the total cost of the network stays the same (the cost stays at C).

This step involves the execution of a linear programming model, formulated as follows:

$$\text{MPMPUS}_{\text{TEP}}3 \begin{cases} \text{maximize} & \sum_{i \in V} (I_i + 1) + \rho_i & (3.28) \\ \text{subject to} & r^{\beta}_{i,v^i_k} \leq \rho_i \leq \overline{r}_{i,\lambda_i} & \\ & \quad \forall i \in V, k \in \{1, 2, ..., l : v^i_l = \lambda_i\} & (3.29) \\ & \sum_{i \in V} \rho_i \leq C & (3.30) \\ & \rho_i \geq 0 \qquad \forall i \in V & (3.31) \end{cases}$$

In this model, the objective (3.28) imposes the maximization of the protections on the terminals against the uncertainty, with the priority given to the ones with the high criticality values. In constraints (3.29), the lower and upper bounds for the power consumption for each terminal i is specified. The lower bound here dictates that the protection on a terminal i can not go lower than the minimum protection imposed by the parameter β. The constraint (3.30) imposes that the total cost can not exceed C, the total cost obtained at the end of step 1. Finally, the constraint (3.31) specifies the domains of the ρ_i variables.

At the end of the execution of the model MPMPUS$_{\text{TEP}}$3, the adjusted power consumption values are obtained. Note that the model in this step is linear without any integer variables. This means that a Simplex (Dantzig [1963]) algorithm execution, which has polynomial-time execution requirement in average, will be enough to complete this step.

3.2.4 Experimental Results

Our 3-step algorithm was implemented in C++, by using Gurobi 4.5 library for solving the mixed integer linear programming models. To evaluate the 3-step approach, MPMPU instances were generated where the terminals are ran-

domly placed, and each generated instance was solved by the straightforward approach, and the 3-step approach. Then, the qualities of the solutions found by each approach on each instance were compared, solution quality being measured in terms of flawed scenarios and total number of disconnections, over 10000 randomly generated scenarios. In other words, a Monte Carlo sampling engine was used to generate scenarios and to test the qualities of the solutions. While we had used the sampling technique for generating scenarios, formulations for a more accurate estimations of the number of flawed scenarios can be found in section 3.2.5. For generating an MPMPU instance, the following procedure was applied:

- a 2-dimensional area with size 100×100 is prepared;
- for each terminal, a random-sized circle with a maximum radius R is generated, R being a parameter representing the size of uncertainty in the experiment. This circle is randomly located within the 100×100 area. This circle represents the area in which the terminal exists, its exact location within this circle being unknown. Therefore, these circles represent the uncertainty of MPMPU.
- given the definitions:
  - B: the base transmitting cost parameter ($B = 1$ in our study);
  - Q: the parameter representing the amount of exponential growth of power requirement values as the distances between the terminals grow ($Q = 4$ in our study);
  - MinEucDist(i, j): considering the location circles of the terminals i and j, this function returns the minimum euclidean distance between the terminals i and j;
  - MaxEucDist(i, j): the maximum euclidean distance between the terminals i and j;

  the following settings were prepared for each arc $(i, j) \in A$:

$$\underline{r}_{ij} = \begin{cases} 0 & \text{if } i = j \\ \left(\text{MinEucDist}(i, j)\right)^Q & \text{if } i \neq j \end{cases}$$

$$\overline{r}_{ij} = \begin{cases} 0 & \text{if } i = j \\ \left(\text{MaxEucDist}(i, j)\right)^Q & \text{if } i \neq j \end{cases}$$

Table 3.9. Experimental results on a randomly generated instance

Approach	Disconnections	Flawed Scenarios
Straightforward approach	36273	6475
3-step approach, $\beta = 0.5$	10983	7701
3-step approach, $\beta = 0.6$	9106	6743
3-step approach, $\beta = 0.7$	7192	5657
3-step approach, $\beta = 0.8$	5400	4499
3-step approach, $\beta = 0.85$	4390	3796
3-step approach, $\beta = 0.87$	5044	3921

- One of the terminals is declared as the source terminal, and the other $|D|$ number of terminals are picked as destination terminals.

Let us now analyze the solutions provided by both approaches according to $\alpha = 0.9$ on a single instance. The instance was generated by following the procedure above, with $R = 1$, with 20 terminals, 10 of them being destination terminals. The results are shown in table 3.9. Considering that the solutions in table 3.9 cost exactly the same, it can be seen that the number of disconnections is significantly decreased by the 3-step approach. While it is difficult to come up with a single value which will work best on all instances, from the table it can be concluded that a huge gap between α and β decreases the success of the 3-step approach. For this particular instance, the best working β value seems to be around 0.85.

By following the procedure for generating instances, multiple instances were generated with different number of terminals, different number of destination nodes and different values for R which controls the sizes of intervals of the uncertain data. Each instance was solved by both approaches with α=0.8, β=0.75; α=0.9, β=0.87; and α=0.95, β=0.93. The values for β were chosen according to the averaged success of the 3-step approach over all considered instances. Results of the experiments can be seen in tables 3.10, 3.11, 3.12, 3.13, representing experiments with $R = 1$, $R = 2$, $R = 5$, $R = 10$, respectively. In these tables, each horizontal area (group of three rows separated by horizontal lines) represents 20 randomly generated instances with the specified number of terminals ($|V|$) and destination terminals ($|D|$). Within each horizontal area, each row represents results obtained with different and values. The columns “Cost Save (%)” show how much, in percentage, the solution cost (which is the same for both the straightforward approach and the 3-step approach since the same α value is used for them on each experiment) was decreased in comparison to the cost of

Table 3.10. Experimental results on instances generated with $R = 1$

$\|V\|$, $\|D\|$	α, β	Cost Save (%)	Straightforward Approach D.	Straightforward Approach F.S.	3-Step Approach D.	3-Step Approach F.S.	Improvement (%) D.	Improvement (%) F.S.
10, 5	0.8, 0.75	11.21	20290.3	6816.35	12399.1	5968.4	36.50	12.08
	0.9, 0.87	5.60	11380.95	4172.5	6675.5	3502.6	38.36	15.29
	0.95, 0.93	2.80	6037.45	2319	3201.95	1884.45	42.43	17.04
10, 9	0.8, 0.75	11.00	24591.25	6171.4	12093.15	5430.45	39.99	10.86
	0.9, 0.87	5.50	13108.55	3650.9	6044.3	3049.8	41.27	4.41
	0.95, 0.93	2.75	6807.05	1970.2	2780.05	1625	44.80	15.08
20, 10	0.8, 0.75	11.42	51216	8680.7	33525.8	8324.75	35.12	4.43
	0.9, 0.87	5.71	30336.05	6121.65	18353.95	5642.9	41.16	8.38
	0.95, 0.93	2.86	16581.9	3676.75	9615.7	3379.8	44.27	8.62
20, 19	0.8, 0.75	11.36	96753.55	9055.85	56647.35	8659.75	41.56	4.40
	0.9, 0.87	5.68	57460.7	6748.35	33631.15	6328.2	41.28	6.17
	0.95, 0.93	2.84	31454.2	4228.25	18502.05	3927.65	42.34	7.67
30, 10	0.8, 0.75	12.01	55495.4	9372.2	37451.85	9184.1	33.16	2.03
	0.9, 0.87	5.99	33471.15	7292.45	21343.35	6981.1	37.63	4.28
	0.95, 0.93	2.99	18596.75	4698	11506.9	4525.25	40.15	3.54
30, 20	0.8, 0.75	11.79	117065.65	9753.9	67257.2	9572.95	42.03	1.88
	0.9, 0.87	5.89	73187.65	8206.05	35806.75	7731.8	49.79	5.83
	0.95, 0.93	2.95	41285.55	5640.55	17926.95	5077.2	55.00	9.88
30, 29	0.8, 0.75	11.70	177218.85	9782.2	110519.4	9554.75	36.63	2.33
	0.9, 0.87	5.85	111365.1	8278.1	58146.75	7955.15	46.29	3.79
	0.95, 0.93	2.92	62791.6	5697	29180.8	5426.55	52.93	4.45

the conservative (α=1) solution, calculated as $1 - \frac{Cost}{ConservativeCost}$. The column groups "Straightforward Approach" and "3-Step Approach" show how much in average the results have disconnections (column denoted by "D.") and flawed scenarios (column denoted by "F.S."). The column group "Improvement (%)" show how much in average are the numbers of disconnections and flawed scenarios are improved by the 3-step approach, in comparison to the straightforward approach. On an instance, the improvement as percentage is calculated as $1 - \frac{a}{b}$, a being the number of disconnections or flawed scenarios of the solution of the 3-step approach and b being the number of disconnections or flawed scenarios of the solution of the straightforward approach.

In the tables 3.10, 3.11, 3.12, and 3.13, it can be seen that there is an average improvement provided by the 3-step approach in the experiments done over each group of 20 randomly generated instances. Note that there were cases in which the number of flawed scenarios were better when the straightforward approach was used. For example, when we look at the experiment shown in table 3.11, with $\alpha = 0.8$ and $\beta = 0.75$, we can see that the average improvement provided in terms of flawed scenarios by the 3-step approach is only 0.42, which means within that set of 20 instances, there were cases where the straightfor-

Table 3.11. Experimental results on instances generated with $R = 2$

$\|V\|$, $\|D\|$	α, β	Cost Save (%)	Straightforward Approach D.	Straightforward Approach F.S.	3-Step Approach D.	3-Step Approach F.S.	Improvement (%) D.	Improvement (%) F.S.
10, 5	0.8, 0.75	11.85	15779.5	6173.25	10010.5	5630	31.04	9.14
	0.9, 0.87	5.92	8348.3	3568.25	5208.25	3222.3	30.33	9.42
	0.95, 0.93	2.96	4328.55	1917.05	2598.85	1733.65	32.17	8.87
10, 9	0.8, 0.75	12.14	27083.45	6476.75	15035.65	5720.75	36.05	11.23
	0.9, 0.87	6.07	14285.95	3812.8	7185.3	3215.35	40.59	14.80
	0.95, 0.93	3.03	7392.25	2060.15	3429.75	1745.2	43.90	14.47
20, 10	0.8, 0.75	12.89	44900.3	8525.35	27646.6	8102.9	35.11	5.01
	0.9, 0.87	6.45	25678.6	5909.7	13733.95	5365.3	41.60	9.30
	0.95, 0.93	3.22	13791.05	3535.15	6696.45	3132.1	47.94	12.51
20, 19	0.8, 0.75	12.66	81451.3	8950.85	39866.9	8217.1	48.55	8.51
	0.9, 0.87	6.33	46087.65	6451.65	20888.7	5726.9	51.03	12.02
	0.95, 0.93	3.16	24291.75	3931.05	10206.85	3438.15	53.76	13.23
30, 10	0.8, 0.75	13.42	53459.5	9211.25	33258.1	8825.7	37.18	4.36
	0.9, 0.87	6.72	31944.5	6984.7	18092.3	6445.75	41.97	8.21
	0.95, 0.93	3.36	17646.3	4430.9	8585.15	3982.2	48.55	10.21
30, 20	0.8, 0.75	13.18	108737.85	9646.55	70539.15	9602.2	35.21	0.45
	0.9, 0.87	6.59	64843.05	7798.35	39072.4	7677.9	40.06	1.44
	0.95, 0.93	3.29	36634.2	5199.85	18375.3	4815.5	53.48	7.94
30, 29	0.8, 0.75	13.47	151890.15	9695.3	91034.45	9653.5	38.36	0.42
	0.9, 0.87	6.73	89855	7958.95	43270.35	7615.15	50.57	4.24
	0.95, 0.93	3.36	49242	5294	18224.1	4817.2	60.72	8.77

Table 3.12. Experimental results on instances generated with $R = 5$

$\|V\|$, $\|D\|$	α, β	Cost Save (%)	Straightforward Approach D.	Straightforward Approach F.S.	3-Step Approach D.	3-Step Approach F.S.	Improvement (%) D.	Improvement (%) F.S.
10, 5	0.8, 0.75	15.46	16103.15	5520.45	9809.15	4781.3	35.54	12.99
	0.9, 0.87	7.72	8471.2	3134.45	4931.15	2618.95	36.87	15.81
	0.95, 0.93	3.86	4351.7	1664.95	2300.55	1321.75	40.93	18.52
10, 9	0.8, 0.75	14.94	27932.1	6120.05	11318.55	5062.65	54.76	17.28
	0.9, 0.87	7.47	14832.45	3569.05	5510.8	2792.75	57.56	21.76
	0.95, 0.93	3.74	7669.7	1939.3	2778.8	1504.75	57.66	22.19
20, 10	0.8, 0.75	16.02	40083.05	8382.8	23446.8	7978.5	39.17	4.81
	0.9, 0.87	8.00	21661.15	5631.55	11778.1	5124.55	42.31	8.70
	0.95, 0.93	4.00	11288.6	3290.05	5357.8	2746	49.42	15.60
20, 19	0.8, 0.75	16.03	72826.6	8865.3	44544.95	8756.35	36.42	1.18
	0.9, 0.87	8.02	39304.2	6249.4	20603.45	5993.2	44.57	3.97
	0.95, 0.93	4.01	20640.5	3724.45	9670.4	3571.05	48.76	3.75
30, 10	0.8, 0.75	17.00	50081.2	8828.4	36303.25	8702.95	28.65	1.60
	0.9, 0.87	8.50	28845.35	6264.2	18531.05	6061.5	37.48	3.81
	0.95, 0.93	4.25	15104.55	3703.55	7371.25	3401.65	50.39	8.60
30, 20	0.8, 0.75	16.12	103116.9	9376.9	61560.2	9168.25	39.99	2.29
	0.9, 0.87	8.05	61408.5	7263.85	34858.5	6983.45	43.64	4.19
	0.95, 0.93	4.03	33453.8	4645.15	17703	4516.75	46.37	3.29
30, 29	0.8, 0.75	16.77	129703.5	9521.8	66174.85	9387.3	47.19	1.38
	0.9, 0.87	8.38	70615.65	7422.15	30569.05	7075.95	53.39	4.44
	0.95, 0.93	4.19	36570.05	4720.85	11899.9	4352.3	64.67	7.29

Table 3.13. Experimental results on instances generated with $R = 10$

$\|V\|$, $\|D\|$	α, β	Cost Save (%)	Straightforward Approach D.	Straightforward Approach F.S.	3-Step Approach D.	3-Step Approach F.S.	Improvement (%) D.	Improvement (%) F.S.
10, 5	0.8, 0.75	16.85	13152	5357.4	8297.3	4992.85	34.09	6.65
	0.9, 0.87	8.41	6695.85	2927.15	4025.95	2659.3	36.23	9.03
	0.95, 0.93	4.20	3402.35	1521.3	1786.4	1327.9	39.83	10.78
10, 9	0.8, 0.75	17.17	23572	5906.2	13549.55	5414.05	33.10	7.77
	0.9, 0.87	8.57	12013.7	3313.95	5995.9	2847.95	40.05	12.68
	0.95, 0.93	4.29	6101.75	1740.85	2916.15	1506.05	39.75	10.67
20, 10	0.8, 0.75	18.25	34826.55	7501.8	19182.85	6761	45.53	10.40
	0.9, 0.87	9.11	17292	4544.5	8178.05	3892.85	50.06	14.75
	0.95, 0.93	4.56	8798.8	2457.35	3605.85	1979.45	56.76	20.23
20, 19	0.8, 0.75	17.68	60889.9	7908.5	33507.2	7717.4	43.77	2.46
	0.9, 0.87	8.82	32434.25	5059.2	16336.3	4795.2	48.98	5.21
	0.95, 0.93	4.41	16464.5	2809.7	6774.2	2582	55.24	7.76
30, 10	0.8, 0.75	16.77	129703.5	9521.8	66174.85	9387.3	47.19	1.38
	0.9, 0.87	9.15	22213.65	5284.55	11910.8	4766.2	46.15	9.77
	0.95, 0.93	4.57	11577.6	3005.4	5720.9	2643.4	48.90	11.10
30, 20	0.8, 0.75	18.33	40803.95	8045.45	24001.6	7556.55	40.66	5.97
	0.9, 0.87	9.22	47779.2	6219.95	24387.3	5997.3	47.03	3.26
	0.95, 0.93	4.61	24803.6	3675.8	10766.95	3459.75	56.01	5.57
30, 29	0.8, 0.75	18.45	89886.15	8911	52252.05	8822.55	41.39	0.92
	0.9, 0.87	9.13	53855.95	6388.95	24372.85	6147.4	50.31	4.11
	0.95, 0.93	4.56	28081.6	3780	11384.05	3640.4	56.70	5.02

ward approach performed better (which reflects as negative improvement provided by the 3-step approach, bringing its overall average improvement towards 0). To get the best the 3-step approach and to prevent negative improvements, one might use the fact that the 3-step approach actually provides 2 solutions: one solution at the end of step 1 (which is equivalent to the straightforward approach), and one solution at the end of step 3. Considering that, in this particular study on MPMPU, we assume that the probability distributions are known, a Monte Carlo sampling engine would quickly tell the decision maker which of the two solutions performs better. This way, the decision maker would always be provided with the better solution.

3.2.5 Calculation of the number of flawed scenarios in an MPMPU solution

During our studies on MPMPU, we used a Monte Carlo sampling engine to generate scenarios and test the qualities of the solutions in terms of flawed scenarios. Now, we give the formulations for estimating the number of flawed scenarios more accurately. Note that, however, these formulations will require a lot of execution time, therefore, a fast Monte Carlo sampling engine is usually much

more practical.

A solution, generated by the straightforward approach or the 3-step approach, is a set of transmission power values for all terminals. According to these transmission power values, for each arc (i, j), there is a probability to reach terminal j. In this section, we provide the formulations to calculate the probability to reach all destination terminals $d \in D$ from the source terminal s.

Let us first define the concept of *important arcs*. An arc is an *important arc* if it exists within a non-looping path which starts from the source terminal s and reaches at least one destination terminal. The set of all important arcs are denoted by A^*.

Since, the arcs are probabilistic, depending on the scenario, different important arcs can be enabled or disabled. Therefore, we define a *configuration* $K \subset A^*$ as a set of important arcs, which represents the assumption that each important arc $(i, j) \in K$ can reach terminal j and each important arc $(i', j') \in (A^* \backslash K)$ can not reach j'. Now we define a *connective configuration* as a configuration in which the arcs provide at least one path to each destination terminal $d \in D$, from the source terminal s.

To estimate the number of flawed scenarios, first, all important arcs are identified. This can be done by following all the paths from source terminal s to all destination terminals by using a recursive depth-first search.

After identifying all important arcs, each possible configuration $K \subset A^*$ is listed and its probability is calculated. For this purpose, we begin by formulating the function $\textsc{ProbArc}(i, j)$ which calculates the probability for arc (i, j) to reach terminal j:

$$\textsc{ProbArc}(i,j) = \begin{cases} 1 & \text{if } \rho_i \geq \overline{r}_{ij} \\ 0 & \text{if } \rho_i < \underline{r}_{ij} \\ \frac{\rho_i - \underline{r}_{ij}}{\overline{r}_{ij} - \underline{r}_{ij}} & \text{otherwise} \end{cases}$$

By using the function $\textsc{ProbArc}(i, j)$, now we can formulate the function to find the probability of a configuration as:

$$\textsc{ProbConfiguration}(K) = \prod_{(i,j)\in K} \textsc{ProbArc}(i,j) \cdot \prod_{(i,j)\in A^*\backslash K} (1 - \textsc{ProbArc}(i,j))$$

The probability of having all destinations reached, $\textsc{ProbFlawless}$, is then calculated as the sum of the probabilities of all connective configurations:

$$\textsc{ProbFlawless} = \sum_{K\subset A^*} \textsc{ProbConfiguration}(K) \tag{3.32}$$

The probability of having at least one destination terminal disconnected is PROBFLAWED $= 1 -$ PROBFLAWLESS. Among n trials of establishing connection, the number of flawed scenarios is finally estimated as $n \cdot$ PROBFLAWED.

Estimating the flawed scenarios via mathematical means, this approach can be used to verify the correctness of the sampling engine available. For this purpose, the number of flawed scenarios of the solutions of some randomly generated instances were estimated by both sampling engine and the formulations presented in this section. The results of these comparison are shown in 3.14. The instances were generated with $R = 1$, and solved by the 3-step approach with parameters $\alpha = 0.9$, $\beta = 0.87$. Within the table, under the column group "Estimated flawed scenarios", the column "Sampling" represents the estimations of the sampling engine, and the column "Formulations" represents the estimations of the approach presented in this section. The column "Errors" represents the error made by the sampling engine, in comparison to the estimations of the formulations, calculated as $(a - b)/10000$ where a is the result of the sampling engine, b is the result of the formulations and 10000 is the number of scenarios. In the table, it can be seen that the absolute values of the errors made by the sampling engine are always below 1%. Therefore, the results of the sampling engine can be verified as close to the formulations.

3.2.6 Summary

A minimum power multicasting/broadcasting problem was tackled, in which there is uncertainty on the power requirement for one terminal to reach another. A straightforward approach, and our proposed 3-step approach were discussed and compared. In the experiments, it was seen that the 3-step approach can provide improvements over the solutions of the straightforward approach, considering the generated instances in which the terminals are randomly placed.

In this study, while defining the uncertainty, we have assumed that the probability distributions are known. Therefore, MPMPU is actually a stochastic optimization problem, not a robust optimization problem. We can say that the relation of this study to our main research is in the method we use: the second step of the 3-step approach involves detecting the parts of a solution most vulnerable to the uncertainty, and patching the solution considering those vulnerable parts. Detection of the vulnerable parts of a solution is inspired by the Bertsimas-Sim approach (see section 2.2.5), which is used in our matheuristic robust optimization studies as well (which will be explained in chapter 5), where the most vulnerable coefficients used by a solution are found, and the evaluation of a solution is done assuming that those coefficients will be perturbed the most by the

Table 3.14. Flawed scenario estimations via the sampling engine and via the formulation 3.32.

			Estimated flawed scenarios		
Instance	Nodes	Destinations	Sampling	Formulations	Errors
1	10	5	2655	2671.01	-0.16%
2	10	5	1926	1944.24	-0.18%
3	10	5	2472	2499.67	-0.28%
4	10	5	1405	1430.72	-0.26%
5	10	5	1851	1838.57	0.12%
6	10	9	3432	3372	0.60%
7	10	9	1997	2028.36	-0.31%
8	10	9	5480	5445.85	0.34%
9	10	9	1026	1050.29	-0.24%
10	10	9	2347	2354.51	-0.08%
11	20	10	4407	4450.17	-0.43%
12	20	10	6710	6734.37	-0.24%
13	20	10	3245	3208.72	0.36%
14	20	10	1632	1600.92	0.31%
15	20	10	2836	2834.71	0.01%

uncertainty. In the 3-step approach, the terminals which are more critical (therefore vulnerable against the uncertainty) for keeping the network connected are heuristically estimated, so that the solution can be patched against the uncertainty. In conclusion, we can say that this study involves the application of a robust-optimization-inspired approach on a stochastic optimization problem.

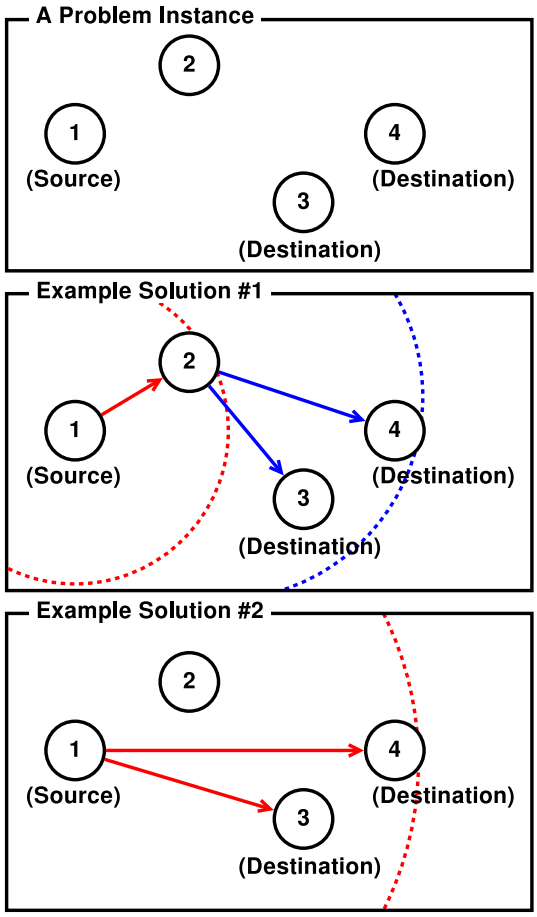


Figure 3.5. An example instance for MPMP, and two example solutions. In solution #1, the source terminal sends its data to the destination terminals 3 and 4 by using the terminal 2 as a router. The terminal 2 is using the wireless multicast advantage for routing the data it received to both destinations: it extends its coverage area enough to include terminal 4 in its coverage area. Terminal 3 also gets included within this coverage area. Therefore, the cost of this solution is $r_{1,2} + r_{2,4}$. In solution #2, the source terminal extends its coverage area enough to include all the terminals, so that it can transmit to the destination terminals. The cost of this second solution is $r_{1,4}$.

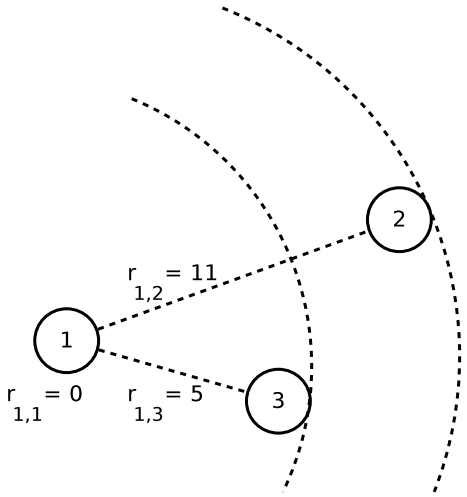


Figure 3.6. Visual explanation of the v^i arrays via a portion of an example MPMP instance.
When we sort all the possible targets for terminal 1 according to their power requirement values, we can see that $r_{1,1} \leq r_{1,3} \leq r_{1,2}$. Therefore, we say that $v^1 = \langle 1, 3, 2 \rangle$.

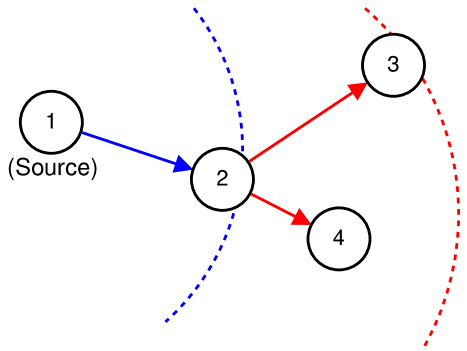


Figure 3.7. Visual explanation of the concept of dependency on a portion of a MPMP instance.
Here, we are considering the scenario in which the power requirement for each $(i, j) \in A$ is r_{ij}^{β}. It can be seen that the terminal 2 is receiving its data from terminal 1 (the source terminal). Therefore, we say that terminal 2 depends on terminal 1. Also, we see that terminals 3 and 4 receive their data from terminal 2. Considering that terminal 2 receives its data from terminal 1, we say that terminal 3 and terminal 4 depend on terminals 1 and 2.

Chapter 4

Matheuristic Studies

We now explain our matheuristic studies. First, we discuss our shared incumbent environment studies (Toklu et al. [2012]; Toklu and Montemanni [2012a]) for solving MPMP (which was previously defined in section 3.2); then we discuss our study for solving a large-scale energy management problem (Anghinolfi et al. [2011b, 2012]; Toklu, Montemanni and Gambardella [to appear]).

4.1 A Shared Incumbent Environment for MPMP

Let us consider again the minimum power multicasting problem (MPMP), for which the exact mixed integer linear programming model was given in section 3.2.1. Since the model contains integer decisions, solving it by using the exact model requires a lot of execution time and memory. In this study, previously discussed in Toklu et al. [2012]; Toklu and Montemanni [2012a], we are not focused on the uncertainty. Instead, we are focused on solving MPMP in a shorter amount of time. For this purpose, we use a technique called *shared incumbent environment* (SIE), which was first proposed in Mojana et al. [2011]. SIE is an approach where its two components, a metaheuristic solver and an exact mathematical programming solver, work in parallel on the same problem. As a hybridization of metaheuristic search with mathematical programming, shared incumbent environment can be classified as a matheuristic approach. The main idea of SIE is that, when one component improves the best known solution, it informs the other component. The other component then imports the new best known solution as its current solution. In other words, the two components share their incumbent solution. The overall method benefits from this sharing, in such a way that one component fixes the weakness of the other. In more details, a metaheuristic search is very good at finding practical solutions in early stages

of the optimization process. Thanks to the metaheuristic search, the mixed integer linear programming solver becomes aware of these practical solutions and realizes more quickly which regions of the solution space are non-promising. Pruning these regions out of consideration and focusing on more promising regions, the efficiency of the exact solver can be increased. On the other hand, metaheuristic search can get stuck in a local minimum. Since an exact solver searches local subtrees in the solution space thoroughly, it can detect a better solution which was missed by the metaheuristic search. After becoming aware of the better solution thanks to the exact solver, the metaheuristic search can become unstuck from its local minimum.

In this study, for solving MPMP, we use a SIE in which the metaheuristic component is a simulated annealing (see Kirkpatrick et al. [1983]) based on the study of Montemanni et al. [2005]. For the mixed integer linear programming component, we use the formulations discussed in section 3.2.1.

4.1.1 The Simulated Annealing Approach

Simulated annealing is a metaheuristic search algorithm which is inspired from the process of annealing in metallurgy (Kirkpatrick et al. [1983]), where the purpose is to change the internal configuration of a material for making it stronger. This process begins with heating the material, thus giving the atoms enough energy to become unstuck from their initial positions. Then, a slow cooling is applied, which slowly decreases the chances for the atoms to change to a worse configuration, turning the process into a controlled local search. In the algorithm, the temperature is represented by a variable t, which affects the chance of accepting a candidate solution vector with a worse quality than the current solution vector.

Simulated annealing for solving MPBP was proposed in Montemanni et al. [2005]. In this study, we adapt the simulated annealing to MPMP by using multicast incremental algorithm, instead of broadcast incremental algorithm, as the initial solution generator. The rest of the simulated annealing approach stays the same, except that the feasibility of a candidate solution is now determined by checking if the destination terminals (instead of all terminals except the source) are reached.

The detailed explanation of the simulated annealing process is as follows. The simulated annealing begins with multicast incremental power algorithm, which was proposed in Wieselthier et al. [2000]. This algorithm keeps a list for connected terminals and another list for terminals which are not connected yet. In the beginning, the only reached terminal is the source terminal. At each

iteration, the algorithm finds the cheapest (i.e. the one with the lowest power requirement value) connection (i, j), i being a connected terminal and j being a terminal which is not connected yet. To realize the connection (i, j), the coverage area of terminal i is increased enough to reach terminal j. Then, terminal j is moved to the list of connected terminals. These iterations go on until all terminals are connected. The algorithm then initiates a phase called sweeping. In this phase, some redundant connections are removed from multicast tree. The idea is that if, for example, terminal i reaches the terminal j and both terminals i and j reach terminal k, then terminal i actually does not have to reach k, so, the coverage area of terminal i can be decreased. The final phase of multicast incremental power algorithm is the removal of the paths which do not lead to any destination terminals. This whole approach can be implemented in $O(n^3)$ time and gives useful initial solutions for our simulated annealing implementation.

Simulated annealing is a process which keeps a current solution and generates candidate solutions in each iteration by modifying the current solution. A candidate solution is probabilistically accepted as the new current solution or ignored, where the probability of acceptance drops with the temperature. In this study, the modification on the current solution is implemented as follows:

- Randomly select a transmitting terminal i and decrease its coverage area so that it reaches its $(k-1)$-th waypoint, instead of its currently reached k-th waypoint.
- If all destination terminals are still connected to the network, the modification is complete. Otherwise, select a terminal $j \neq i$, and increase its coverage area so that all the destination terminals are connected again (i.e. the disconnectivity is fixed). The way of selecting terminal j is probabilistic. With a probability p_r, terminal j is selected in such a way that fixing the disconnectivity will have the least addition of transmission power; with a probability $1 - p_r$, terminal j is selected randomly.

Finally, the whole simulated annealing approach can be expressed as follows:

- *Step 1:* Execute multicast incremental power algorithm and label its generated solution as the current solution and the best known solution.
- *Step 2:* Initialize the simulated annealing search, set the temperature t to the value of the initial temperature parameter t_{init} and set the iteration counter to 1.
- *Step 3:* Modify the current solution. Name the modified version of the current solution as the candidate solution. If the candidate solution is better

than the current solution, label the candidate solution as the new current solution. If not, label it as the new current solution with a probability expressed by $e^{(\text{cost(candidate)}-\text{cost(current)})/t}$. Also, if the candidate solution is better than the best known solution, label the candidate solution as the new best known solution.

- *Step 4:* If, for the last c_t iterations, the best known solution is not improved, decrease the temperature by multiplying it with the cooling parameter α where $0 < \alpha < 1$. If the temperature is less than the minimum temperature t_{min}, finish the search; otherwise, increment the iteration counter and return to *Step 3*.

In previous experiments Montemanni et al. [2005], it was observed that the following parameter values are effective: $p_r = 0.2$, $t_{init} = 0.2$, $c_t = 30000$, $\alpha = 0.9$, $t_{min} = 0.1$.

4.1.2 The Shared Incumbent Environment Implementation

Mixed integer linear programming is a problem-independent approach which explores the solution space thorougly and can proceed until the optimum is found. The problem independency comes with the cost of large execution time requirements on large solution spaces. On the other hand, a metaheuristic search can be designed in a very problem-specific way. In more details, problem-specific heuristic operations can be performed iteratively in the metaheuristic search, so that a very fast convergence is obtained. Such metaheuristics, however, can result in early convergence: they can get stuck in a local optimum.

The shared incumbent environment is a hybrid approach which executes an exact mixed integer linear programming solver component and a metaheuristic search component in parallel. These two components work on a shared solution vector. Therefore, when a component improves the solution, the other will be aware of the improvement. The shared incumbent environment is based on the idea that the different natures of exact and metaheuristic search can help each other. For example, the metaheuristic search can be unstuck from a local optimum when the exact solver discovers a new one in the solution space. Also, the exact solver can be aware of many useful heuristic solutions thanks to the metaheuristic search component, so, it can realize more quickly which areas of the solution space are non-promising.

By using the components described in sections 3.2.1 and 4.1.1, it is possible come up with a shared incumbent environment approach.

The shared incumbent environment is implemented as a multi-threaded program. The first thread executes a mixed integer linear programming solver and the second thread executes the simulated annealing approach. A portion of memory, called the *shared memory*, which is accessible by both threads, is allocated for storing the shared incumbent solution. The thread which finds the first solution copies the solution to the shared memory. Each thread, when it improves its own current solution, checks the shared memory and if its new current solution is better than the shared incumbent solution, that new current solution is exported as the new shared incumbent solution. Also, each thread periodically checks the shared memory to see if the shared incumbent solution is better than the current solution of the thread. If so, the shared incumbent solution is imported into the thread as the new current solution.

In the simulated annealing thread, when the simulated annealing search finishes because it reaches the minimum temperature, the search is restarted with another random seed, with the maximum temperature.

When the time execution limit is reached, both threads end, and the best solution known so far is obtained as the final solution.

4.1.3 Experimental Results

In this section, we present our experiments for evaluating the mentioned three approaches (MILP – mixed integer linear programming approach, simulated annealing, and shared incumbent environment). In short, the experiments were conducted by randomly generating instances and evaluating the three approaches by executing them on each instance. Details on generation of the instances and evaluations of the approaches are presented in sections 4.1.3.1, 4.1.3.4 and 4.1.3.5. The comments on the obtained results are listed in section 4.1.3.6.

4.1.3.1 Generation of the instances

The procedure for generating an instance with $|V|$ number of terminals and $|D|$ number of destination terminals is as follows. All $|V|$ terminals are positioned into random coordinates in a two dimensional area. One of them is labeled as the source terminal and the other $|D|$ of them are labeled as destination terminals. For each pair of terminals (i, j) where $i \neq j$, the power requirement value r_{ij} is calculated as the euclidean distance between the two terminals to the power of 4. Also, for each terminal i, r_{ii} is set as 0.

4.1.3.2 Implementation details of the approaches

The three approaches, MILP, simulated annealing and shared incumbent environment, were implemented in C++ programming language. For the MILP approach, the solver was allowed to use two cores of the processor. The simulated annealing approach was implemented in such a way that two simulated annealing threads work in parallel and they inform each other when they improve the best known solution. The mathematical formulation used in MILP and shared incumbent environment approaches were solved by using IBM ILOG CPLEX 12.3 (IBM CPLEX [2014]).

4.1.3.3 Analysis on a single instance

According to the experiments, the most successful cases of shared incumbent environment are when the considered instance is too difficult for the MILP solver to quickly converge to a solution. In such cases, the shared incumbent environment is advantageous as it incorporates a metaheuristic which will make the MILP component reach to more desired lower and upper bounds under limited time.

As a representation of the usual behavior of the considered approaches on difficult instances, let us take the results on a single instance with 180 terminals where all terminals except the source terminal are destination terminals. The behavior of the three approaches are shown in figure 4.1. In the figure, it can be seen that the upper bound of MILP is still above 20 000 000 at the end. While simulated annealing converges very quickly to useful results, it stagnates. The shared incumbent environment keeps improving the solution cost while also having a higher lower bound.

4.1.3.4 Comparison of the approaches on broadcasting problems

Multiple instances with different numbers of terminals were generated. For each instance, all the terminals except the source terminal were declared as destination terminals. Therefore, these are MPBP problems. On each generated instance, each approach was executed with a time limit of 3 000 seconds, and the results were compared. All the experiments were done in a computer with Intel Core 2 Duo P9600 2.66GHz processor and with 4GB RAM.

In Toklu et al. [2012], the comparison results on the broadcasting instances were presented, shown in table 4.1. Further results on the broadcasting instances were discussed in Toklu and Montemanni [2012a], which are shown in table 4.2. In these tables, each row represents the average of results over 10

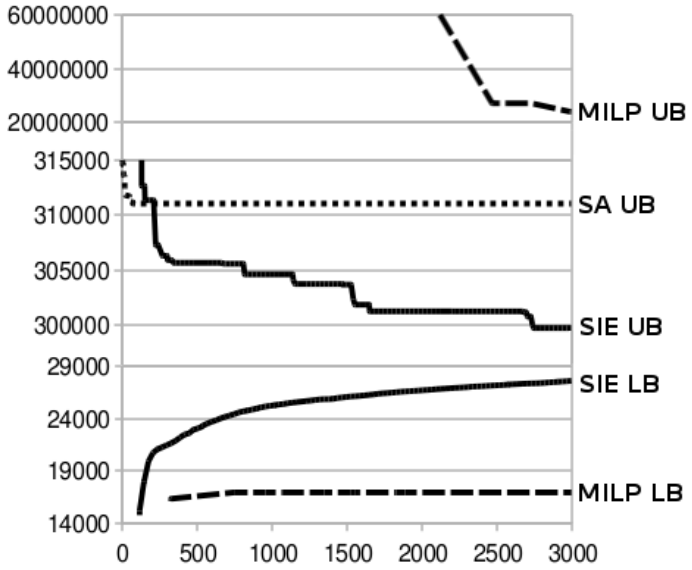


Figure 4.1. The behavior of the three approaches (MILP, simulated annealing (SA) and shared incumbent environment (SIE)) on a single example instance with $|V| = 180$ and $|D| = 179$. The lower and upper bounds are titled as "LB" and "UB", respectively.

instances generated with the specified number of terminals (reported under the column "$|V|$"). The most bottom row represents the average of results over all considered instances. The column groups "Improvement over SA" and "Improvement over MILP" show how much the shared incumbent environment was able to improve in comparison to the simulated annealing approach and to the MILP approach, respectively. The comparisons are done in terms of lower bounds and upper bounds (i.e. solution costs) and they are reported as percentages. For lower bounds, the improvements were calculated as $(a/b) - 1$ where a is the lower bound reached the shared incumbent environment and b is the lower bound reached by the MILP approach. For upper bounds, the improvements were calculated as $1 - (a/b)$ where a is the upper bound reached the shared incumbent environment and b is the upper bound reached by the specified approach. The numbers of instances where the shared incumbent environment was able to improve are given in parentheses, next to the improvement percentages.

4.1.3.5 Comparison of the approaches on multicasting problems

Multiple instances with different numbers of terminals and different numbers of destinations were generated. On each generated instance, each approach was

Table 4.1. Comparisons of shared incumbent environment to other approaches on broadcasting instances (Toklu et al. [2012])

	Improvement over SA	Improvement over MILP	
$\|V\|$	Lower bound	Upper bound	Upper bound
130	3.32 (10/10)	-79.23 (0/10)	-6.65 (0/10)
140	11.43 (10/10)	-78.02 (0/10)	-0.86 (1/10)
145	7.34 (9/10)	4.77 (6/10)	68.77 (7/10)
150	9.86 (10/10)	35.35 (7/10)	80.12 (9/10)
160	14.75 (10/10)	25.56 (7/10)	63.95 (9/10)
170	26.50 (10/10)	72.29 (9/10)	89.20 (9/10)
180	2.89 (10/10)	36.95 (9/10)	89.40 (9/10)
190	10.85 (10/10)	24.95 (9/10)	89.72 (9/10)
200	14.90 (9/10)	21.92 (9/10)	89.67 (9/10)
Average	*11.32 (88/90)*	*7.17 (56/90)*	*62.59 (62/90)*

Table 4.2. Comparisons of shared incumbent environment to other approaches on broadcasting instances (Toklu and Montemanni [2012a])

	Improvement over SA	Improvement over MILP	
$\|V\|$	Upper bound	Lower bound	Upper bound
160	22.40 (10/10)	-78.24 (0/10)	-28.93 (4/10)
170	26.50 (10/10)	72.29 (9/10)	89.20 (9/10)
180	2.89 (10/10)	36.95 (9/10)	89.40 (9/10)
190	10.85 (10/10)	24.95 (9/10)	89.72 (9/10)
200	6.16 (10/10)	34.61 (9/10)	90.19 (10/10)
Average	*13.76 (50/50)*	*11.00 (36/50)*	*65.92 (41/50)*

executed for 3 000 seconds and their results were compared. Instances with number of terminals 160 and 200 were solved in a computer with Intel Core 2 Duo P9600 2.66GHz processor and with 4GB RAM. Instances with number of terminals 240 and 280 were solved in a computer with Intel Core i5 2.3GHz processor and with 4GB RAM. The comparisons are shown in table 4.3 (note that, for consistency within table 4.3, some results in table 4.2 are repeated). In the table, each row represents the average of results on 10 instances generated according to the specified number of terminals (reported under the column "$|V|$") and number of destinations (reported under the column "$|D|$"). For each group of rows with the same number of terminals, there is a row, titled "Average" which shows the results averaged over all instances in that group. The most bottom row in the table represents the average of results over all generated instances. In table 4.3, like in table 4.2, the column groups "Improvement over SA" and "Improvement over MILP" show how much the shared incumbent environment was able to improve in comparison to the simulated annealing approach and to the MILP approach, respectively. The comparisons are reported in terms of percentage improvements of lower bounds and upper bounds. The numbers of instances where the shared incumbent environment provided improvements are given in parentheses.

4.1.3.6 Comments on the results

From the results obtained on broadcasting instances, which are reported in table 4.2, it can be observed that the success of the shared incumbent environment in comparison to the MILP approach depends heavily on the sizes of the instances. In other words, it can be seen that, as the number of terminals increase, the percentage improvements made over the solutions of the MILP increase. The implication behind such results is that the exchange of information between a metaheuristic and a MILP solver is useful in converging to low-cost results on big instances. In addition to the improvements in terms of solution costs, especially in broadcasting instances, improvements in terms of lower bounds can be observed. This implies that the MILP solver was able to prune the non-promising areas of the solution space more efficiently when it was supported by a metaheuristic search component.

On the results obtained from the multicasting instances, which are reported in table 4.3, the importance of the instance size can again be observed. Like in the broadcasting results, as the number of terminals increase, the percentage improvements made over the solutions of the MILP increase. In addition, with the same number of terminals considered, and with the exception of $|V| = 160$, the

Table 4.3. Comparison of the shared incumbent environment to the other approaches

		Improvement over SA	Improvement over MILP	
$\|V\|$	$\|D\|$	Upper Bound	Lower Bound	Upper Bound
160	40	11.78 (10/10)	-71.30 (0/10)	21.74 (3/10)
160	80	23.85 (10/10)	-81.51 (0/10)	8.47 (2/10)
160	120	12.72 (10/10)	-37.72 (1/10)	29.17 (3/10)
160	159	22.40 (10/10)	-78.24 (0/10)	-28.93 (4/10)
Average		*17.69 (40/40)*	*-67.19 (1/40)*	*7.61 (12/40)*
200	50	26.86 (10/10)	164.96 (7/10)	69.61 (7/10)
200	100	13.89 (10/10)	25.64 (6/10)	63.00 (8/10)
200	150	16.06 (10/10)	-10.41 (6/10)	54.29 (8/10)
200	199	6.16 (10/10)	34.61 (9/10)	90.19 (10/10)
Average		*15.74 (40/40)*	*53.70 (28/40)*	*69.27 (33/40)*
240	60	17.35 (10/10)	23.98 (6/10)	48.72 (7/10)
240	120	19.13 (10/10)	-18.00 (5/10)	47.23 (5/10)
240	180	29.45 (10/10)	27.89 (8/10)	68.67 (9/10)
240	239	31.72 (9/10)	41.20 (10/10)	95.12 (10/10)
Average		*24.41 (39/40)*	*18.77 (29/40)*	*64.94 (31/40)*
280	70	17.76 (10/10)	49.62 (4/10)	38.45 (8/10)
280	140	27.38 (10/10)	-7.09 (5/10)	54.03 (9/10)
280	210	27.95 (10/10)	-36.77 (4/10)	31.27 (7/10)
280	279	35.54 (10/10)	20.97 (7/10)	68.21 (7/10)
Average		*27.16 (40/40)*	*6.68 (20/40)*	*47.99 (31/40)*
Overall		*21.25 (159/160)*	*2.99 (78/160)*	*47.45 (107/160)*

average percentage improvements seem to be the highest when $|D| = |V|-1$ (especially with $|V| = 200$ and $|V| = 240$, both average percentages are the highest and the number of improved instances are 10 on broadcasting instances). The reason behind this could be that, some instances generated with $|D| < (|V| - 1)$ are actually equivalent to smaller problem instances when their the outermost non-destination terminals are removed. As the MILP solver is more efficient on simpler instances (as also implied by table 4.2 and table 4.3 for instances with small number of terminals) the MILP solver could be more successful on instances which are simplified because of this kind of equivalency, resulting in a decrease of solution quality improvement of shared incumbent environment in comparison to MILP on non-broadcasting instances.

Finally, both table 4.2 and table 4.3 show that the shared incumbent environment was more successful than simulated annealing in most of the cases. This implies that the shared incumbent environment is a competitive heuristic.

4.1.4 Summary

A shared incumbent environment, which involves solution sharing between a simulated annealing solver, and a mixed integer linear programming solver, Within limited execution time, it was seen that, in general, over the considered instances, the shared incumbent environment was able to improve over simulated annealing and mixed integer linear programming approaches which work independently.

Although in this study, the focus is on increasing the heuristic solution qualities, and the uncertainty is not considered, the solution sharing mechanism used here becomes an important component of our final matheuristic framework too, within the robust multiple ant colony system, which will be explained in chapter 5.

4.2 Large-Scale Energy Management Problem

In this section, we discuss our studies (Anghinolfi et al. [2011b, 2012]; Toklu, Montemanni and Gambardella [to appear]) on a large-scale energy management problem (LSEMP). This problem is related to our studies both in the sense that uncertainty is considered, and that matheuristic techniques are used.

This work is a joint effort between IDSIA (L.M. Gambardella, R. Montemanni, N.E. Toklu), and DIST (Dipartimento di Informatica, Sistemistica e Telematica, University of Genoa; D. Anghinolfi, C. Nattero, M. Paolucci). The DIST team de-

veloped mathematical-programming-based modules (heuristic outage schedule generator, weekly production planner, MIP outage scheduling model, timestep production planner). The work contributed by N.E. Toklu is the local search module, which contains a metaheuristic search, communicating with the mathematical programming-based module. Involving a metaheuristic search with embedded exact methods for solving an optimization problem under uncertainty, this work is closely related to the matheuristic robust optimization framework we ultimately propose in chapter 6. Both the mathematical-programming-based modules and the metaheuristic local search module are explained in details in section 4.2.3.

4.2.1 Introduction

As the requirement for electrical energy is growing rapidly, optimizing and coordinating the operations of power plants in an economical way to satisfy energy demand has become a crucial practical issue. Maintenance scheduling and production planning of power plants are important for making sure that available human resources and material resources for maintenance are used in reasonable amounts and the equipment involved are kept in perfect efficiency, while satisfying the customer demands (Porcheron et al. [2010]).

Studies have been made on power plant maintenance scheduling, by using mixed integer programming (Dopazo and Merrill [1975]; Ahmad and Kothari [2000]) and also by using metaheuristic methods, including genetic algorithms (Gil et al. [2003]), and ant colony optimization (Foong et al. [2008]).

In LSEMP, two types of power plants, based on different power generation technologies, are used to satisfy the power demands. Power plants of type 1 are those that can be refueled while still operating. Power plants of type 2 are those that need to be shut down from time to time to be refueled and maintained (these are typically nuclear power plants). Overall, the problem considered is to optimize both the maintenance scheduling for the nuclear power plants (taking into account different resources and technical constraints) and the production planning for all the power plants (again, taking into account technical constraints), with the aim of minimizing overall costs while fulfilling energy demands. Demands are affected by uncertainty in the model considered.

Since the LSEMP considered is a very large and complex problem, the approach proposed here involves the decomposition of the problem into smaller subproblems, giving the opportunity of working separately on the maintenance scheduling for nuclear plants and on the production planning of all the plants. The method used for this decomposition approach is a combination of mixed

integer linear programming for production planning, and of a simulated annealing (Kirkpatrick et al. [1983]) metaheuristics (embedding again a mixed integer linear programming core) for maintenance schedule.

4.2.2 Problem definition

The problem studied in here is the one used within the ROADEF/EURO 2010 Challenge (Porcheron et al. [2010]). The data to decide on this problem are the refueling amounts and production plans of all power plants, and also the scheduling of the maintenance shutdown (*outage*) weeks for the power plants of the second type. Multiple scenarios, modeling alternative customer demands and therefore representing the uncertainty, are also considered in the problem: a feasible solution is able to satisfy all possible scenarios. Because we have to satisfy all the constraints in all scenarios, the problem can be classified as a robust optimization problem. The objective function is defined as the cost averaged over various scenarios (we can say that this is where LSEMP differs from a classical robust optimization problem, because, with a pessimistic robust optimization perspective, the highest cost encountered among all the scenarios would be considered).

The problem includes constraints (numbered as 1 to 21), which effect the production plans of power plants and scheduling the outages of type 2 plants. These constraints will be formally defined in the reminder of this section, together with the other elements used to describe each problem instance. We refer the reader to Porcheron et al. [2010] for the official description of the problem.

Main concepts. The definition of the LSEMP is based on the basic concepts defined in the reminder of this section.
A *timestep* is the most elementary time unit of the problem. The index variable for timestep is t. The first timestep is $t = 0$ and the last timestep is $t = T - 1$.
Some events depend on *weeks*, where each week contains an instance-dependent number of timesteps. The index variable for weeks is h. The first week is $h = 0$ and the last week is $h = H - 1$.
Type 1 (T1) power plants can be refueled during production. The index variable for these plants is j. The first T1 plant is $j = 0$ and the last is $j = J - 1$.
Type 2 (T2) power plants cannot be refueled during production. The index variable for these plants is i. The first T2 plant is $i = 0$ and the last is $i = I - 1$.
Each T2 plant has $K + 1$ *cycles* (indexed as k, ranging between $k = -1$ – starting cycle – and $k = K - 1$), which are based on weeks and represent the production/outage alternation. More precisely, within a cycle of T2 plant i, the follow-

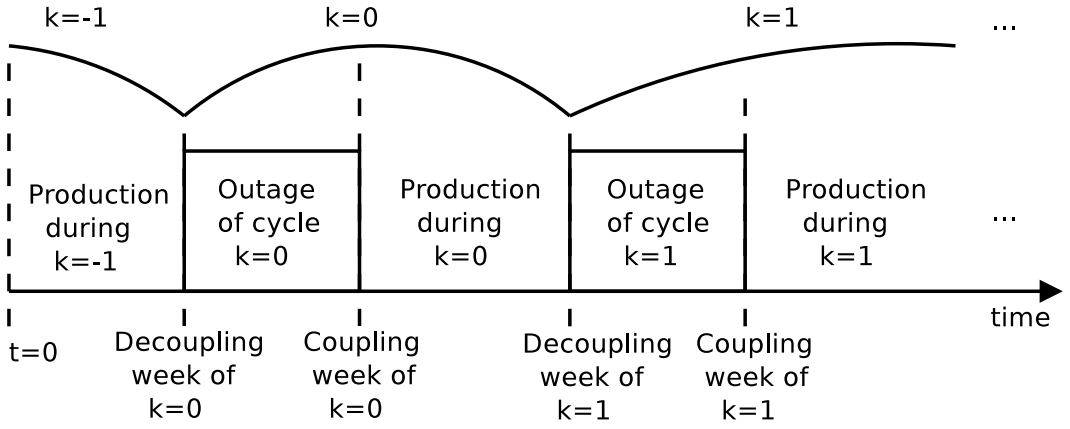


Figure 4.2. Sequence of cycles within a T2 power plant.

ing elements are defined: the *outage weeks* are those weeks in which a T2 plant is shut off to be refueled; the first outage week is referred to as the *decoupling week*; a set of consecutive weeks in which a T2 plant is active and producing is called *production campaign*. The *coupling week* is the first week of a production campaign. The sequence of cycles of a T2 plant is summarized is figure 4.2. Note that outage weeks are to be decided during the optimization.
Each instance has multiple *scenarios*. Each scenario provide alternative values for customer demands and different production limits for T1 plants. The index variable for the scenarios is s. The first scenario is $s = 0$ and the last is $s = S - 1$.

Decision Variables. The LSEMP has the following decision variables.
$ha(i,k)$: decoupling week of T2 plant i at cycle k.
$p(j,t,s)$: production of T1 plant j at timestep t in scenario s.
$p(i,t,s)$: production of T2 plant i at timestep t in scenario s.
$r(i,k)$: amount of refueling during the outage k of T2 plant i.
The following variable can be inferred by the previous ones, but it is useful to define them explicitly.
$x(i,t,s)$: stock level of T2 plant i at timestep t in scenario s.
$ec(i,k)$: timesteps of the production campaign of cycle k for T2 plant i.
$ea(i,k)$: outage weeks of cycle k for T2 plant i.

Constraints. The LSEMP considers the following 21 constraints.
[CT1]: For each scenario s the total production at each timestep t must be equal to the given demand $DEM^{t,s}$.
[CT2]: The production of each T1 plant must be between predefined time and

scenario dependent given boundaries $PMIN_j^{t,s}$ and $PMAX_j^{t,s}$.
[CT3]: Production of a T2 plant must be zero, when it is on outage.
[CT4]: The production of a T2 plant cannot be negative.
[CT5]: During each production campaign k of a T2 plant i, if the fuel level is greater than or equal to a given threshold $BO_{i,k}$, the production of that plant must cannot be more than a given threshold $PMAX_i^t$.
[CT6]: During the production campaign of a T2 plant i, if the fuel level is less than the given threshold $BO_{i,k}$, the production of that plant must follow a given profile $PB_{i,k}$, with a given tolerance ϵ.
[CT7]: The stock refilled during outage k of T2 plant i must be between given boundaries $RMIN_{i,k}$ and $RMAX_{i,k}$.
[CT8]: Fuel stock at $t = 0$ must equal a given initial fuel stock XI_i.
[CT9]: The fuel stock $x(i,t,s)$ of T2 plant i evolves according to production $p(i,t,s)$ and to the given lengthD^t of timestep t.
[CT10]: The fuel stock of T2 plant i right after outage k is defined according to some given refueling coefficients $Q_{i,k}$.
[CT11]: The fuel stock levels before and after the refueling are bounded by the given threshold $AMAX_{i,k}$ and $SMAX_{i,k}$.
[CT12]: Modulation (oscillation) in the power produced by T2 plants causes a wear of the equipment. Modulation is therefore constrained by two given parameters $PMAX_i^t$ and $MMAX_{i,k}$.
The remaining constraints are specific to T2 plants outage scheduling.
[CT13]: Outage k of T2 plant i must start between given boundaries $TO_{i,k}$ and $TA_{i,k}$. Moreover, outages cannot overlap: $DA(i,k)$ is the given length of outage k for plant i.
[CT14]: The entire outage periods of T2 plants in a given set A_m must be scheduled at least Se_m weeks apart from each other.
[CT15]: This constraint is same with [CT14], except that a specific time interval $[ID_m; IF_m]$ is given, and the constraint only applies within the given time interval.
[CT16]: All decoupling weeks of all T2 plants within the given set A_m must be scheduled at least Se_m weeks apart from each other.
[CT17]: All coupling weeks of all T2 plants within the given set A_m must be scheduled at least Se_m weeks apart from each other.
[CT18]: All coupling weeks of all T2 plants within the given set A_m must be scheduled at least Se_m weeks apart from all decoupling weeks of the plants in A_m.
[CT19]: The outages of T2 plants require a number of regional-limited shared human resources. The number of simultaneous outages in a same region is

therefore constrained: A_m is a set of T2 plants; $L_{i,k,m} \in [0, DA_{i,k}[$ is a time interval (in weeks) indicating how long after the beginning of the outage k the shared resources is required; $TU_{i,k,m}$ specifies the number of weeks the shared resource is required during outage k; Q_m finally defines the available quantity for the regional-limited shared resource.
[CT20]: It is forbidden to have too many simultaneous T2 plants' outages. The following instance-dependent parameters are required: $A_m(h)$ is the set of T2 plants. The set depends on the week index h; $N_m(h)$ is the maximum number of simultaneous outages allowed in week h.
[CT21]: The power capacity of the T2 plants on outage in a same region, in a given time period, has to be below a given threshold. The following instance-dependent parameters are required: C_m is a set of T2 plants; IT_m is a time period in weeks; $IMAX_m$ is the threshold for total offline power capacity in a week.

Objective Function. The objective of the problem is to minimize the sum of:
(i) the cost of production for T1 plants averaged over all scenarios
(ii) the total cost of refueling for T2 plants, reduced by the scenario-averaged cost of the remaining fuel at the end of the time horizon.

4.2.3 The MATHDEC approach

LSEMP is a difficult problem which requires both discrete decisions like outage dates and continuous decisions like production plans and refueling/stock values.

Considering the complex nature of the problem and the huge values of the parameters encountered in real instances, the problem can be classified, as a matter of facts, as a very difficult problem to solve as a whole. The method we propose, MATHDEC, decomposes the problem into smaller and easier-to-solve subproblems. They cover, namely, the scheduling of the outages, a week-based production planning and a more detailed timestep-based production planning.

To deal with the different subproblems, different modules have been developed. The overall architecture of the MATHDEC approach is depicted in figure 4.3, where the interactions among the different modules are shown. The remainder of this section is devoted to the description of these modules.

4.2.3.1 The Heuristic Outage Schedule Generator module

The Heuristic Outage Schedule Generator (HOSG) module is an iterative heuristic approach to generate outage scheduling for T2 plants. It does not consider timesteps, but only weeks and does not take care of production/refueling issues.

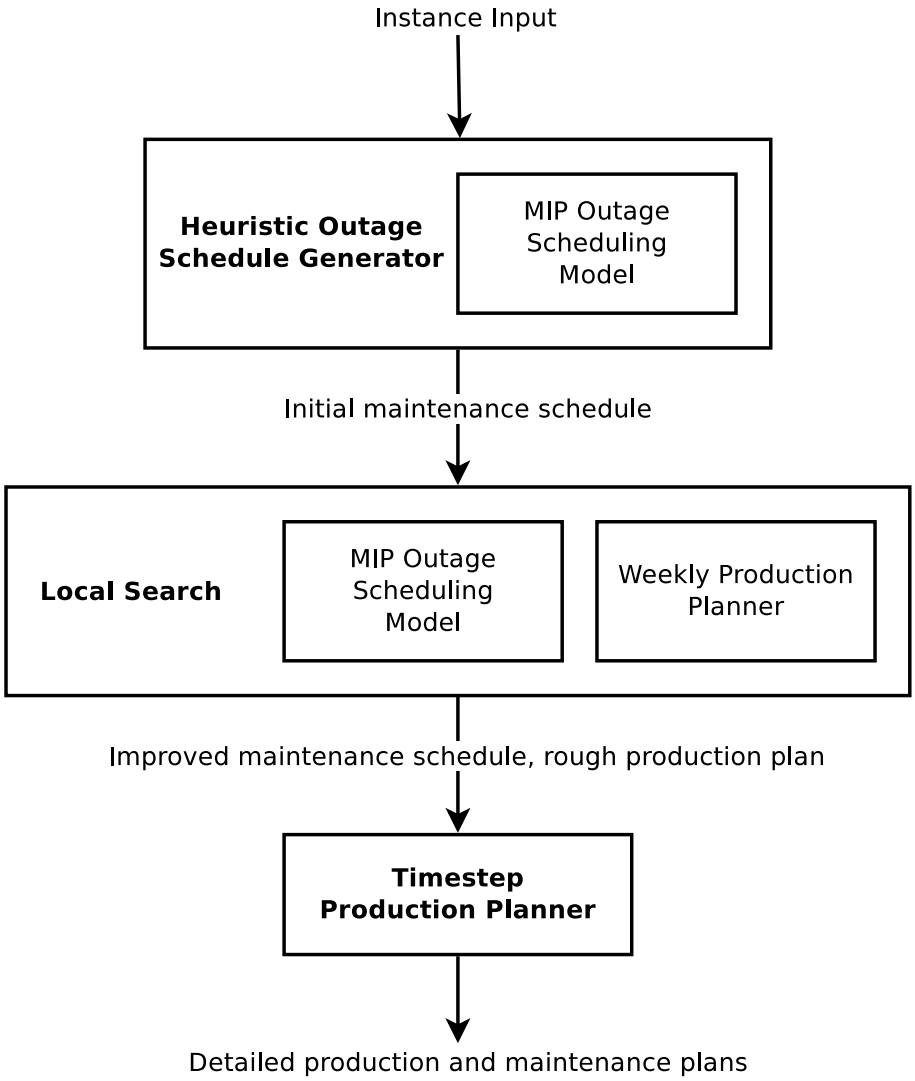


Figure 4.3. The overall architecture of the MATHDEC approach. In this figure, the rectangles represent modules and the arrows represent the data passed between these modules.

The main idea of HOSG is to have as many active cycles as possible (i.e. make $ha(i,k) \neq -1$ for as many (i,k) possible), so that the work load on T1 plants will be less: it has been experimentally found that high quality solutions have this characteristic.

HOSG iteratively produces outage schedules and calls the *MIP outage scheduler* submodule, which is in charge of checking if a scheduling is feasible or not according to constraints [CT13] to [CT21]. It is based on a mixed integer programming model. A detailed definition of the model can be found in the appendix section A.1.

In its first stage, HOSG executes MIP Outage Scheduler with all week intervals containing -1, which means that it gives the disabling option for all cycles of each T2 plant. After receiving a feasible schedule, HOSG initiates its second stage.

In the second stage, HOSG initializes $k' = 1$, which represents a cycle. Iteratively increasing k', HOSG executes MIP Outage Scheduler, with intervals up to k' excluding -1 and the other intervals including -1. This means, cycles up to k' must be executed, they do not have the disabling option (i.e. $ha(i,k) \neq -1 \mid k \leq k'$). When a k' is reached where a failure signal is received from MIP Outage Scheduler, HOSG initiates its final stage.

The final stage is another iterative process which takes the last feasible schedule received as a base. At each iteration, HOSG randomly picks a small subset of the T2 plants and increases their number of mandatory cycles (i.e. number of cycles k with $ha(i,k) \neq -1$). Then, MIP Outage Scheduler is executed. If a feasible schedule is received, that schedule becomes the new base schedule for the next iteration. This loop is exited after a given computation time (between 20 and 30 minutes in our implementation) is elapsed.

The output of this module is a feasible outage schedule given by $ha(i,k)$ values, typically with the number of mandatory cycles increased with respect to the initial one.

4.2.4 The Local Search module

The Local Search (LS) module starts from the solution provided by HOSG, and tries to improve it. LS still works on weeks only, but now production and refueling are taken into account, although in a relaxed way: they are aggregated per weeks (instead of timesteps). A simulated annealing algorithm is implemented where random variation to outage weeks are introduced at each iteration. A solution vector, for each T2 plant i, stores K decoupling week values ($ha(i,k)$). Each $ha(i,k)$ is extended into a time interval $[ha(i,k) - \delta, ha(i,k) + \delta]$, where

δ is a radius value monotonically decreased during the computation. In our implementation, the temperature T is in $[0.1; 1.0]$. The search starts with $T = 1.0$ and at each iteration T is reduced by multiplying it by a cooling factor of 0.999. When $T \leq 0.1$ the search is restarted with $T = 1.0$. The set of time intervals obtained is sent to the MIP outage scheduler submodule (see 4.2.3.1 and the appendix section A.1 for more details) that return a feasible schedule fulfilling the given time intervals if it exists, or a negative answer otherwise. In case a feasible schedule is found, it is passed to the *Weekly Production Planner* submodule (WPP), which returns weekly production plans together with an estimation of the objective function of the problem obtained by aggregating timesteps into weeks. More in details, the WPP submodule receives an outage schedule $ha(i,k)$ and determines a rough approximation of the plant production, refueling and consequent costs. In particular, WPP solves a linear programming model, which explicitly considers all the scenarios but takes planning decisions still on a weekly base, and generates feasible refueling values. A detailed description of the linear program is available in the appendix section A.2.

4.2.4.1 The Timestep Production Planner module

The Timestep Production Planner (TSPP) module implements the final stage for solving the problem. The input of this module is an outage schedule (as $ha(i,k)$ values) generated by HOSG and improved by LS, and modulation reference values mod_{iks} and weekly refueling values generated by WPP.

The main idea of TSPP is to generate all production and refueling values expanded into timesteps, with deviations from the reference modulation values minimized.

At first, the weekly refueling plan is expanded into timesteps and adjusted to satisfy [CT7] and [CT11]. Then, for each scenario, a linear programming model is executed iteratively to generate the final solution. A complete description of the linear program can be found in ithe appendix section A.3.

The linear programming model considers [CT1] to [CT5], as other constraints were already handled by other modules. The objective here is to minimize the total deviations from the reference modulation given by WPP, and also the average refueling cost over all scenarios.

For each scenario, a loop is executed, in which this linear programming formulation is iteratively used. At the end of each iteration, the program adjusts the mod_{iks} values if the deviation values are greater than 0. The program also checks [CT7] and [CT11]. If they are not satisfied, it adjusts the fuel stock values. When the deviation values are 0 and [CT7] and [CT11] are satisfied, the

loop reaches its end.

Finally, the module gives the timestep-based and scenario-aware production, fuel stock and refueling values, which, together with the outage schedule generated previously, form a solution for the problem instance.

The module is started 12 minutes before the end of the available computation time, in order to safely have a feasible solution before the deadline.

4.2.5 Experimental results

The algorithm MATHDEC, described in section 4.2.3, has been implemented in C++, and ILOG CPLEX 11.0 IBM CPLEX [2014] has been used to solve mixed integer linear programs.

In table 4.4 the results obtained by MATHDEC on the 10 realistic benchmark instances *Data Instances of ROADEF/EURO 2010 Challenge* [2010] used within the ROADEF/EURO 2010 Challenge Porcheron et al. [2010] are compared with those obtained by the 18 teams that took part into the competition *Results of the ROADEF/EURO 2010 Challenge* [2010]. Among the instances considered, the following maximum values were encountered: number of timesteps 5817; number of weeks 277; number of T1 plants 27; number of type T2 plants 56; number of campaigns 6 and number of scenarios 121. For each instance, the best and the average solution costs obtained within the competition (1 hour on a Intel Xeon 5420 2.5GHz/8GB computer) are reported together with those obtained by the MATHDEC approach (1 hour on a less performing but comparable Intel Core 2 Duo 2.4GHz/3GB computer). The averages for each column are reported in the last line of the table.

Table 4.4 suggests that the method MATHDEC obtains results comparable with those of the methods presented at the ROADEF/EURO 2010 Challenge: MATHDEC is never able to improve best challenge results, but finds better results than the average ones found during the challenge for all but 2 instances. MATHDEC is also able to retrieve results of the same order of magnitude of the best challenge results for 8 instances out of the 10 considered (interestingly enough, the 2 problematic instances do not coincide with those for which MATHDEC is worse than the average results). The method presented is therefore fairly robust: the best solutions of the competition were found by different methods, each one typically "specialized" on some of the instances. Notice that the method MATHDEC depends on heuristic choices, and multiple runs on the same instance provide results deviating from those reported in table 4.4 by relatively factors in the order of 1.00E+8. This does not affect our general considerations.

Table 4.4. Results on the ROADEF/EURO 2010 Challenge instances

Instance	Best	Average	MATHDEC
B6	8.34E+10	8.79E+10	9.11E+10
B7	8.12E+10	9.23E+10	8.63E+10
B8	8.19E+10	5.85E+11	3.19E+11
B9	8.18E+10	6.84E+11	2.50E+11
B10	7.78E+10	9.45E+10	8.52E+10
X11	7.91E+10	8.67E+10	8.43E+10
X12	7.76E+10	8.45E+10	8.33E+10
X13	7.64E+10	8.16E+10	8.37E+10
X14	7.62E+10	8.97E+10	8.49E+10
X15	7.51E+10	9.73E+10	8.02E+10
Averages	*7.90E+10*	*1.98E+11*	*1.25E+11*

4.2.6 Summary

A very large-scale real energy management problem has been described, and a new matheuristic optimization approach to solve it has been discussed. The method decomposes the initial problem into smaller subproblems, that are solved by mathematical programming and metaheuristic tools. The benefit brought by the decomposition have been shown empirically, through experimental results on realistic instances.

This study is a milestone towards our matheuristic robust optimization framework, as it shows that large robust optimization problems can be heuristically solved by using matheuristic techniques.

Chapter 5

Matheuristic Robust Optimization Studies On Vehicle Routing

We now explain our matheuristic robust optimization methodology (Toklu et al. [2013a,b]; Toklu, Gambardella and Montemanni [to appear]; Toklu et al. [2014]; Toklu, Montemanni and Gambardella [to appear]) for solving the well-known vehicle routing problem (VRP). The section 5.1.1 gives a basic definition of the capacitated vehicle routing problem (CVRP), and then the capacitated vehicle routing problem with time window constraints (CVRPTW).

In the literature, among the significant robust optimization studies on CVRP are Lee et al. [2012] and Agra et al. [2013], where the authors used exact methods inspired by the Bertsimas-Sim approach. In Lee et al. [2012], a CVRP with uncertain travel times and deadline constraints are studied. The experiments were done over instances with up to 100 customers. In Agra et al. [2013], a CVRP with uncertain travel times/costs with time window constraints was considered. The experiments were done over instances with up to 50 customers. These studies did not present results over larger instances, most probably because of the execution time and/or memory issues. In our matheuristic robust optimization studies, although it is very unlikely to obtain optimal solutions because of the heuristic nature of our approaches, it is possible to obtain solution pools (not just single solutions), over larger instances, in shorter amount of time. In our experiments, we have obtained solution pools over instances with up to 200 customers.

In section 5.1, we make the basic definitions. In more details, we define the capacitated vehicle routing problem (CVRP), our target problem; and then we define the ant colony system (ACS) metaheuristic, an important component of our study. After finishing the basic definitions, we discuss our studies in section

5.2.

5.1 Basic Definitions

5.1.1 Capacitated Vehicle Routing Problem

The VRP is a combinatorial optimization problem faced frequently in the domain of logistics (Dantzig and Ramser [1959]; Laporte [1992]; Toth and Vigo [2001]; Baldacci et al. [2008]; Golden et al. [2008]; Baldacci et al. [2010]). In VRP, there is a central location, called the *depot*, where the goods are stored and there are customer locations. The goods stored in the depot are to be delivered to all customer locations in an optimal visiting order by the vehicles. At different customer locations, different amounts of the goods are demanded.

In this study, we consider *capacitated* VRP (CVRP). The term *capacitated* means that the vehicles are limited in terms of how much demand they can handle in this problem.

CVRP can be expressed on a graph $G = (L, A)$ where L is the set of locations (including the depot) and A is the set of arcs. An arc is a pair of locations (i, j), where $i, j \in L$. In the finite set L, the locations are stored as indices in this manner: $L = \{0, 1, 2, ...\}$, where 0 represents the depot and nonzero integers represent the customer locations. For each customer location, there is a positive amount of demand (i.e. $d_i > 0 \,|\, i \in (L \backslash \{0\})$). For the depot, the demand is 0 (i.e. $d_0 = 0$). We have a set of vehicles V. Each vehicle has the same storage capacity, shown as Q. A CVRP solution consists of routes for all vehicles, a route being a vector of integers representing the locations to be visited in the specified order. According to a solution x, the route of a vehicle $v \in V$ is shown as $x[v]$, the length of the route $x[v]$ is shown as $|x[v]|$, the k-th location to be visited in route $x[v]$ is shown $x[v, k]$. Considering that for each arc $(i, j) \in A$ there is a different travel cost c_{ij}, the optimality is defined by the solution which has the minimum total travel cost. In terms of a solution x, the constraints of CVRP can be explained as follows. The tour of a vehicle v begins and ends at the depot:

$$x[v, 1] = x[v, |x[v]|\,] = 0 \quad \forall v \in V \tag{5.1}$$

Other than the depot, a vehicle v must visit locations of valid customers:

$$x[v, k] \in (L \backslash \{0\}) \quad \forall v \in V;\, k \in \{2, 3, ..., |x[v]| - 1\} \tag{5.2}$$

A customer can be visited by a vehicle v only once:

$$x[v,k] \neq x[v,k'] \quad \forall v \in V;\ k,k' \in \{2,3,...,|x[v]|-1\};\ k \neq k' \tag{5.3}$$

Two different vehicles v and v' can not visit the same customer:

$$x[v,k] \neq x[v',k'] \quad \forall v,v' \in V;\ v \neq v';\ k \in \{2,3,...,|x[v]|-1\};\ k' \in \{2,3,...,|x[v']|-1\} \tag{5.4}$$

A vehicle v must not try to handle a route with a total demand more than the vehicle capacity Q:

$$\sum_{k \in \{2,3,...,|x[v]|-1\}} d_{x[v,k]} \leq Q \quad \forall v \in V \tag{5.5}$$

In CVRP without the consideration of uncertainty, the solution cost of x, which is to be minimized, can be defined as:

$$\textsc{TravelCost}(x) = \sum_{v \in V} \sum_{k=2}^{|x[v]|} c_{x[v,k-1],x[v,k]} \tag{5.6}$$

We can now sum up CVRP as:

$$\text{CVRP} \begin{cases} \text{minimize} & \textsc{TravelCost}(x) \\ \text{subject to} & (5.1), (5.2), (5.3), (5.4), (5.5) \end{cases}$$

An illustration of deterministic CVRP can be seen in figure 5.1.

CVRP with time window constraints. Let us now define the CVRP with time window constraints (CVRPTW). In CVRPTW, for each arc (i,j), we have two kinds of associated data: c_{ij} which represents the travel cost like in CVRP; and t_{ij} which represents the travel time. Also, in CVRPTW, each location i has these additional associated data: time window beginning T_i^B, time window ending T_i^E, and the service time T_i^S. According to CVRPTW, each customer at location i has to be served within the time window $[T_i^B; T_i^E]$. If a vehicle arrives at the location i before T_i^B, it has to wait until T_i^B, and then it can start serving the customer. If a vehicle arrives at the location i later than T_i^E, the solution is declared as infeasible. At each location i, the vehicle has to wait for T_i^S amount of time for unloading its load and serving the customer at that location. The time window

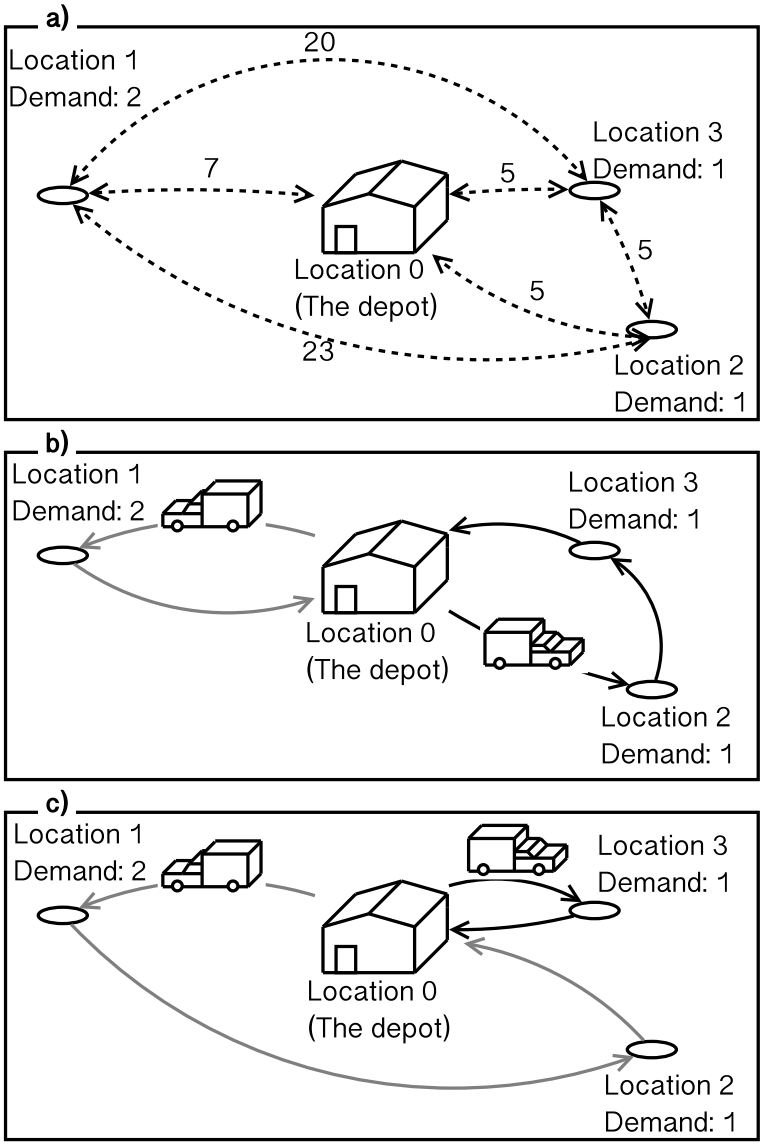


Figure 5.1. a) An example instance for CVRP. b) A practical solution which has a total travel cost of 29. c) A less practical solution which has a total travel cost of 45.

for the depot is $[0; EndTime]$, where $EndTime$ represents the ending time of the working day. Finally, the service time for the depot is $T_0^S = 0$.

Let us now express the time window constraints. For this purpose, we call the time in which a vehicle v starts serving the customer at the k-th destination on its route, the appearence time, denoted by A_k^v. At the beginning, each vehicle appears at the depot at time 0:

$$A_1^v = 0 \quad \forall v \in V \tag{5.7}$$

The appearance time of vehicle v to its k-th destination is equal to the time window beginning of its k-th destination, or its appearance to its $(k-1)$-th destination plus the service time at its $(k-1)$-th destination plus the time required to travel to its k-th destination, whichever has the higher value:

$$A_k^v = \max(T^B_{x[v,k]}; A^v_{k-1} + T^S_{x[v,k-1]} + t_{x[v,k-1],x[v,k]}) \\ \forall v \in V;\ k \in \{2, 3, ..., |x[v]|\} \tag{5.8}$$

A vehicle v must not appear to any location on its route later than that location's time window ending:

$$A_k^v \leq T^E_{x[v,k]} \quad \forall v \in V;\ k \in \{1, 2, ..., |x[v]|\} \tag{5.9}$$

Given these time window constraints, we can now summarize the CVRPTW as:

$$\text{CVRPTW} \begin{cases} \text{minimize} & \textsc{TravelCost}(x) \\ \text{subject to} & (5.1), (5.2), (5.3), (5.4), (5.5), (5.7), (5.8), (5.9) \end{cases}$$

5.1.2 Ant Colony System Metaheuristic

Ant colony optimization (ACO; see Dorigo et al. [1991]; Dorigo [1992]) is a name given to a class of metaheuristic optimization algorithms. The main idea of ACO is to simulate how the ants in nature find efficient routes from their nests to the food source. The inspiring behaviour of the ants is as follows. When an ant finds a food source, it marks the path leading to it with pheromones. Over time, various ants find various paths leading to the food source. However, since going back and forth on the shorter paths is easier, pheromones on the shorter paths will be reinforced frequently, and the pheromones on the longer paths will evaporate in time. In the end, it can be observed that most of the ants will gather around the shortest known path. In ACO, artificial ants "walk" on the solution space of a combinatorial optimization problem, adding a decision to the solution

vector at each step. When the solution is complete, it is evaluated according to the objective function of the problem at hand. Then, the decisions made by the ants are marked by pheromones, the quantity of the pheromones depending on how good the solution is. In the case of a transportation problem like the traveling salesman problem or the VRP, the arcs contained in a solution are marked. In the next iterations, when artificial ants walk, they become influenced by the pheromones, and their decisions are biased towards the pheromoned options.

In this study, we use an ACO variation called *ant colony system* (ACS), presented in Gambardella et al. [1999]. In Gambardella et al. [1999] what is presented is actually an improved ACS named *multiple ant colony system* (MACS). The MACS algorithm activates two ant colonies, one responsible for minimizing the number of vehicles needed by the solution, and one responsible for minimizing the total travel cost of the solution with minimum number of vehicle usage. When the minimum number of vehicles are calculated easily at the beginning (as in the case of CVRP without time window constraints), the MACS approach becomes equivalent to the regular ACS. In the rest of this chapter, we will use the term *ant colony system* (ACS) to collectively refer to the original ACS and its extension MACS.

The important characteristics of ACS are:

- Each iteration of ACS, a new generation of ants walks, instead of a single ant. This means, multiple solutions are constructed in one iteration.
- Only the ants that have found the best solution known so far is allowed to attract other ants by putting pheromones.

We now look at the technical details of the ACS. The algorithm can be explained as follows:

- *Step 1:* An initial solution x^{init} is generated by using a fast heuristic. In the case of transportations problem like CVRP, this fast heuristic is usually nearest neighbourhood heuristic (NNH; see Johnson and McGeoch [1997] for details) which is going to be explained in details in section 5.1.2.1.
- *Step 2:* Another solution storage field, x^{best}, is defined to keep the best solution found by the algorithm. Initially, x^{best} is set as the initial solution x^{init}.
- *Step 3:* A new generation of ants is activated. Each generation of ants have a fixed number of artificial ants. Each ant within this generation walks and constructs a solution. When the construction of a solution is

finished, that solution's quality is determined by the objective function. If an ant comes up with a solution x^{new} which has a higher quality than x^{best}, then the variable x^{best} is set as the solution x^{new} (i.e. x^{new} is declared as the new x^{best}).

- *Step 4:* Artificial pheromones are put on the arcs of x^{best} to attract the ants of the next generations towards the best solution known so far.

- *Step 5:* If the finishing criterion is not met, then we return to step 3. Otherwise, the algorithm stops.

Now, let us have a detailed look at how an artificial ant generates a solution during its walk. Especially this part of ACS is usually specific to the problem at hand. Therefore, in the remaining part of this section, we will consider the construction of CVRP solutions. The procedure followed by an ant for constructing a solution is as follows:

- *Step 1:* We begin by considering the first vehicle. The depot (i.e. the location indexed as 0) is added to the solution as the first visited location by the first vehicle.

- *Step 2:* Let us call a location *visitable*, if it is still not visited (i.e. if it is not yet added to our solution by the artificial ant), and if adding this location to the solution does not violate the capacity constraint of the currently considered vehicle. Also, if the problem at hand has time window constraints, we say that a location is visitable if adding it to the solution does not violate the time window constraints of that location. As long as there are visitable locations, the artificial ant keeps adding new locations to the solution. When there is no more visitable customer location, the artificial ant adds the depot to the solution. Adding the depot to the solution means that the tour of the current vehicle is finished. If, according to the solution, there are customer locations that are still not visited, a new vehicle is considered, and the step 2 is repeated.

We now take a detailed look at the implementational details of the pheromones. For this, we now make the following definitions:

- τ_{ij}: Amount of pheromones on the arc $(i, j) \in A$. Larger values for τ_{ij} mean larger attraction towards the arc (i, j).

- W: The set of ants. At each generation this set is renewed, but the number of ants, $|W|$, is fixed.

- N_w: The set of locations visitable by the ant $w \in W$. At each step of the ant w, this set is updated.
- HEURDIST(i, j): A heuristic expression of the distance between the locations $i \in L$ and $j \in L$.
- η_{ij}: The heuristic closeness value, calculated as $1/\text{HEURDIST}(i, j)$.
- β: This is a parameter which specifies the importance of the closeness factor in the decision process of an ant.
- α: This is a parameter within the interval $[0; 1]$. This parameter configures the probability for an ant to exploration, or exploitation. In short, exploration means to probabilistically make a decision; and exploitation means to deterministically choose an arc which maximizes the value returned by a certain formulation.

Let us now consider an ant $w \in W$, which is currently at location i (i.e. which has added the location i to its solution in its most recent move) and now has to decide its next location j. The ant w makes its choice as follows. With probability α, the ant goes for exploitation: a location j is chosen such that the value of $\tau_{ij} \cdot (\eta_{ij})^\beta$ is maximized. Otherwise, with probability $1 - \alpha$, the ant goes for exploration: a location j is chosen probabilistically. The probability for an exploring ant w at location i to pick the location j is formulated as follows:

$$p_{ij}^w = \begin{cases} \left(\tau_{ij} \cdot (\eta)^\beta\right) / \left(\sum_{j' \in N_w} \tau_{ij'} \cdot (\eta_{ij'})^\beta\right) & \text{if } j \in N_w \\ 0 & \text{otherwise} \end{cases}$$

At the beginning of the optimization process, all the arcs are given this same amount of pheromone:

$$\tau_0 = 1/(|L| \cdot \text{OBJFUNC}(sol\_init))$$

where OBJFUNC(x) is the objective function of the CVRP variation that we are solving. For the regular CVRP, this would correspond to the function TRAVELCOST(x) which was defined in (5.6).

The ACS, during its execution, does two types of pheromone updates, local and global:

- *Local pheromone update:* when an ant adds the arc (i, j) into its solution, to decrease the interest of the other ants of that generation on the same

path (so that they will be more interested in trying different paths), the pheromone amount τ_{ij} on that arc is slightly decreased, by using the following formulation:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho\cdot\tau_0$$

where $\rho \in [0;1]$ is the parameter which configures the amount of decrease in pheromones within the procedure of the local pheromone update.

- *Global pheromone update:* At each generation, to attract the ants of the next generations towards a high-quality path, the pheromone amount τ_{ij} on each arc (i,j) of the best known solution x^{best} is updated as according to the following formulation:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \rho/\textsc{ObjFunc}(x^{best})$$

At the end of each ant walk, a generated solution is improved by using a local search heuristic called *3-opt*. The details about this heuristic are given in section 5.1.2.1.

5.1.2.1 Heuristic operators used by the ant colony system

Here, we explain the heuristic operators used by the ant colony system: nearest neighbourhood search and 3-opt. Both these operators work on *giant tours*, a giant tour meaning a vector which stores all the tours of all the vehicles. For example, let us assume that we have two vehicles, and we have a solution where the first vehicle visits the customers 1 and 2, and the second vehicle visits the customers 3, 4, and 5. The giant tour representation of this solution would be $\langle 0,1,2,0,3,4,5,0\rangle$ (where 0s represent the beginning and ending of a solution, and also they act as separators between the tours of different vehicles).

Nearest neighbourhood heuristic. Nearest neighbourhood heuristic is a popular constructive algorithm in the literature. In the case of our studies, it adds a new location into the giant tour at each step. In short, the algorithm can be explained as follows:

- *Step 1:* The giant tour is initialized as $\langle 0\rangle$. Also, the current location is initialized as 0. We start considering the first vehicle.
- *Step 2:* Among the customer locations which are not visited yet, the nearest visitable neighour (nearest meaning the location which has the

least cost to travel to from the current location; visitable meaning that visiting that location does not violate capacity and time window constraints), is chosen as the target location. If the current vehicle has enough capacity to visit the target location, the target location is added into the solution. We then declare the target location as our new current location. If none of the remaining customers are visitable, we add 0 to the solution and our current location becomes 0 again, meaning that the current vehicle has finished its tour and that we consider a new vehicle now.

- *Step 3:* If there are customer locations which are not visited yet, we go to step 2. Otherwise, we finally add the last 0 into the solution, finish the execution of this algorithm.

In the end, we have constructed a giant tour, which represents an initial solution for the ACS.

For more details about nearest neighbourhood search, one can see Johnson and McGeoch [1997].

3-opt local search. The 3-opt local search is another popular heuristic in the literature. Again, in the case of our studies, it is applied on the giant tours.

The main idea is as follows. First, the initial giant tour is modified by removing three arcs from it, these three arcs having been selected randomly. The modified tour we have obtained is now a disconnected one. This disconnected tour is then reconnected in multiple ways, giving us alternative modified connected tours. If the best one among these reconnected tours is better than the original tour, then that reconnected tour is chosen as the result of this 3-opt operation; otherwise, the result is the original tour. The disconnection and reconnection operations of the 3-opt algorithm are visualized in figure 5.2.

For more details about 3-opt, one can see Bock [1958], Lin [1965], Johnson and McGeoch [1997]. A simpler variation of this method, 2-opt, can be found in Croes [1958].

5.2 Our Studies for Solving CVRP with Uncertain Data

Here, we discuss our studies on CVRP with uncertain travel costs (CVRPU), and CVRP with time window constraints with uncertain travel times (CVRPTWU), in sections 5.2.1 and 5.2.2, respectively. In these studies, we present extended versions of the ACS implementation reported in Gambardella et al. [1999]. In the end, general comments about these studies are made in section 5.2.3.

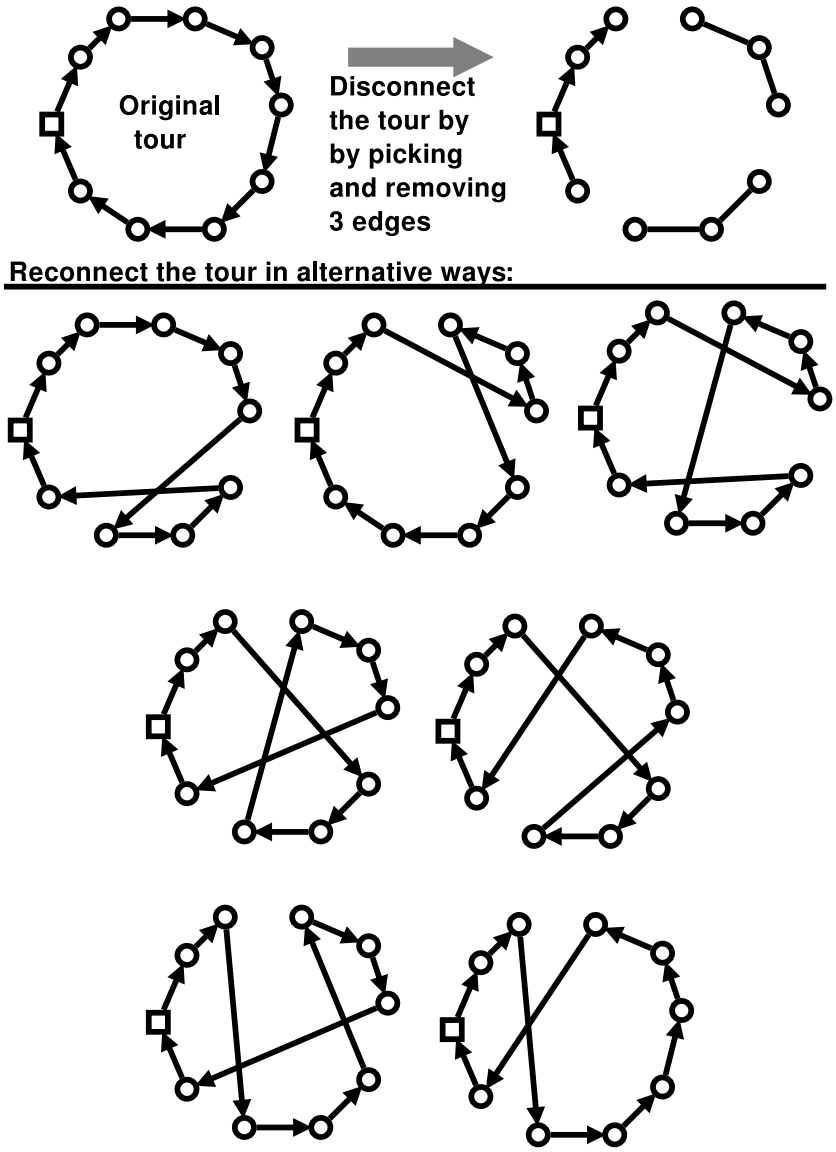


Figure 5.2. The disconnection and reconnection operations of 3-opt, explained on a simple, single-vehicle tour example.

5.2.1 The Capacitated Vehicle Routing Problem with Uncertain Travel Costs

Let us now consider the CVRPU. According to different perspectives of robust optimization, different methods can be taken to represent the travel cost uncertainty in CVRP. In our studies, we express the uncertain travel costs as intervals. Therefore, in CVRPU, for each arc $(i,j) \in A$, we say $c_{ij} \in [\underline{c}_{ij}; \overline{c}_{ij}]$. This way of modeling the CVRP with uncertain travel costs was previously done in the literature in Lee et al. [2012]; Agra et al. [2013].

The objective function (5.6) of CVRP, TRAVELCOST, depends on c_{ij}. Since c_{ij} are intervals in CVRPU, the function TRAVELCOST can not be used directly here. Being inspired by the Bertsimas-Sim approach explained in section 2.2.5, for handling the uncertain c_{ij} cost coefficients, we now define another objective function, PERTURBEDCOST, as follows:

$$\text{PERTURBEDCOST}(x,\Gamma) =$$

$$\begin{cases} \text{maximize} & \displaystyle\sum_{v\in V}\sum_{k=2}^{|x[v]|} \left(\underline{c}_{x[v,k-1],x[v,k]} + \gamma_{x[v,k-1],x[v,k]} \cdot (\overline{c}_{x[v,k-1],x[v,k]} - \underline{c}_{x[v,k-1],x[v,k]})\right) \\ \text{subject to} & \displaystyle\sum_{(i,j)\in A} \gamma_{ij} \le \Gamma \\ & 0 \le \gamma_{ij} \le 1 \quad \forall (i,j) \in A \end{cases} \tag{5.10}$$

Based on a linear programming formulation, the function PERTURBEDCOST is a function which finds the worst possible cost of a solution x, given that Γ amount of perturbation on the travel cost data is assumed. The perturbation assumption on a single arc (i,j) is represented by the variable γ_{ij}. When $\gamma_{ij} = 0$, it is assumed that there is no perturbation at all on the travel cost of arc (i,j), so, c_{ij} becomes equal to $\underline{c}_{ij}$. On the other hand, when $\gamma_{ij} = 1$, it is assumed that there is full perturbation on the travel cost of arc (i,j) towards its highest value, so, c_{ij} becomes equal to $\overline{c}_{ij}$. Also, when γ_{ij} is equal to, say, 0.5, the travel cost of (i,j) is perturbed halfway towards its highest value, therefore, c_{ij} becomes equal to $\underline{c}_{ij} + 0.5 \cdot (\overline{c}_{ij} - \underline{c}_{ij})$. Since the total amount of γ_{ij} perturbations is limited by Γ, the Γ value works as the conservativeness degree parameter in this formulation.

The function PERTURBEDCOST can be expressed in an alternative way, without actually running a linear programming solver. For making this alternative expression, let A^x be the set of all arcs used in the solution x. Also, let us call $\hat{c}_{ij} = \overline{c}_{ij} - \underline{c}_{ij}$ the *potential perturbation* on the cost of the arc (i,j). The idea here is as follows: we know that an arc $(i,j) \in A^x$ will contribute at least $\underline{c}_{ij}$ to

the total cost of the solution x. However, more perturbation could occur on c_{ij} towards its worst-case value, which would result in an extra contribution to the solution cost of x from the arc (i, j). So, $\hat{c}_{ij}$ expresses this possible extra contribution. Now, let us define $SortedA^x$ as an array, which contains all the arcs $(i, j) \in A^x$, sorted non-increasingly according to their $\hat{c}_{ij}$ values. We also define $SortedA^x_k$ as the k-th arc of the array $SortedA^x$. We are now ready to express the function PERTURBEDCOST: for the first $\lfloor \Gamma \rfloor$ arcs within $SortedA^x$, the travel costs are assumed to be at their highest values, for the $\lceil \Gamma \rceil$-th arc (i, j) of $SortedA^x$, the travel cost is assumed to be equal to $\underline{c}_{ij} + (\Gamma - \lfloor \Gamma \rfloor) \cdot \hat{c}_{ij}$, and for the rest of the arcs within $SortedA^x$, the travel costs are assumed to be at their lowest values. This approach corresponds to the following way of thinking: perturbations towards the higher values will happen on the arcs with the highest potential perturbations. By using this way of thinking, we find the maximum possible cost of a solution x, given that Γ amount of perturbations can happen in total. The detailed algorithmic expression of the function PERTURBEDCOST can be found in algorithm 2. Within this algorithm, the performance bottleneck is the sorting, which, by using a non-specialized standard technique, requires $O(|A^x| \cdot \log(|A^x|))$ execution time.

Algorithm 2 Algorithmic expression of the function PERTURBEDCOST

```
1: function PerturbedCost(x, Γ)
2:     cost ← 0
3:     uncertainty_budget ← Γ
4:     for k = 1 To |A^x| do
5:         (i, j) ← SortedA^x_k
6:         if uncertainty_budget ≥ 1 then
7:             cost ← cost + c̄_ij
8:             uncertainty_budget ← uncertainty_budget − 1
9:         else if 0 < uncertainty_budget < 1 then
10:            cost ← cost + c_ij + uncertainty_budget · ĉ_ij
11:            uncertainty_budget ← 0
12:        else if uncertainty_budget = 0 then
13:            cost ← cost + c_ij
14:        end if
15:    end for
16:    return cost
17: end function
```

Having defined the cost function PERTURBEDCOST in (5.10) and in algorithm

2, we are now ready to fully describe CVRPU as:

$$\text{CVRPU} \begin{cases} \text{minimize} & \textsc{PerturbedCost}(x, \Gamma) \\ \text{subject to} & (5.1), (5.2), (5.3), (5.4), (5.5) \end{cases}$$

that is, the maximum cost that can be encountered (expressed by PERTURBED-COST) according to the perturbation upper bound Γ (which is our conservativeness degree parameter) is to be minimized, while satisfying the constraints of the original CVRP model.

5.2.1.1 A Robust Ant Colony System for the CVRP

We have developed a robust ant colony system (RACS) for near-optimally solving CVRPU. The RACS is a version of ACS in which the objective function is PERTURBEDCOST. Therefore, the conservativeness degree is configured by the decision maker by setting the parameter Γ.

Experimental Environment. A term named *the price of robustness* is discussed in Bertsimas and Sim [2004a]. In general, the price of robustness means the sacrifice that must be made by the decision maker to have a robust solution. In Bertsimas and Sim [2004a], the authors analyze the price of robustness by showing how much is sacrificed from the optimality to increase robustness.

In this study, we analyze the price of robustness in two ways. First, we analyze the computational price of robustness: we compare the execution speeds of the regular ACS approach where a simple non-robust objective function is used, and of the RACS approach which uses PERTURBEDCOST as its objective function. This gives us an idea about how much speed is sacrificed to turn a regular metaheuristic into an uncertainty-aware one. Second, we analyze the operational price of robustness: given that the cost of a solution in the best-case scenario is its potential of being low-cost, we analyze how much is sacrificed from this potential for having a robust solution.

We have used CVRP instances called tai100a, tai100b, tai100c, and tai100d where 100 customers are considered; and tai150a, tai150b, tai150c, and tai150d where 150 customers are considered. These instances can be found in NEO Networking and Emerging Optimization [2012]. All tai100 solutions require at least 11 vehicles. All tai150 solutions require at least 14 vehicles, except for tai150a, where 15 vehicles are required. Since these instances were originally created for CVRP without the considerations of uncertainty, we have applied the following procedure to convert them into CVRPU instances: for each arc (i, j),

given that c'_{ij} is the deterministic travel cost in the original CVRP instance, we set $\underline{c}_{ij} = c'_{ij}$ and $\overline{c}_{ij} = c'_{ij} \cdot \text{RND}(1, 1.5)$, where $\text{RND}(a, b)$ is a random real number between a and b.

The ACS and RACS approaches were implemented in C programming language, configured to have 10 ants in a colony (like in Gambardella et al. [1999]), with settings $\alpha = 0.99$, $\beta = 1$, and $\rho = 0.1$ after some preliminary tuning. The computer used for the experiments is Intel Core 2 Duo P9600 @ 2.66GHz with 4GB of RAM. Note that this configuration of number of ants in a colony, α, β, and ρ is used for our further ACS-based methodologies that will be presented in sections 5.2.1.2 and 5.2.2.1 of this chapter. Also, all experiments presented within this chapter were done on the same computer mentioned above.

Computational Price of Robustness. As mentioned previously, in CVRPU, for evaluating a solution x by using the objective function PERTURBEDCOST, we have to spend $O(|A^x| \cdot \log(|A^x|))$ amount of time. This is an increase on the execution time requirement, considering $O(|A^x|)$ amount of time which would be needed by the objective function TRAVELCOST of the deterministic CVRP.

We now do an analysis of execution time requirement from a practical point of view. The two metaheuristic algorithm implementations, ACS (which uses TRAVELCOST objective function), and RACS (which uses PERTURBEDCOST objective function), are executed 5 times on the same instance files. Within 10 seconds, the average number of iterations performed by each implementation is noted. The results are presented in figure 5.3. From the figure, it can be seen that the deterministic ACS approach works at approximately twice the speed of the RACS approach.

Note that in the execution time comparison we do here, the entire iterations of ACS and RACS are measured, where each iteration involves the solution construction of all the ants of a colony, the execution of the *3-opt* local search and then the execution of the objective functions. Therefore, this is a comparison between ACS and RACS, rather than a comparison only between TRAVELCOST and PERTURBEDCOST. While the results on the execution time difference between only these functions would be much greater, we believe that the results presented here are the important ones, since these objective functions are expected to be used within metaheuristic algorithms, not by themselves.

Operational Price of Robustness. We now analyze how much we sacrifice from the solution's potential in terms of being low-cost (i.e. solution cost in the best-case scenario), when we increase the conservativeness degree for having more robust solutions. For this analysis, we execute the RACS approach multiple times

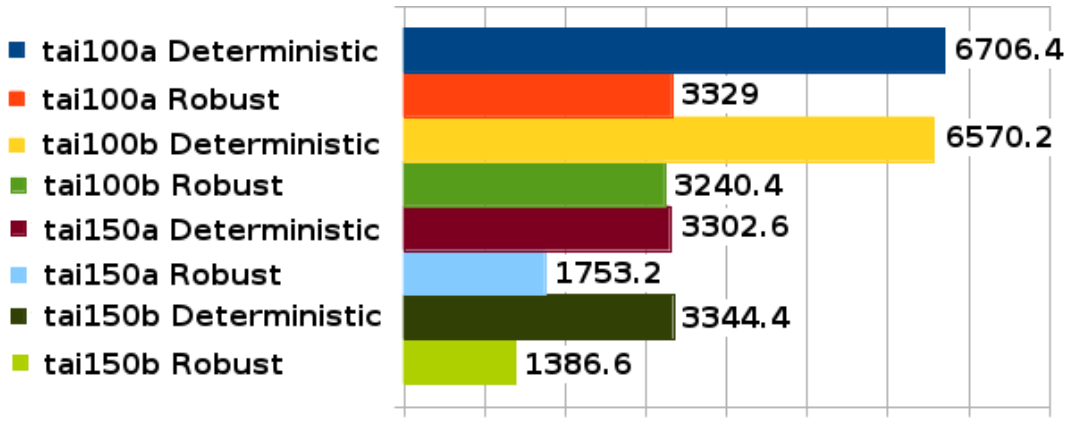


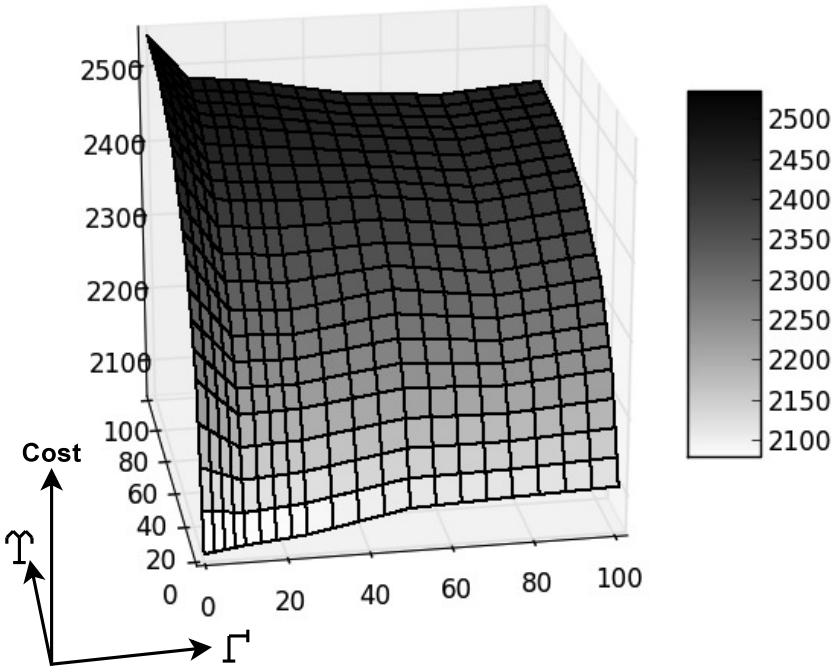
Figure 5.3. The average number of iterations performed by the ACS (deterministic) and RACS (robust) approaches.

on each instance, each time with a different conservativeness degree, configured by the parameter Γ. By collecting the different solutions generated by these multiple executions, we form a solution pool. To see how each solution within the solution pool performs under various scenarios, we now employ another parameter, Υ, that we call the *scenario assumption parameter*. Each solution x within the solution pool is evaluated according to PERTURBEDCOST(x, Υ). By this test, we learn how much a solution costs in the best-case scenario, in the worst-case scenario, and in partially pessimistic scenarios. The results of these tests are given in tables 5.1, 5.2, and 5.3. Among these results, let us have a closer look at the results obtained on the instances tai100a and tai150a, for which the surface plots are shown in figure 5.4. In the figure, for both tai100a and tai150a, it can be seen that when $\Gamma = 0$, the obtained results have the lowest costs in the best-case scenario ($\Upsilon = 0$). However, as the scenario assumptions become worse (as Υ is increased), it can be seen that those solutions are not robust, as they turn into the most expensive solution. With the conservativeness degree around $\Gamma = 10$, it can be seen that compromise solutions are found, where the best-case costs are slightly more expensive, and the worst-case costs are lower. As the Γ value is increased, it can be seen that we go towards the most conservative solutions, where the best-case costs are relatively high, and the worst-case costs are relatively low. With $0 \leq \Gamma \leq 25$ in the case of tai100a, and with $0 \leq \Gamma \leq 30$ in the case of tai150a, there seems to be a visible change in the behaviour of the solutions. So, these regions show us the trade-off between cost potential and robustness. With $\Gamma > 25$ in the case of tai100a, and with $\Gamma > 30$ in the case of tai150a, the solutions seem to behave similar.

Table 5.1. Results obtained from the experiments on the instances with 100 customers (Toklu et al. [2013a])

Instance	Solution	Cost evaluations					
	Γ	Υ=0	Υ=10	Υ=25	Υ=50	Υ=75	Υ=101
tai100a	0	2059.3	2262.51	2385.58	2483.72	2527.28	2542.42
	10	2067.02	2214.54	2319.65	2413.89	2464.96	2484.68
	25	2075.1	2217.22	2314.17	2407.53	2456.8	2477.0
	50	2100.52	2234.9	2332.78	2405.9	2440.32	2453.49
	75	2105.73	2229.88	2316.01	2390.74	2428.53	2444.34
	101	2110.14	2242.17	2331.53	2405.37	2440.95	2455.19
tai100b	0	2059.3	2262.51	2385.58	2483.72	2527.28	2542.42
	10	2067.02	2214.54	2319.65	2413.89	2464.96	2484.68
	25	2075.1	2217.22	2314.17	2407.53	2456.8	2477.0
	50	2100.52	2234.9	2332.78	2405.9	2440.32	2453.49
	75	2105.73	2229.88	2316.01	2390.74	2428.53	2444.34
	101	2110.14	2242.17	2331.53	2405.37	2440.95	2455.19
tai100c	0	1406.2	1574.7	1669.28	1725.35	1750.76	1762.02
	10	1421.6	1540.59	1623.88	1676.82	1700.11	1710.43
	25	1442.17	1539.99	1603.83	1657.4	1682.63	1694.73
	50	1463.51	1564.07	1621.19	1665.74	1687.35	1697.2
	75	1447.11	1557.79	1620.49	1668.53	1694.19	1707.81
	101	1463.25	1554.89	1606.65	1654.52	1677.93	1688.95
tai100d	0	1596.31	1736.59	1835.16	1921.65	1969.65	1986.28
	10	1607.26	1721.96	1812.32	1904.64	1954.99	1973.84
	25	1604.11	1712.9	1802.47	1888.03	1932.95	1948.57
	50	1629.59	1737.5	1810.49	1884.54	1927.25	1943.91
	75	1606.7	1728.08	1817.47	1894.48	1930.1	1944.75
	101	1668.21	1750.43	1821.35	1887.88	1925.2	1938.9

a) tai100a



b) tai150a

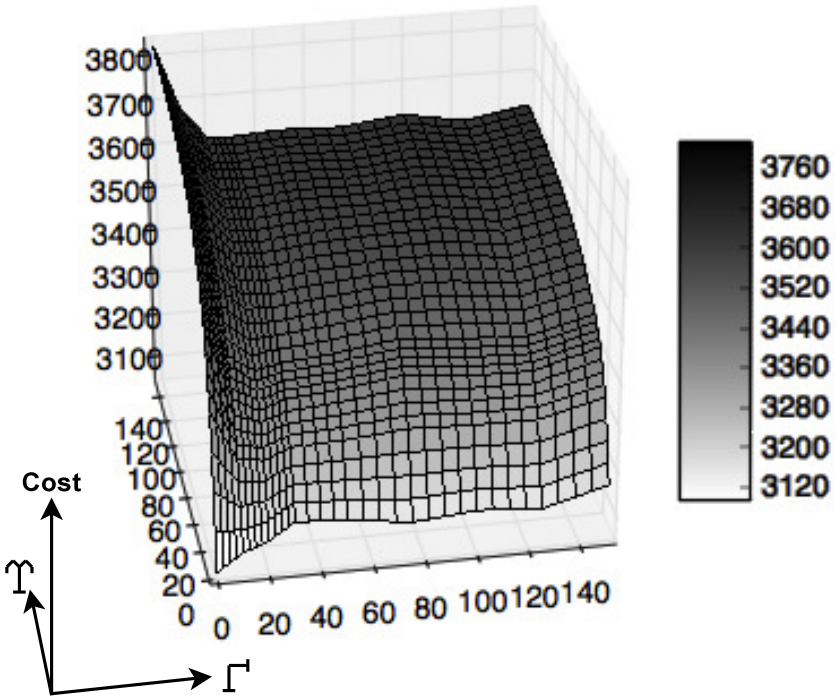


Figure 5.4. The surface plots of the costs of the solutions generated for the instances tai100a and tai150a. These plots show how different solutions are obtained with different conservativeness values (Γ) and how these solutions perform on different scenarios (Υ).

Table 5.2. Results obtained from the experiments on the instances with 150 customers (tai150a, tai150b)

Instance	Solution	Cost evaluations				
	Γ	ϒ=0	ϒ=10	ϒ=20	ϒ=30	ϒ=50
		ϒ=75	ϒ=100	ϒ=125	ϒ=151	
tai150a	0	3057.94	3390.46	3538.96	3615.69	3707.36
		3763.75	3797.9	3815.98	3825.48	
	10	3097.5	3277.93	3382.75	3450.47	3542.32
		3609.26	3648.49	3669.62	3679.91	
	20	3118.97	3270.62	3353.47	3411.85	3492.84
		3547.98	3584.12	3604.21	3614.06	
	30	3156.47	3294.29	3371.55	3423.79	3490.68
		3543.25	3579.6	3598.69	3608.56	
	50	3149.19	3298.85	3382.76	3433.91	3502.35
		3555.47	3589.55	3607.79	3615.42	
	75	3129.4	3310.89	3400.62	3453.07	3517.03
		3562.12	3587.66	3602.02	3610.41	
	100	3142.33	3305.57	3386.24	3441.66	3514.01
		3564.67	3597.65	3616.71	3627.67	
	125	3131.15	3304.53	3384.42	3432.99	3495.98
		3544.47	3576.53	3593.37	3602.18	
	151	3171.07	3340.74	3424.33	3470.37	3528.64
		3573.75	3601.08	3617.73	3626.56	
tai150b	0	2739.21	3069.54	3198.94	3274.44	3359.1
		3418.57	3453.35	3474.16	3484.27	
	10	2766.41	2921.46	2986.57	3028.48	3085.07
		3131.84	3161.56	3179.8	3189.79	
	20	2832.26	2948.69	3000.81	3039.63	3096.09
		3141.99	3170.57	3186.99	3195.47	
	30	2828.24	2931.2	2985.24	3029.07	3091.45
		3142.14	3173.95	3193.3	3202.86	
	50	2778.8	2904.21	2955.55	2992.29	3044.94
		3091.15	3120.9	3137.98	3146.97	
	75	2799.36	2919.01	2973.59	3013.9	3070.32
		3118.89	3151.56	3171.03	3180.76	
	100	2825.5	2925.73	2979.14	3016.48	3072.69
		3123.69	3156.0	3175.6	3185.57	
	125	2792.57	2912.71	2962.05	2998.15	3050.61
		3096.77	3127.12	3144.79	3153.86	
	151	2845.26	2957.83	3021.25	3061.43	3110.56
		3148.17	3172.65	3188.03	3196.82	

Table 5.3. Results obtained from the experiments on the instances with 150 customers (tai150c, tai150d)

Instance	Solution	Cost evaluations				
	Γ	ϒ=0	ϒ=10	ϒ=20	ϒ=30	ϒ=50
		ϒ=75	ϒ=100	ϒ=125	ϒ=151	
tai150c	0	2424.0	2710.5	2835.77	2891.73	2948.93
		2991.88	3019.11	3035.43	3044.21	
	10	2498.13	2657.34	2751.06	2811.87	2878.65
		2925.59	2952.89	2969.05	2976.76	
	20	2524.18	2671.85	2719.62	2752.32	2799.04
		2839.41	2867.92	2885.6	2894.25	
	30	2484.0	2629.85	2671.19	2703.18	2752.03
		2795.3	2824.23	2841.68	2851.8	
	50	2508.54	2632.85	2695.3	2733.76	2780.41
		2818.37	2840.73	2855.34	2862.65	
	75	2469.62	2577.21	2626.77	2658.43	2703.07
		2741.03	2765.74	2780.03	2787.32	
	100	2459.48	2616.64	2688.32	2730.48	2788.26
		2831.81	2858.76	2874.55	2881.33	
	125	2515.31	2639.38	2691.0	2724.58	2771.67
		2809.58	2833.72	2849.56	2856.55	
	151	2532.29	2657.48	2702.28	2732.72	2778.58
		2818.02	2842.06	2858.33	2867.75	
tai150d	0	2662.84	2932.68	3075.43	3143.09	3226.07
		3280.73	3311.14	3326.35	3333.13	
	10	2700.91	2884.32	2983.02	3050.67	3133.85
		3192.25	3224.2	3242.3	3251.39	
	20	2750.85	2913.03	2987.75	3043.98	3118.76
		3168.32	3196.22	3212.07	3219.87	
	30	2768.92	2903.33	2971.89	3020.14	3088.88
		3143.1	3178.74	3198.66	3207.74	
	50	2739.32	2889.63	2949.71	2994.59	3054.79
		3102.39	3131.79	3148.18	3156.05	
	75	2809.45	2924.43	2986.61	3024.55	3082.63
		3132.18	3163.69	3182.11	3190.63	
	100	2737.14	2929.63	2997.12	3035.84	3089.46
		3132.93	3157.66	3172.01	3179.5	
	125	2754.36	2915.53	2979.61	3027.07	3089.16
		3136.65	3164.26	3179.6	3186.36	
	151	2776.35	2928.92	2993.73	3033.26	3084.55
		3123.54	3148.25	3163.04	3169.71	

In tables 5.1, 5.2, and 5.3 further results can be found in which solution pools show similar characteristics. The common behavior is that the solution with $\Gamma = 0$ is the one which is the cheapest in the best-case scenario, but is also the most risky solution in the sense that it is most expensive in the worst-case scenario. The quickest decrease of this risk happens with Γ values close to (but larger than) 0. As the Γ value is further increased, this region of quickest-risk-decrease ends, and the solutions start to behave similar. In general, this shows us the effect of the travel cost uncertainty on CVRPU. By generating and analyzing a solution pool for the problem instance at hand, the decision maker can see this region of quickest-risk-decreasing, see the trade-off between potential cost and robustness within this region, and then pick the most practical solution.

Because of the heuristic nature of our approach here, one thing we notice is that, there are some noises in the solution pools. For example, let us say we have two solutions in a solution pool, namely χ^1 and χ^2. Let us also say that χ^1 was prepared with conservativeness degree $\Gamma = 10$, and that χ^2 was prepared with conservativeness degree $\Gamma = 25$. In the ideal case, when tested with $\Upsilon = 10$, χ^1 would cost less than χ^2; and when tested with $\Upsilon = 25$, χ^2 would cost less than χ^1. However, this is not always the case and we can have inconsistencies because our optimization algorithm is heuristic and it can get stuck around dominated solutions. Such inconsistencies cause noise in the solution pool. For example, we can observe this noise in the solution pool visualizations in figures 5.4a and 5.4b. In both solution pool visualizations, one can observe that there is noise in the solutions with the highest Γ configurations (let's call those solutions the *highest-Γ solutions*). The highest-Γ solutions are supposed to be the most conservative solutions, i.e. they should have the lowest costs in their pools when the highest Υ is considered. However, in both examples, one can observe that there are solutions which have lower costs than the highest-Γ solutions under the consideration of the highest Υ. This means that, when working with the highest Γ, the ant colony metaheuristic was stuck on a dominated solution. This is a problem that we face because we execute the ant colonies independently. In section 5.2.1.2, we explain an improved version of this approach, where this problem is addressed.

5.2.1.2 Robust Multiple Ant Colony System for the CVRPU

In section 5.2.1.1, we had pointed out the noises in the solution pools. To have a protection against these noises, we propose an approach that we call *robust multiple ant colony system* (RMACS). The main idea of RMACS is that multiple ACS processes, instead of a single one, are started, and executed concurrently.

Each ACS process focuses on a different conservativeness degree. During their execution, these ACS processes inform each other about their best solutions. This communication between the processes, allows a colony to become aware if its best solution is dominated by the another colony's best solution. In that case, the dominated colony receives the dominating colony's best solution, and starts improving that solution according to its own conservativeness degree. When the entire execution of RMACS is finished, the final solution of each ACS process is put into a solution pool, which is the result of the RMACS approach. This approach has interesting advantages, such as:

- *High quality solution pools*: Because of the solution sharing mechanism of RMACS, the ACS processes which would get stuck around a dominated solution can get unstuck by receiving better solutions, and improving them. In the end, this gives us solution pools which have better qualities.

- *Parallelism*: Because of the concurrent nature of the RMACS approach, different ACS processes can be started in different cores of the computer. A full parallelism can be achieved if the computer has a number of cores at least equal to the number of ACS processes. This would result in minimization of the interruption caused by one ACS process on another.

Now we explain the technical details about RMACS. The RMACS approach depends on a set of conservativeness level parameters: $S^{\Gamma} = \{\Gamma^1, \Gamma^2, ...\}$. In RMACS, $|S^{\Gamma}|$ number of ACS processes are executed concurrently, each ACS process being focused on a different conservativeness degree value within S^{Γ}. Up to Δ number of generations, these ACS processes work independently, so that they are given the chance of exploring various regions of the solution space without getting influenced by each other for a while. Starting with the Δ-th generation, each ACS process $AcsProc$ with the conservativeness degree $\Gamma^{AcsProc}$ sends its best solution into the shared memory, and scans all the other solutions sent by other ACS processes, periodically with an interval of δ generations. While scanning all the other solutions in the shared memory, the ACS process $AcsProc$ evaluates each solution χ by using PERTURBEDCOST($\chi, \Gamma^{AcsProc}$). If the evaluation result suggests that the solution χ is a better solution than the current best solution of $AcsProc$, an artificial ant within $AcsProc$ is forced to repeat the moves of the solution χ, therefore, the solution χ is imported.

Experimental Environment. We now present two types of experiments to test our RMACS approach. The first type of experiment involves the comparison of

the solution pools generated by the RMACS approach, and by a set of independent RACS processes with different conservativeness degree. This first type of experiments, therefore, is focused on seeing if the solution sharing mechanism of RMACS is really useful. The second type of experiments is the analysis of the RMACS solution pools on CVRPU instances generated according to different policies: an instance where the customer locations are randomly placed, an instance where the customer locations are clustered, and a hybrid instance where some customer locations are clustered and the others are randomly placed.

Solution pool comparison between RACS and RMACS. We now present our solution pool comparison studies, which were reported in Toklu et al. [2013b]. In this analysis, we use the Taillard instances, tai100{a,b,c,d} and tai150{a,b,c,d}, converted into CVRPU instances by following the procedure previously mentioned in the experimental environment explanation of our original RACS study in section 5.2.1.1. We have set the RMACS-specific parameters as $\Delta = 9000$ and $\delta = 500$. For both approaches (the RMACS approach and the approach of multiple RACS processes running independently), the conservativeness degrees that we consider are: $S^{\Gamma} = \{0, 10, 25, 50, 75, 100, M\}$ for tai100, and $S^{\Gamma} = \{0, 10, 20, 30, 50, 75, 100, 125, 150, M\}$ for tai150, where $M \geq |L| + |V| - 1$ is a number big enough to make all the cost assumptions equal to their highest values. For both approaches, the execution time limit was set as 420 seconds for tai100, and 600 seconds for tai150.

Let us now look at the comparison results between RMACS and RACS, shown in tables 5.4 and 5.5. In these tables, for each instance, the costs of the best performing solutions according to various scenario assumption parameter values found by each approach (RMACS and RACS) are reported under the column group “Best solutions”. With the exception of tai100b, we can observe that the RMACS approach was able to come up with better solutions. Therefore, we can conclude that the solution sharing mechanism was useful in increasing the qualities of the solution pools.

We now present our further experiments on RMACS approach, reported in Toklu, Gambardella and Montemanni [to appear], over bigger instances. In more details, these experiments were done over three Homberger instances (available online at NEO Networking and Emerging Optimization [2012]): c1_2_1, where the customers are clustered, r1_2_1, where the customers are randomly placed, and rc1_2_1, where some customers are clustered and the rest are randomly placed. In these instances, 200 customers are considered. Like the Taillard instances, these instances which were originally created for the deterministic CVRP, were converted into CVRPU instances in our

Table 5.4. Comparison of the performances of RMACS and RACS over tai100 instances

Instance	Approach	Best solution Υ=0	Υ=10	Υ=25	Υ=50	Υ=75
tai100a	RMACS	**2111.69**	**2289.01**	**2379.00**	**2455.40**	**2489.98**
	RACS	2153.36	2292.30	2390.51	2480.97	2512.53
tai100b	RMACS	1982.75	2127.34	2217.51	2301.44	2346.44
	RACS	**1979.76**	**2081.67**	**2174.16**	**2273.68**	**2324.43**
tai100c	RMACS	**1430.51**	**1557.58**	**1617.43**	**1668.98**	**1691.29**
	RACS	**1430.51**	1567.21	1632.17	1678.19	1700.25
tai100d	RMACS	**1635.54**	1751.05	**1837.03**	**1912.00**	**1950.40**
	RACS	1641.08	**1747.34**	1843.79	1925.99	1968.40

Instance	Approach	Best solution Υ=100	Υ=M
tai100a	RMACS	**2503.67**	**2504.95**
	RACS	2527.19	2528.79
tai100b	RMACS	2364.31	2365.82
	RACS	**2346.52**	**2349.11**
tai100c	RMACS	**1701.50**	**1702.49**
	RACS	1709.44	1710.64
tai100d	RMACS	**1964.51**	**1965.84**
	RACS	1983.86	1985.53

experiments. For solving each instance, the execution time limit was set as 1200 seconds. The RMACS was configured as: $\Delta = 9000$, $\delta = 500$, $S^{\Gamma} = \{0, 10, 25, 50, 75, 100, 150, M\}$ where $M \geq (|L|+|V|-1)$ is a number big enough to make all the cost assumptions equal to their highest values.

The tables 5.6, 5.7, and 5.8 show the experimental results. The illustration of the solution pool generated for rc1_2_1 can also be seen in figure 5.5. When we look at the figure, we can say that the observed behaviour in the solution pool generated by the RMACS approach for the instance rc1_2_1 is consistent with our previous findings reported in section 5.2.1.1: the solution with the biggest potential in terms of being low-cost, but also with the biggest risk is the one with $\Gamma = 0$. Close to this solution, there is a region $0 \leq \Gamma \leq 25$, where solutions which have different levels of protections against the uncertainty can

Table 5.5. Comparison of the performances of RMACS and RACS over tai150 instances

		Best solution				
Instance	Approach	Υ=0	Υ=10	Υ=20	Υ=30	Υ=50
tai150a	RMACS	3164.09	**3309.12**	**3372.54**	**3416.03**	**3477.72**
	RACS	3164.09	3361.52	3469.17	3538.94	3617.44
tai150b	RMACS	2861.56	**2988.24**	**3042.75**	**3081.17**	**3137.42**
	RACS	2861.56	3012.18	3067.24	3105.59	3161.47
tai150c	RMACS	**2450.88**	**2647.74**	**2700.43**	**2735.82**	**2785.53**
	RACS	2535.16	2705.01	2760.79	2795.46	2846.54
tai150d	RMACS	2800.31	**2952.66**	**3016.70**	**3059.61**	**3115.96**
	RACS	**2790.64**	2980.95	3050.67	3104.67	3168.07

		Best solution				
Instance	Approach	$\Upsilon = 75$	Υ=100	Υ=125	Υ=150	Υ=M
tai150a	RMACS	**3527.92**	**3560.96**	**3580.08**	**3589.48**	**3590.67**
	RACS	3675.82	3714.07	3735.12	3744.55	3745.65
tai150b	RMACS	**3180.28**	**3210.21**	**3226.96**	**3235.23**	**3236.44**
	RACS	3204.27	3233.80	3251.54	3260.07	3261.34
tai150c	RMACS	**2830.46**	**2859.45**	**2876.52**	**2885.80**	**2886.75**
	RACS	2892.05	2922.07	2940.33	2950.87	2952.14
tai150d	RMACS	**3163.34**	**3191.15**	**3208.45**	**3216.26**	**3217.44**
	RACS	3216.02	3243.09	3259.39	3266.32	3267.08

be found. Therefore, a compromise solution can be found within this region. When we get past this region, we reach an area $\Gamma \geq 25$, in which the solutions are very conservative and behave almost the same. It can be observed from the values in the tables 5.6 and 5.7 that the solution pools for the instances c1_2_1 and r1_2_1 also behave similarly. In more details, the quick decrease of the worst-case-scenario costs within the region $0 \leq \Gamma \leq 25$ can be observed when one looks at the solution costs under the scenario assumptions $\Upsilon = M$ for the solutions with $\Gamma = 0$, $\Gamma = 10$, and $\Gamma = 25$.

Although the RMACS approach was designed to remove the noise problem introduced in our RACS studies, in the experimental results here, some noises can still be observed. For example, in table 5.8, the solution with $\Gamma = M$ performs better than the solution with $\Gamma = 150$, under the scenario assumption

Table 5.6. Results obtained for the instance c1_2_1

	ϒ=0	ϒ=10	ϒ=25	ϒ=50
Γ=0	2605.96	2866.45	3034.68	3153.61
Γ=10	2681.62	2820.88	2928.86	3019.85
Γ=25	2699.18	2824.15	2914.27	3003.31
Γ=50	2732.36	2848.15	2930.63	2999.28
Γ=75	2734.13	2849.92	2932.41	2999.83
Γ=100	2734.13	2849.92	2932.41	2999.83
Γ=150	2758.34	2871.99	2943.99	3005.08
Γ=M	2734.13	2849.92	2932.41	2999.83

	ϒ=75	ϒ=100	ϒ=150	ϒ=M
Γ=0	3197.64	3226.57	3259.61	3275.24
Γ=10	3069.5	3101.97	3140.08	3155.45
Γ=25	3052.59	3084.66	3123.87	3139.54
Γ=50	3039.24	3066.04	3098.75	3114.38
Γ=75	3038.34	3064.58	3096.96	3112.59
Γ=100	3038.34	3064.58	3096.96	3112.59
Γ=150	3041.88	3067.79	3099.15	3114.47
Γ=M	3038.34	3064.58	3096.96	3112.59

$\Upsilon = 150$. In other words, the solution with $\Gamma = M$ dominates the solution with $\Gamma = 150$. The general explanation for this noise might be that, towards the end of the execution of the RMACS approach, an ACS process $AcsProc1$ finds a solution which dominates the best solution of another ACS process $AcsProc2$, but the global execution time limit is reached before the dominating solution of $AcsProc1$ is uploaded to the shared memory and received by $AcsProc2$. However, the remaining noise is relatively small, which can also be visually analyzed from the smooth surface plot in figure 5.5. Such noise can easily be removed by the decision maker, after a final analysis is done on the solution pool and the dominated solutions are removed.

Note that, here, for the sake of experimenting, we have used a large set $S^\Gamma = \{0, 10, 25, 50, 75, 100, 150, M\}$, which dictates the concurrent execution of 8 ant colonies according to the RMACS approach. However, the region with differently behaving solutions is actually $0 \leq \Gamma \leq 25$. A small subset of S^Γ covers this region. Therefore, while solving CVRPU, in practice, one can ignore

Table 5.7. Results obtained for the instance r1_2_1

	Υ=0	Υ=10	Υ=25	Υ=50
Γ=0	3051.78	3227.33	3374.83	3526.16
Γ=10	3100.14	3198.38	3308.91	3439.13
Γ=25	3099.16	3199.03	3308.5	3438.54
Γ=50	3099.16	3199.03	3308.5	3438.54
Γ=75	3103.04	3201.29	3310.76	3438.67
Γ=100	3106.31	3205.29	3314.76	3441.95
Γ=150	3099.14	3207.5	3316.19	3441.05
Γ=M	3099.49	3208.58	3317.27	3442.13

	Υ=75	Υ=100	Υ=150	Υ=M
Γ=0	3630.15	3699.6	3787.16	3826.32
Γ=10	3527.06	3592.27	3676.18	3711.87
Γ=25	3525.68	3589.44	3670.35	3705.07
Γ=50	3525.68	3589.44	3670.35	3705.07
Γ=75	3525.03	3588.31	3670.2	3705.46
Γ=100	3526.48	3586.82	3666.77	3700.49
Γ=150	3525.3	3585.26	3665.06	3698.77
Γ=M	3525.64	3585.2	3664.99	3698.71

the conservativeness degrees outside this region (for which the conservativeness degree upper bound would change according to the size of the instance, but still it would be less than $|L|/2$ most of the time, probably around $|L|/4$ based on the experimental results we present here). Ignoring unnecessary conservativeness degree would result in smaller S^{Γ} sets, less number of concurrent ant colonies, and, therefore, less processes in the computer and a higher performance.

5.2.2 Capacitated Vehicle Routing Problem with Time Windows and Uncertain Travel Times

We now present our studies on CVRP with time window constraints under travel time uncertainty (CVRPTWU), discussed in Toklu et al. [2014]. First, let us remember that in CVRPTW, for each arc (i, j), we have two types of information: the travel cost c_{ij}, and the travel time t_{ij}. The objective here is the same with CVRP: the minimization of total c_{ij} over each traveled arc (i, j). The travel

Table 5.8. Results obtained for the instance rc1_2_1

	Υ=0	Υ=10	Υ=25	Υ=50
Γ=0	2959.8	3174.16	3359.28	3523.84
Γ=10	2995.42	3144.16	3283.99	3422.38
Γ=25	3000.15	3145.25	3277.53	3410.51
Γ=50	3010.08	3155.25	3272.44	3391.08
Γ=75	3012.8	3163.98	3289.29	3409.08
Γ=100	3008.45	3159.64	3289.65	3411.82
Γ=150	3009.94	3154.56	3286.9	3409.16
Γ=M	3007.04	3156.12	3274.97	3389.75

	Υ=75	Υ=100	Υ=150	Υ=M
Γ=0	3617.19	3672.79	3734.4	3759.04
Γ=10	3504.58	3554.7	3610.7	3632.0
Γ=25	3492.55	3542.9	3598.97	3620.67
Γ=50	3466.75	3514.89	3568.81	3589.72
Γ=75	3481.8	3528.47	3582.41	3602.34
Γ=100	3482.49	3527.72	3580.69	3600.35
Γ=150	3484.08	3530.92	3585.13	3606.17
Γ=M	3462.04	3506.51	3558.52	3578.99

time data t_{ij} are used to determine if a vehicle is able to satisfy a customer's time window constraints (which are expressed as $[T_j^B; T_j^E]$ for customer j also considering the service time T_j^S).

In CVRPTWU, the uncertainty is on travel times. Like in our CVRPU study, we put this uncertainty into our model in a compatible way with the previous related studies in the literature (Lee et al. [2012]; Agra et al. [2013]). Therefore, to express this uncertainty, we now say that t_{ij} data are intervals: $t_{ij} \in [\underline{t}_{ij}; \overline{t}^{ij}]$. We now have to express the time window constraints in such a way that the uncertain t_{ij} values are handled.

Let us first start with the appearance time. Because the travel times are uncertain, a vehicle's appearance to its k-th destination, A_k^v, is now scenario-dependent. However, in all scenarios, the vehicles start from the depot at time 0, therefore, the constraint (5.7) is still valid. Let us now define a vehicle's

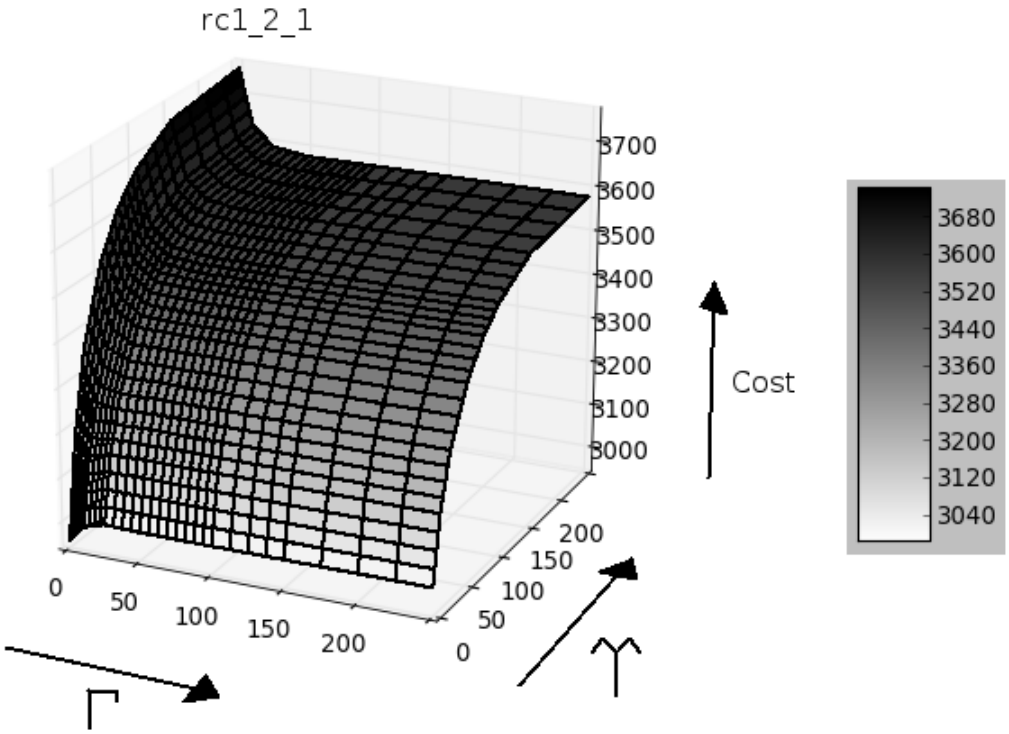


Figure 5.5. The solution pool generated by the RMACS approach for the instance rc1_2_1.

appearance time in the best-case scenario as $\underline{A}_k^v$, and formulate it as follows:

$$\underline{A}_k^v = \max(T^B_{x[v,k]}; \underline{A}^v_{k-1} + T^S_{x[v,k-1]} + t_{x[v,k-1],x[v,k]}) \quad \forall v \in V;\ k \in \{2,3,...,|x[v]|\} \tag{5.11}$$

Now we say, at least in the best-case scenario, we have to make sure that a vehicle does not appear to a location on its route later than that location's time window ending:

$$\underline{A}_k^v \leq T^E_{x[v,k]} \quad \forall v \in V;\ k \in \{1,2,...,|x[v]|\} \tag{5.12}$$

Note that (5.11) and (5.12) are the same with the classical CVRP time window constraints (5.8) and (5.9). In more details, (5.11) and (5.12) are re-applications of (5.8) and (5.9) on the best-case scenario. Using these constraints according to the best-case scenario assumptions guarantees the satisfaction of the time window deadlines in the best-case scenario. However, even if the feasibility of the best-case scenario is guaranteed, there could be (perhaps many)

other scenarios in which the time window deadlines are violated. To diminish such scenarios as much as possible, we have to define a function called TIMEWINDOWVIOLATIONPENALTY, which returns a penalty amount when there is a time window violating scenario. Note that, by using a penalty function, our study differs from Lee et al. [2012] and Agra et al. [2013], where absolute measures are taken to make sure that the time windows are always satisfied. Our perspective here is that when the ACS metaheuristic can not find a solution which satisfies all the time window constraints considering the given uncertainty set, it should at least find solutions with less number of violations by minimizing the return value of the time window violation penalty function.

As a first step towards reaching a complete definition of the function TIMEWINDOWVIOLATIONPENALTY, let us use another function which was proposed first in Agra et al. [2013], and let us call that function MAXIMUMLATENCY in this study. The function MAXIMUMLATENCY depends on a conservativeness degree configuration mechanism inspired by Bertsimas and Sim [2003]. Therefore, like the Bertsimas-Sim approach explained in section 2.2.5, the function MAXIMUMLATENCY depends on a Γ parameter. Given a CVRPTWU solution x, a vehicle v, a destination k (as in k-th destination on the route $x[v]$, which means $x[v,k]$); the function call MAXIMUMLATENCY(x, v, k, Γ) returns the latest appearance time of a vehicle v to its k-th destination, with the assumption that, on the route $x[v]$, the travel time requirements of Γ number of arcs are at maximum and the travel time requirements of the rest of the arcs on the route are at minimum. If the return value of this function call is greater than the time window ending of the k-th destination on the route, (i.e. if MAXIMUMLATENCY$(x, v, k, \Gamma) > T^E_{sol[v,k]}$) then we can conclude that there is at least one scenario in which there is a time window violation.

Let us now look at the formulation of the function MAXIMUMLATENCY:

$$\textsc{MaximumLatency}(x, v, k, \Gamma) =$$

$$\begin{cases} 0 \qquad \text{if } k = 1 \\ \\ \max\Big(T^B_{x[v,k]}, \textsc{MaximumLatency}(x, v, k-1, 0) + T^S_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]}\Big) \\ \qquad \text{if } 2 \leq k \leq |x[v]| \text{ and } \Gamma = 0 \\ \\ \max\Big(T^B_{x[v,k]}, \\ \quad \textsc{MaximumLatency}(x, v, k-1, \Gamma-1) + T^S_{x[v,k-1]} + \overline{t}_{x[v,k-1],x[v,k]}, \\ \quad \textsc{MaximumLatency}(x, v, k-1, \Gamma) + T^S_{x[v,k-1]} + \underline{t}_{x[v,k-1],x[v,k]} \Big) \\ \qquad \text{if } 2 \leq k \leq |x[v]| \text{ and } 1 \leq \Gamma \leq k-1 \\ \\ -\infty \quad \text{if } 1 \leq k-1 \leq |x[v]| \text{ and } \Gamma \geq k \end{cases}$$

One can see that the function MaximumLatency is a recursive function: excluding the special cases, the result of the function for the k-th destination depends on the result of the function for the $(k-1)$-th destination.

Here, differently from Agra et al. [2013] where a conservativeness degree mechanism inspired by the Bertsimas-Sim approach is used, we define a CVRPTWU-specific conservativeness degree configuration mechanism, dependent on a parameter $\Psi \in [0; 1]$. According to this mechanism, considering the k-th destination of vehicle v in solution x (i.e. considering $x[v,k]$) over the path leading to that k-th destination, we assume that the travel time requirements of $\lceil \Psi \cdot (k-1) \rceil$ number of arcs over that path will be at their maximum, and the travel time requirements of the rest of the arcs over that path will be at their minimum. In other words, the parameter Ψ configures our assumption on the ratio of the arcs over a path which are subject to perturbation towards the worst-case values in terms of travel time requirements. Therefore, $\Psi = 0$ means that we assume all the arcs over a path will have their best-case travel time requirements, and $\Psi = 1$ means that we assume all the arcs over a path will have their worst-case travel time requirements. The purpose of this conservativeness degree mechanism is to allow the decision maker to express her/his pessimism relative to the length of a route. On a route with 2 arcs, $\Gamma = 2$ would mean full conservatism. But on a route with 4 arcs, $\Gamma = 2$ would mean half conservatism. However, when we use the Ψ-based mechanism, half conservatism can be expressed regardless of the route length, by setting $\Psi = 0.5$.

Now, by using the Ψ-based conservativeness degree mechanism, we define a

function, IsLate, which determines if there is a scenario in which the vehicle v can miss the time window deadline of its k-th destination.

$$\text{IsLate}(x, v, k, \Psi) = \begin{cases} 1 & \text{if MaximumLatency}(x, v, k, \lceil \Psi \cdot (k-1) \rceil) > T^E_{x[v,k]} \\ 0 & \text{otherwise} \end{cases}$$

We are now ready to define the function TimeWindowViolationPenalty, which depends on the Ψ conservativeness degree parameter, as:

$$\text{TimeWindowViolationPenalty}(x, \Psi) =$$

$$\sum_{v \in V} \sum_{k=1}^{|x[v]|-1} \left(\Pi_{x[v,k]} \cdot \text{IsLate}(x, v, k, \Psi) \right)$$

where Π_i is a penalty coefficient to be decided by the decision maker. According to how important the location $i \in L$ is, and according to the attributes of the specific problem at hand, the decision maker can come up with various criteria for setting Π_i values.

Note that, in our experiments presented in Toklu et al. [2014], we had the assumption that the uncertainty-caused delays to the final destination (i.e. arrivals to the final destination later than $EndTime$) are tolerated, because the final destination is not a customer, but the depot. Because of this reason, in the function TimeWindowViolationPenalty, the summation which iterates over k values is going up to $|x[v]| - 1$. In cases where such delays are not acceptable from a management point of view (because of working hour regulations and/or availability of the depot itself), the decision maker can easily rearrange the formulation by setting the upper bound of the summation to $|x[v]|$, instead of $|x[v]| - 1$.

Finally, by incorporating the time window violation penalty mechanism into the objective, we are ready to complete the definition of CVRPTWU:

$$\text{CVRPTWU} \begin{cases} \text{minimize} & \text{TravelCost}(x) + \text{TimeWindowViolationPenalty}(x) \\ \text{subject to} & (5.1), (5.2), (5.3), (5.4), (5.5), (5.7), (5.11), (5.12) \end{cases}$$

5.2.2.1 A Robust Multiple Ant Colony System for the CVRPTWU

To solve CVRPTWU, we propose a version of our RMACS approach (which was originally proposed for our CVRPU studies, explained in section 5.2.1.2). Let us call the CVRPTWU adaptation of the RMACS approach RMACS-TW. Like the original RMACS approach, RMACS-TW activates multiple ACS processes,

each focused on a different conservativeness degree. Again, like in the original RMACS approach, in RMACS-TW, the ACS processes exchange information about their solutions, and an ACS process with a dominated solution imports a better solution from another ACS process and continues to work on that imported solution. At the end of the execution of RMACS-TW, a solution pool is generated.

The details of the RMACS-TW approach are as follows. The approach depends on a set of considered conservativeness degrees, $S^{\Psi} = \{\Psi^1, \Psi^2, ..., \Psi^{|S^{\Psi}|}\}$. When the algorithm is started, $|S^{\Psi}|$ number of ACS processes are activated: one ACS process for each Ψ conservativeness degree value within the set S^{Ψ}. Since the beginning of the execution, until Φ seconds have passed, each ACS process works independently, without engaging in any interprocess communication. Starting with the Φ-th second, at each period of ϕ seconds, each ACS process uploads its best solution into the shared memory, and scans the memory to see if its own best solution is dominated by the best solution of another ACS process. For simplicity of expression, let us define: $\text{Ç}(x, \Psi) = \textsc{TravelCost}(x) + \textsc{TimeWindowViolationPenalty}(x, \Psi)$. While scanning the shared memory, if an ACS process $AcsProc1$, with the conservativeness degree $\Psi^{AcsProc1}$ and the best solution $x^{AcsProc1}$, finds out that another ACS process $AcsProc2$ has uploaded a solution $x^{AcsProc2}$ which is better according to the conservativeness degree $\Psi^{AcsProc1}$ (i.e. if $\text{Ç}(x^{AcsProc2}, \Psi^{AcsProc1}) < \text{Ç}(x^{AcsProc1}, \Psi^{AcsProc1})$), the process $AcsProc1$ imports the solution $x^{AcsProc2}$ by forcing one of its ants repeat the moves stored in $x^{AcsProc2}$.

Experimental environment and the results. An implementation of RMACS-TW was done in C, and the set of considered conservativeness degrees was configured as $S^{\Psi} = \{0, 0.25, 0.5, 1\}$.

For testing our approach, we have used the CVRPTW instances of Homberger with 200 customers, with vehicle capacity 200. These instances can be found online at NEO Networking and Emerging Optimization [2012]. Originally, these instances were created for the deterministic CVRPTW without the consideration of uncertainty on the travel time data. To convert these CVRPTW instances into CVRPTWU instances, the following procedure was followed: for each arc (i, j) in the instance, t'_{ij} being the travel time coefficient in the original CVRPTW instance, the uncertain travel time data were set as $\underline{t}_{ij} = t'_{ij}$ and $\overline{t}_{ij} = t'_{ij} \cdot \textsc{Rnd}(1, 1.1)$, where $\textsc{Rnd}(a, b)$ means a random real number between a and b. The time window violation penalty factor Π_i for each location i was set as: $\Pi_i = (20 \cdot \overline{t}_{0,i})$.

The results can be seen in table 5.9. The table shows the solution pools for each instance. Within each solution pool, each solution with a different conservativeness degree $\Psi \in S^{\Psi}$, and its evaluation result (evaluation result being $Ç(x, \Theta)$ for solution x, under a scenario assumption Θ) under each scenario assumption $\Theta \in S^{\Psi}$ is reported. From the results, we can see the following patterns:

- *Solution pools showing a trade-off between robustness and low solution cost*: In some solution pools, we can observe that the conservativeness degree Ψ has a visible effect on the behaviours of the solutions. With $\Psi = 0$, the lowest-cost solution is found, but the return value of the evaluation function $Ç(x, \Psi)$ increases quickly under worse scenarios, because of the time window violation penalties. To decrease the time window violation possibilities, one has to look at solutions generated with higher Ψ values. Increasing Ψ, however, also increases the cost. With $\Psi = 1$, the most robust solution is generated, suffering none from the time window violation penalty. In the solution pools generated for the instances c1_210, rc1_210, r1_2_2, rc1_2_6, this kind of pattern can be seen. The solution pools found for the instances r1_2_1 and rc1_2_2 also show a very similar pattern, the difference being that even in the most robust solution with $\Psi = 1$, the time window violation penalty is minimized but not gone.

- *Solution pools in which there is only one dominating solution*: In the solution pools of r1_2_3 and rc1_2_9, one can see that there is only one solution. This means that, during the execution of RMACS-TW, a particular solution was found to be cheapest one and also the most robust one, dominating all other solutions.

- *Solution pools with noises*: It can be seen that the solution pools generated by the RMACS-TW approach has noises in some cases. For example, in the solution pool of rc1_2_5, the best-performing solution under the scenario assumption $\Theta = 0$ is not the solution with $\Psi = 0$, but it is the solution with $\Psi = 0.25$. Like our explanation about the noise in our RMACS study in section 5.2.1.2, a general explanation for this kind of behaviour might be that, after the last solution sharing, an ACS process finds a dominating solution, but before uploading that dominating solution to the shared memory, the global execution time limit is reached. As we have noted in our previous RMACS study in section 5.2.1.2, this noise does not pose a problem, as it can be easily nullified when the decision maker makes a final analysis on the solution pool, and eliminates the dominated solutions.

Table 5.9. CVRPTWU solution pools obtained over the Homberger instances

		Solution Pool			
Instance	Ψ	Θ =0	Θ =0.25	Θ =0.5	Θ =1
r1_2_1	0	5110	19494.35	20142.4	26099.18
	0.25	5254	7849.96	10959.84	17071.08
	0.5	5441	8219.64	8219.64	11171.77
	1	5495	8273.64	8273.64	9746.38
rc1_2_2	0	3772	15635.07	15635.07	15635.07
	0.25	4559	4559	9825.95	14576.69
	0.5	3792	6275.62	6275.62	6275.62
	1	3792	6275.62	6275.62	6275.62
r1_2_3	0	4175	4175	4175	4175
	0.25	4175	4175	4175	4175
	0.5	4175	4175	4175	4175
	1	4175	4175	4175	4175
c1_2_9	0	2711	4522.33	5876.63	5876.63
	0.25	2734	2734	2734	2734
	0.5	2734	2734	2734	2734
	1	2734	2734	2734	2734
rc1_2_7	0	3797	5161.16	5161.16	5161.16
	0.25	3810	3810	3810	3810
	0.5	3810	3810	3810	3810
	1	3810	3810	3810	3810
c1_2_6	0	2633	7175.61	7175.61	8180.72
	0.25	2710	2710	3704.32	4709.43
	0.5	2711	2711	2711	3716.11
	1	2784	3897.46	3897.46	3897.46
c1_2_3	0	2747	2747	2747	2747
	0.25	2747	2747	2747	2747
	0.5	2747	2747	2747	2747
	1	2747	2747	2747	2747
c1_2_1	0	2637	2637	4019.08	4019.08
	0.25	2637	2637	4019.08	4019.08
	0.5	2643	2643	2643	2643
	1	2643	2643	2643	2643
rc1_2_5	0	4200	19002.08	21417.9	25143.77
	0.25	4148	4148	4148	5717.38
	0.5	4506	4506	4506	4506
	1	4514	4514	4514	4514
rc1_2_3	0	3547	3547	3547	3547
	0.25	3547	3547	3547	3547
	0.5	3547	3547	3547	3547
	1	3547	3547	3547	3547

Table 5.9 continued

Instance	Ψ	Solution Pool Θ =0	Θ =0.25	Θ =0.5	Θ =1
c1_2_8	0	2638	5567.98	5567.98	5567.98
	0.25	2888	2888	2888	2888
	0.5	2888	2888	2888	2888
	1	2632	2632	2632	2632
r1_2_6	0	4577	22366.84	22366.84	22366.84
	0.25	4708	4708	4708	5983.74
	0.5	4708	4708	4708	5983.74
	1	4713	4713	4713	4713
r1_2_4	0	3617	3978.65	3978.65	3978.65
	0.25	3629	3629	3629	3629
	0.5	3629	3629	3629	3629
	1	3618	3618	3618	3618
r1_2_2	0	4713	11063.71	17508.34	17508.34
	0.25	4723	4723	7351.3	7351.3
	0.5	4775	4775	4775	4775
	1	4775	4775	4775	4775
r1_2_5	0	4769	15842.03	18544.94	21465.66
	0.25	5150	5150	5150	5150
	0.5	5150	5150	5150	5150
	1	5150	5150	5150	5150
r1_2_7	0	3853	10125.12	11581.65	13599.23
	0.25	3971	3971	3971	3971
	0.5	3971	3971	3971	3971
	1	3971	3971	3971	3971
c1_2_4	0	2754	4139.71	4139.71	4139.71
	0.25	2815	2815	2815	2815
	0.5	2815	2815	2815	2815
	1	2815	2815	2815	2815
rc1_2_9	0	3724	3724	3724	3724
	0.25	3724	3724	3724	3724
	0.5	3724	3724	3724	3724
	1	3724	3724	3724	3724
r1_2_8	0	3350	3350	3350	3350
	0.25	3350	3350	3350	3350
	0.5	3350	3350	3350	3350
	1	3350	3350	3350	3350
rc1_2_6	0	3838	5437.27	9803.87	9803.87
	0.25	3847	3847	6316.43	7742.8
	0.5	3849	3849	3849	3849
	1	3849	3849	3849	3849

Table 5.9 continued

Instance	Ψ	Solution Pool Θ =0	Θ =0.25	Θ =0.5	Θ =1
r1_210	0	3961	3961	3961	3961
	0.25	3961	3961	3961	3961
	0.5	3961	3961	3961	3961
	1	3961	3961	3961	3961
rc1_2_8	0	3866	3866	3866	3866
	0.25	3866	3866	3866	3866
	0.5	3866	3866	3866	3866
	1	3866	3866	3866	3866
c1_2_5	0	2634	3929.72	3929.72	5204.48
	0.25	2636	2636	2636	3910.76
	0.5	2636	2636	2636	3910.76
	1	2645	2645	2645	2645
rc1_210	0	3529	8361.52	9377.92	9377.92
	0.25	3619	3619	4950.9	4950.9
	0.5	3621	3621	3621	3621
	1	3621	3621	3621	3621
c1_210	0	2675	5820.03	5820.03	7138.31
	0.25	2770	2770	4097.4	5321.76
	0.5	2828	2828	2828	2828
	1	2828	2828	2828	2828
c1_2_7	0	2628	30076.99	30076.99	30076.99
	0.25	2640	2640	2640	2640
	0.5	2640	2640	2640	2640
	1	2640	2640	2640	2640
c1_2_2	0	2830	6037.26	7958.18	8666.53
	0.25	2945	2945	3604.84	5190.4
	0.5	3076	3076	3076	3076
	1	2905	2905	2905	2905
rc1_2_1	0	3995	16403.61	17187.83	22312.59
	0.25	4265	4958.12	9095.4	11738.39
	0.5	4458	5848.23	5848.23	6668.15
	1	4390	6082.6	6082.6	6902.53
r1_2_9	0	4453	10911.1	10911.1	10911.1
	0.25	4684	4684	4684	4684
	0.5	4684	4684	4684	4684
	1	4684	4684	4684	4684
rc1_2_4	0	3395	3395	3395	3395
	0.25	3395	3395	3395	3395
	0.5	3395	3395	3395	3395
	1	3395	3395	3395	3395

5.2.3 General comments on our studies

In our studies explained in sections 5.2.1 and 5.2.2, we have used robust ACS-based metaheuristic techniques (RACS and RMACS) to generate solution pools for medium-to-large sized CVRPU and CVRPTWU instances. Inspired by Bertsimas and Sim [2004a], these solution pools contain solutions with various degrees of conservativeness. The results we have obtained show that the solution pools contain useful alternative solutions with various lowest and highest possible costs, or various levels of immunities against infeasibility caused by uncertainty. We have also seen that our RMACS technique, thanks to its concurrently-running ant colonies which periodically inform each other of their best solutions, increases the qualities of the generated solution pools. In general, under the uncertainty of travel costs or travel times, it can be difficult to tell which routing scheme is the best solution. With the solution pools generated by our RMACS method, the decision maker can analyze alternative solutions, and pick the one which seems the most practical.

Chapter 6

Matheuristic Robust Optimization Framework

Previously, in section 5.2.1.2, we discussed a concurrent matheuristic robust optimization approach called robust multiple ant colony system (RMACS), focused on the uncertainty-aware vehicle routing problems. In this chapter, we first focus on the concept of matheuristics and its practicality within the field of robust optimization in general, and then discuss how the concurrent matheuristic robust optimization approach can be treated as a general framework, and how it can be applied on many combinatorial optimization problems.

6.1 Using matheuristics for robust optimization

As previously mentioned in chapter 1, a matheuristic algorithm can be described as a hybridization between a mathematical programming approach (linear programming, integer programming, etc.), and a metaheuristic approach. We now discuss how this hybridization is especially practical in the case of robust optimization, and therefore also in the case of our framework.

Usually, in robust optimization, a solution is evaluated according to its worst-case scenario. This means, when calculating a solution's cost, we consider a special scenario in which the uncertain coefficients maximize that solution's cost (see, for example, Kouvelis and Yu [1997]). Because of this, we usually encounter a *minimax* scheme, which can be expressed in a compact way as follows:

$$\underset{x \in X}{\text{minimize}} \left(\underset{c_j \mid j \in J}{\text{maximize}} \left(\sum_{j \in J} c_j x_j \right) \right) \tag{6.1}$$

that is, we are trying to minimize the cost by looking for a practical solution x within the solution space X, and we are evaluating the cost of the solution x under a scenario in which the uncertain cost coefficients $c_j \in J$ maximize the expenses of the solution x. In (6.1), the maximization part represents a subproblem of finding the worst-case scenario within the main optimization problem. Similarly, when we consider the uncertainty in the constraint coefficients, the usual practice within the field of robust optimization is to find the worst-case scenario in which the uncertain coefficients are configured in such a way that the constraint is closest to being unsatisfied (or most far away from being satisfied). In other words, the satisfaction of the constraint is checked under the most "dangerous" scenario (Ben-Tal and Nemirovski [2000]). Considering upper bound constraints indexed as $i \in I$, this practice can be expressed in a compact way as follows:

$$\underset{a_{ij} \mid j \in J}{\text{maximize}} \left(\sum_{j \in J} a_{ij} x_{ij} \right) \leq b_i \quad \forall i \in I \tag{6.2}$$

that is, considering the solution x, and the uncertain coefficients a_{ij} for the constraint $i \in I$, whether the left-hand side exceeds the upper bound b_i or not is checked under the worst-case scenario in which the left-hand side is at its highest value. The maximization term within (6.2) represents the subproblem of finding that worst-case scenario, which looks for the most dangerous values for a_{ij}. In the rest of this chapter, let us refer to these subproblems demonstrated in (6.1) and (6.2) as *worst-case-finding subproblems*. The formulation of worst-case-finding subproblems can be observed also in the Bertsimas-Sim approach discussed in section 2.2.5, within the model (2.5), where inner maximization terms can be seen both in the objective and in the constraints.

In general, metaheuristics are used when it is acceptable to solve a problem to near-optimality, instead of to exact optimality. However, even when using metaheuristics and therefore looking for near-optimal solutions, it is usually desirable to solve the worst-case-finding subproblems to their optimalities, to make sure that a solution's worst-case cost and worst-case feasibility are correctly evaluated. To solve these subproblems into optimality, embedding mathematical models into the metaheuristics and solving them by using exact methods, therefore using matheuristic hybridization techniques, is a practical choice. Therefore, within our matheuristic framework as well (for which a detailed discussion takes place in section 6.2), one of the most important steps is to make the objective function and the constraints uncertainty-aware by embedding such models of subproblems into the metaheuristic algorithm. Note that, however, ac-

cording to the types of subproblems obtained, there can be efficient alternative algorithms, in addition to the classical techniques like the Simplex algorithm, to solve them into optimality. For example, in the case of our uncertainty-aware vehicle routing study presented in chapter 5, we solve the subproblem modelled as (5.10), by using the algorithm 2.

6.2 Description of the matheuristic robust optimization framework

We now give a general description of our framework, its design principles, and how to apply it.

In short, the matheuristic robust optimization framework we propose can be explained as a general guideline for a developer to prepare an uncertainty-aware heuristic for solving a combinatorial optimization problem and obtain a solution pool, containing solutions with various conservativeness degrees and costs in the end. The design principles of this framework are as follows:

Execution time. An optimization problem has to be solved to near-optimality within reasonable amount of time, without requiring too much computer memory. Because of this, the framework relies on a metaheuristic algorithm for performing a search within the solution space.

Reliable solution evaluation. Solutions have to be evaluated in a reliable way, considering the uncertainty. To achieve this, we use robust optimization techniques to express the uncertainty-aware objective and the constraints. To evaluate a solution in an accurate, reliable way in its worst-case, we use the matheuristic techniques discussed in section 6.1: we embed exact solvers into the metaheuristic algorithm to solve the worst-case-finding subproblems into their optimalities.

Alternative solutions. The decision maker should be provided with solution pools, containing alternative solutions with different conservativeness degrees, so that an analysis on the trade-off between robustness and solution cost can be made. Such an analysis is helpful for making a practical decision. This principle is satisfied by executing multiple processes of the metaheuristic algorithm, with the help of a solution sharing mechanism, like in the RMACS approach discussed in section 5.2.1.2.

In section 6.2.1, we continue with a step-by-step explanation on how to apply the framework in a problem-independent way.

6.2.1 Applying the framework

Let us assume that we want to solve a combinatorial optimization problem. Let us name the simple, uncertainty-unaware version of this problem as P, and its uncertainty-aware counterpart as PU. We now explain which steps are required to apply the framework to solve PU.

6.2.1.1 Step one

The first step to solve the problem PU by using our framework is to select a metaheuristic algorithm proposed for solving P. In the literature, various metaheuristic algorithms are proposed for various combinatorial optimization problems of different natures. Therefore, the selection of metaheuristic in this step depends heavily on the nature of the problem P. Selecting a metaheuristic algorithm reported to be efficient for solving P will satisfy the design principle of obtaining a solution within reasonable amount of time without requiring too much memory, and will positively effect the qualities of solution pools we generate for PU in the end.

6.2.1.2 Step two

The second step of applying the framework is to select a robust optimization approach, which were previously mentioned in section 2.2, and then re-express the objective function evaluator and/or the constraint satisfaction checker of the metaheuristic algorithm according to the selected robust optimization approach. At this step, the selection of the robust optimization approach depends on the type of uncertainty considered on PU. Since, in the end, we want to obtain solution pools that store different solutions with different conservativeness degrees, it is important that the selected robust optimization approach can be configured in terms of conservativeness. In general, if the uncertainty can be considered in the interval form, the approach of Bertsimas-Sim is usually a good choice, because it provides the possibility of configuring the conservativeness degree, and evaluating a solution according to the perspective of Bertsimas-Sim takes polynomial time in many problems, which means it avoids big losses in terms of execution speed of the metaheuristic algorithm. After selecting the robust optimization approach, the parts of the metaheuristic algorithm which deal with

uncertain coefficients (the objective function evaluator and/or the constraint checker) are modified, so that a solution's quality and feasibility is analyzed according to the selected robust optimization approach.

Since this step involves re-expressing the objective function and the constraint checkers of the metaheuristic to make them uncertainty-aware, this is where we encounter worst-case-finding subproblems mentioned in section 6.1, and this is where we make use of the technique of embedding exact algorithms into the metaheuristic to solve those subproblems. By doing so, we satisfy the design principle of evaluating a solution correctly within its worst-case scenario.

6.2.1.3 Step three

In the third and final step of applying the framework, our goal is to execute multiple processes of the modified metaheuristic algorithm, where each process is configured to have a different conservativeness degree. In the end, all the solutions produced by these processes form a solution pool: a collection of alternative solutions with different costs and conservativeness degrees. Also, with the help of a shared memory, these metaheuristic processes exchange information from time to time, so that when a metaheuristic process is stuck in a local minimum, it can import a better solution from another process. This step can be divided into following substeps:

- A shared memory is implemented. This shared memory can be implemented as a separate program which listens to the running processes of the metaheuristic optimization implementations. In this shared memory, there is a memory slot for the best solution known so far for each conservativeness degree. The shared memory has two important interfaces: (*i*) when the process of a metaheuristic implementation contacts the shared memory and sends its best known solution, the shared memory process will save that best solution in the correct slot of memory; (*ii*) when the process of a metaheuristic implementation demands from the shared memory to see all the solutions, the shared memory process will send that process all the solutions kept in its records.

- Metaheuristic algorithms are modified such that they send their best solutions to the shared memory from time to time, and they also import all the solutions to see if there is a better solution in a different slot of the shared memory (sent by another running process of the metaheuristic with a different conservativeness degree). When such a better solution is found, the metaheuristic process imports that solution as its current solution.

This methodology of handling the uncertainty is summarized in figure 6.1.

6.3 Using different metaheuristics within the framework

From the description made in section 6.2, one can notice that the shared memory is a component completely independent of the metaheuristic algorithm used: it stays the same regardless of the choice of the metaheuristic. In addition, many various metaheuristics can be modified to have communication with the shared memory without doing fundamental changes in the algorithm. We now discuss more about using various metaheuristics within the framework.

Ant colony system (ACS). In our studies explained in chapter 5, we had used ACS. In the original definition of ACS, the best solution found is kept by the metaheuristic. So, we modify the original definition of ACS by imposing that this best solution has to be sent to the shared memory from time to time. When it comes to importing a better solution, as previously mentioned in chapter 5, we forcefully make an ant repeat the moves of that better solution. When the solution is imported this way, the best solution of the ant colony and the artificial pheromones are updated by the ACS, without further modification in the code.

Iterated local search. A simple iterated local search would work around the concept of a *current solution*: at each iteration, the algorithm would try to improve the current solution by applying a local search on it. After the local search, if the solution is improved, the improved solution would be the new current solution. This means, the current solution is the same as the best solution known so far by the algorithm. Therefore, importing and exporting a solution would be equivalent to taking a better solution from the shared memory and labeling it as the new current solution, and sending the current solution from time to time, respectively.

Simulated annealing. Simulated annealing is an algorithm similar to the iterated local search, except that it has a temperature variable, and according to the state of this temperature variable, the current solution can be replaced by a worse solution, because of the fact that accepting worse solutions from time to time can make the heuristic search escape from a local minimum and can lead to better results in the long run. Because of this behaviour, the current solution is not always the best solution known so far, and the best solution known so far is kept separately. Solution export can be handled simply by sending that best

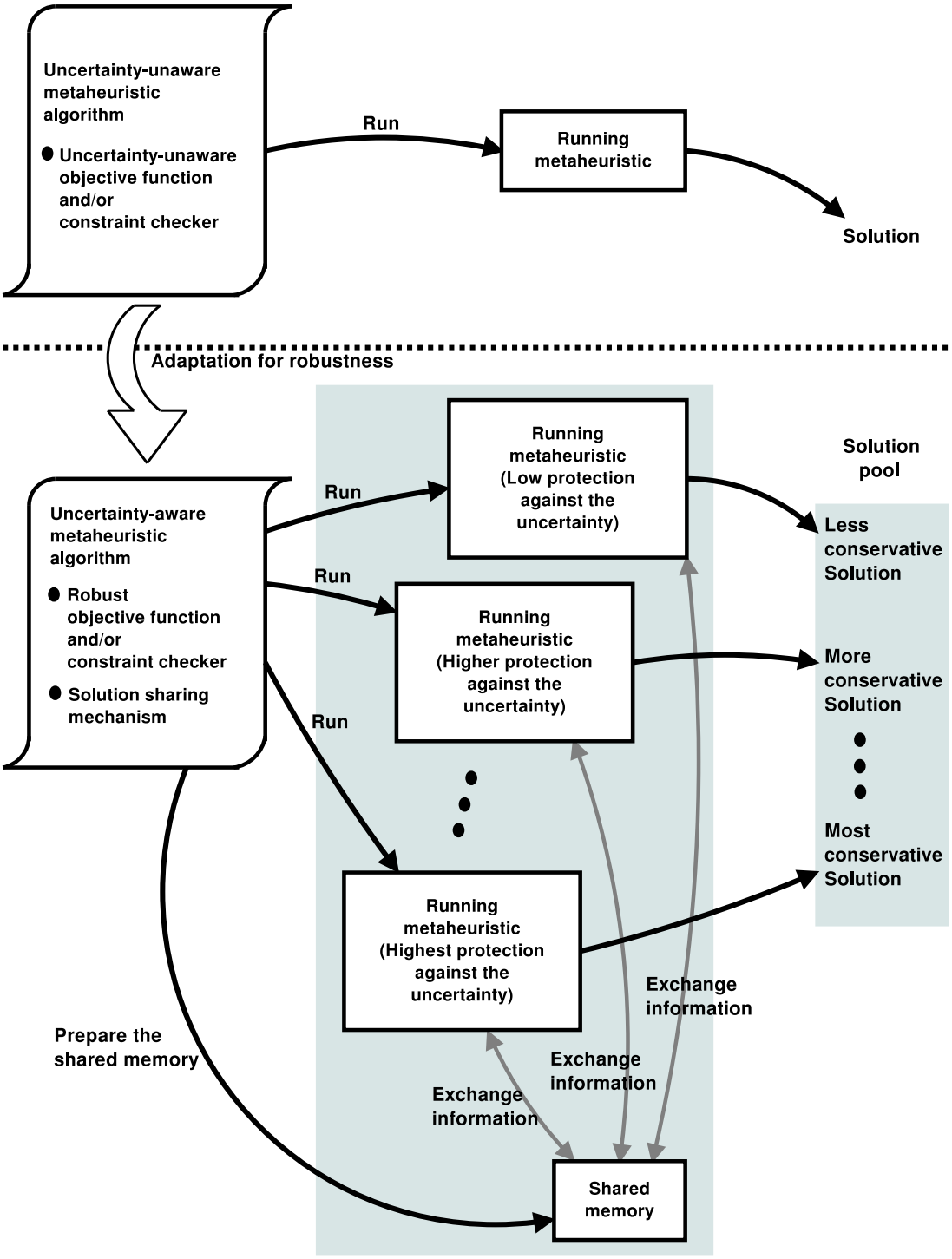


Figure 6.1. The summary of the methodology proposed.

solution from time to time to the shared memory. One technique for solution importing could be to take a better solution from the shared memory and label it both as the new current solution and as the new best solution. For deciding the policy about the temperature variable during the exporting of a better solution is a research question: one can find an appropriate policy on this after running some experiments.

Tabu search. An important metaheuristic algorithm within the literature is tabu search (Glover and Laguna [1997]). Like iterated local search and simulated annealing, tabu search has the concept of "current solution". This current solution is to be iteratively improved by using local search techniques. The unique aspect of tabu search is that it involves memory structures ("tabu list") which remember the characteristics of the solutions previously visited and labeled as unpromising local minimum and/or infeasible. Those characteristics kept in the memory are declared as "tabu", and the algorithm avoids exploring new solutions showing those characteristics. The tabu search algorithm has been used for important combinatorial optimization problems, including vehicle routing problem (Gendreau et al. [1994]; Barbarosoğlu and Özgür [1999]), scheduling (Xhafa et al. [2009]), etc. Being another metaheuristic using the concept of current solution, tabu search could also be used as a component of our framework. However, further experiments are needed to find the best policy for the solution sharing mechanism about the tabu list.

Population-based metaheuristic algorithms. Genetic algorithm, evolution strategies are examples to population-based algorithms: they update a "population" (a collection of solutions) instead of a single "current solution". In the case of a population-based algorithm, importing and exporting could be implemented getting a better solution from the shared memory and writing it over the worst solution in the population, and sending the best solution in the population to the shared memory, respectively.

6.4 Using the framework on various problems

Now, let us discuss the generality of our proposed framework over various combinatorial optimization problems. In chapter 5, we had considered capacitated vehicle routing problem, as it is a well-known optimization problem, frequently encountered and studied both in academia and industry. However, the same framework can be used for many other popular combinatorial optimization prob-

lems, in which uncertainty is an important factor. Below, we discuss some of these problems which we have not covered in our studies, but which we think as possible target problems for our framework.

Orienteering problem. The orienteering problem is another transportation-related combinatorial optimization problem, for which a survey can be found in Vansteenwegen et al. [2011]. According to a very basic definition of the orienteering problem, we have a depot, a vehicle, and customers at various locations. Visiting each customer means profit, the amount of the profit depending on the particular customer. Given that a global deadline is set, and the time available until this global deadline is not enough to visit all the customers, our purpose is to generate a route for our vehicle for visiting a subset of these customers, such that the profit is maximized. An uncertainty-aware version of the basic orienteering problem is discussed in Campbell et al. [2011], where the travel and service times are subject to uncertainty. In their study of uncertainty-aware orienteering problem, the authors consider that, if the company commits to serving a customer but the assigned vehicle can not reach that customer before the deadline (because the travel/service times take longer than anticipated), the profit of the company suffers a penalty. The authors use the variable neighbourhood search metaheuristic (a method initially proposed in Mladenović and Hansen [1997]) for finding heuristic solutions for this problem. A robust optimization variation of the orienteering problem (where the travel and service times are subject to uncertainty, and their probability distribution information are not known) could be considered, in which higher conservativeness degrees would trigger pessimistic assumptions about the travel/service times, and would lead to tours that avoid dangerous commitments. Our matheuristic robust optimization framework can be adapted for the robust optimization variation of the orienteering problem, in which multiple processes of variable neighbourhood search are executed concurrently, each process being focused on a different conservativeness degree.

Home healthcare service optimization problem. Increasing the expectancy of life within the society is a very important topic. For this purpose, there has been a growing interest since the last decade on the home healthcare service optimization problem. This problem is encountered by health institutions, and it involves the assigning of the nurses to patients, scheduling the treatments on the patients, and routing of the nurses so that they use efficient paths and reach their destinations quickly. These patients to be visited could be people who are discharged from hospital but still need final treatments for their full recovery, el-

derly people who are in need of care without coming to the hospital, etc. It has been stated in Nickel et al. [2009] that this problem is NP-Hard, therefore usage of metaheuristics is required for medium-to-large sized instances. In Nguyen and Montemanni [to appear], the authors mention the challenges on this problem brought by the uncertainty. Indeed, availabilities of the nurses, and travel and service times are all data which can be perturbed significantly because of uncertainty. A metaheuristic algorithm which takes the uncertainty into account, and which can be configured in terms of conservativeness degree, could be implemented and run in parallel with the support of solution sharing mechanism. In the case of the travel time/cost uncertainty consideration, the Bertsimas-Sim-inspired formulations we have used in chapter 5 could be adapted.

Scheduling problems. Scheduling problems are encountered in various fields, and they are affected by various uncertainties. The energy management problem mentioned in section 4.2 involves the scheduling of the maintenance operations and the productions of the power plants in such a way that multiple customer demand scenarios are satisfied. A variant of this energy management problem, in which the customer demand scenarios are labeled as optimistic, pessimistic, extremely-pessimistic etc., could be considered. This labeling would enable us to have a conservativeness degree configuration, and therefore our framework could be applied to obtain solution pools. Also, as mentioned previously, a very important subproblem within home healthcare optimization problem is the scheduling of the visits of the nurses to patients. This scheduling problem is affected by the uncertainty of the availability of the nurses, and the conservativeness degree is configured according to how pessimistic we are when we make our assumptions on the number of missing nurses. Considering this conservativeness degree configuration, nurse scheduling is another problem which can be studied by using our framework. Finally, another important scheduling problem is the project scheduling with resource constraints (see Hartmann and Briskorn [2010] for a survey). Shortly, in this problem, the purpose is to schedule activities of a project in such a way that the project finishes as early as possible. It is also common in this problem to have constraints imposing specifing time windows for beginnings or endings of some activities. Studies on handling the uncertainty in activity durations within this problem have been reported in the literature (see Herroelen and Leus [2005] for a survey; also see Artigues et al. [2013] for a recent study). A matheuristic approach developed according to our framework could result in an interesting alternative methodology for handling this problem.

Chapter 7

Conclusions

Uncertainty is a very important factor in optimization problems, and ignoring the uncertainty could result in solutions which turn out to be far from being optimal in reality. During this PhD research, our goal was to develop heuristic methodologies for robust optimization problems for solving large problems subject to uncertainty.

For achieving the goal of developing heuristic methodologies for robust optimization problems, we first explored the field of robust optimization which has given us interesting results. In more details, our studies on aggregate blending problem has resulted in a simple yet practical linear-programming-based approach, in which the uncertainty is handled, and the degree of conservativeness is configurable; and our studies on minimum power multicasting problem has resulted in a heuristic technique which can find reliable solutions power requirement data subject to stochasticity. In addition to these contributions, we obtained further interesting results in the field of matheuristics: our studies on large-scale energy management problem has shown the effectiveness of matheuristic techniques for handling large robust optimization problems; and our studies for solving the minimum power multicasting problem by using shared incumbent environment has shown the effectiveness of solution sharing mechanism, which is also used in our final matheuristic robust optimization framework. Besides being stepping stones towards our final robust optimization framework, these studies we have done resulted in contributions into the fields of robust optimization and matheuristics on their own.

After our previous stepping-stone studies, we focused on combining the robust optimization techniques with matheuristics. Within this perspective, our first study was a robust ant colony system (RACS), for solving a vehicle routing problem with uncertain travel costs. In our experiments, RACS was seen

to be able to provide solution pools in which different solutions with different costs and robustness levels are kept, so that the decision maker can see and analyze the trade-off between the cost and robustness, and then make a practical decision. Therefore, the price of robustness addressed previously by the ellipsoid approach and the Bertsimas-Sim approach was put into a metaheuristic environment, bringing the advantage of handling medium-to-large sized vehicle routing problems with uncertainty. Then, to increase the qualities of the solution pools generated by RACS, we introduced a matheuristic approach called robust multiple ant colony system (RMACS), in which multiple processes of RACS run concurrently with different configurations of conservativeness degrees, and exchange their solutions from time to time, so that when a RACS process is stuck at a local minimum, it becomes aware and imports a better solution from another RACS process. In our experiments, we have shown that the solution sharing mechanism we introduced for RMACS indeed increases the qualities of the solution pools, compared to the results obtained by RACS.

Finally, we can argue that the matheuristic approach we used for solving vehicle routing problems with uncertain data is actually a problem-independent robust optimization framework for handling uncertainty and solving medium-to-large sized problems while avoiding the big memory and/or execution time requirements of the pure mathematical programming based approaches. To elaborate more on the generality of the framework, a very important component within our approach, the solution sharing mechanism, does not depend on any attribute of the problem being solved. The other components of the framework, the metaheuristic algorithm and the robust optimization approach, are more problem-dependent, but the decision maker/developer can make use of the existing work available in the literature for having these components prepared. In short, the steps to be taken to apply the framework on a robust optimization problem can be described as: (*i*) select a metaheuristic algorithm suitable for the uncertainty-unaware version of the problem at hand; (*ii*) modify the objective function evaluator and the constraint checker of the selected metaheuristic by using a robust optimization approach, so that the metaheuristic algorithm becomes uncertainty-aware; (*iii*) implement a shared memory and a solution exchange mechanism, so that multiple instances of the modified metaheuristic will run concurrently and notify each other of their own best solutions, and get each other unstuck from local minima. Although we considered the vehicle routing problems under uncertainty in our studies, the generality of our framework makes it possible to solve many combinatorial optimization problems under uncertainty. Especially, if there is a well-known metaheuristic algorithm proposed in the literature for efficiently solving the uncertainty-unaware version of a prob-

lem to near-optimality, it can be very practical for the decision maker/developer to adopt our framework, and prepare a matheuristic solver for uncertainty-aware version of that problem. By using our framework, the possibilities of future research include solving uncertainty-aware counterparts of various scheduling problems within the fields of home healthcare and project management, and further transportation problems (orienteering problems, etc.).

Appendix A

Large-Scale Energy Management Problem: Technical Details

In section 4.2, we discussed our matheuristic MATHDEC approach for the large-scale energy management problem (LSEMP). Here we present technical details of the mathematical models we used within our approach. These formulations are previously presented in the online addendum (Anghinolfi et al. [2011a]) of the study Anghinolfi et al. [2012]. We also list them below for providing a complete description of the MATHDEC approach.

A.1 Details of the MIP Outage Scheduling submodule

The MIP module defines and produces a schedule of outages for T2 plants, using values passed by the HOSG or by the LS. These values specify an upper and a lower bound for the week when the decoupling of each plant can be scheduled. The schedule is feasible with respect to scheduling constraints ([CT13]-[CT21]). Constraints on power production are not taken into account in this procedure.

The original objective function is not taken into account in this phase, so, MIP objective corresponds to a constraint feasibility problem.

Details of the model follow.

Sets and indices

- $i \in \mathbb{I} \triangleq \{0, \ldots, I-1\}$ is the index of type 2 power plants
- $k \in \mathbb{K} \triangleq \{-1, 0, \ldots, K-1\}$ is the index of cycles, $k = -1$ represents the campaign in progress at the beginning of the time horizon, before the first outage

- $t \in \mathbb{T} \triangleq \{0, \ldots, T-1\}$ is the index of steps
- $h \in \mathbb{W} \triangleq \{0, \ldots, H\}$ is the index of weeks. If an outage is scheduled in week H it means that it is not going to be executed in the planning horizon. $\mathbb{W}_f \subseteq \mathbb{W}$ is a set of weeks where scheduling constraints (A.16) hold
- W_h is the set of all timesteps t contained in week h
- CT_f is the set of outages indexed by scheduling constraint$f$, with $f \in \mathbb{F} \triangleq \{14, \ldots, 21\}$.

Data preparation

- the length of a week $L_h^{\mathbb{H}} = \sum_{t \in W_h} L_t^{\mathbb{T}}$ is calculated as the sum of the lengths $L_t^{\mathbb{T}}$ of the timesteps in that week
- the maximum cumulative fuel consumption up to week h is the sum on all of the week of the maximum power output multiplied by the week durations: $\mathrm{P}_{ih}^{\max} \triangleq \sum_{z=0}^{h} P_{iz}^{\max} \cdot D_z^{\mathbb{H}}$
- the refueling ratio $R_{ik}^r = \frac{Q_{ik}-1}{Q_{ik}}$ is calculated using the refueling coefficients Q_{ik} specified in the datasets
- N_{ik}^L is the first possible week h where plant i can start outage of cycle k.
  It is a lower bound defined as $N_{ik}^L = \max(T_{ik}^o, L_{ik}^b)$, where T_{ik}^o is specified in the datasets by [CT13], and L_{ik}^b is a lower bound given in input to the MIP by the HOSG or by the LS (see respectively sections **??** and **??**).
- N_{ik}^U is the last possible week h for plant i to start outage of cycle k.
  It is an upper bound defined as $N_{ik}^U = \min(T_{ik}^a, U_{ik}^b)$, where T_{ik}^a is specified in the datasets by [CT13] and U_{ik}^b is an upper bound given in input to the MIP by the HOSG or by the LS (see respectively sections **??** and **??**)

Constants

- $L_h^{\mathbb{H}}$ is the length of a week
- $\mathrm{P}_{ih}^{\max}$ is the maximum cumulative fuel consumption up to week h
- N_{ik}^L is the first possible week h where plant i can start outage of cycle k
- N_{ik}^U is the last possible week h where plant i can start outage of cycle k

- L^O_{ik} is the length of outage k for plant i
- $R^{\min}_{ik}$ and $R^{\max}_{ik}$ are respectively the minimum and the maximum reload for plant i in cycle k
- $S^{\max}_{ik}$ and $A^{\max}_{ik}$ are the maximum bounds on stock of fuel respectively during production campaign and at the time of outage of cycle k
- B_{ik} is a threshold on stock of fuel for plant i in cycle k. Under the threshold a power profile is imposed. The constant is also used in constraints (A.4)
- R^r_{ik} is a refueling ratio
- A_f is a set of plants. It is indicated as $A_f(h)$ in case it depends on week h
- Se_f is a spacing between outages for set f
- I_f and F_f are the initial and final week of set f where constraints (A.10) hold
- ST_{ikf} is the number of weeks after the start of outage k of plant i in set f where the use of a certain kind of resource is used, $\mathrm{ST}_{ikf} \in \left[0, L^O_{ik}\right)$
- TU_{ikf} is the time of use, in weeks, of a certain resource. It applies to plant i of set f in cycle k
- E_{fh} is the maximum number of outages of set f that can be executed in parallel in week h
- $O^{\max}_f$ is a bound on maximum offline power capacity for plants in set f
- M is a *big-M*, a constant bigger than any other constants in the model.

Decision Variables

- $d_{i,k,h} \in \mathbb{B}$ is 1 if plant i begins the k-th outage in week h, defined $\forall i,k,h : N^L_{ik} \leq h \leq N^U_{ik}$. It is the decoupling date.
- $x^B_{ik} \in \mathbb{R}_+$ is the remaining stock of fuel in plant i at the end of cycle $k-1$, that is to say right before cycle k
- $x^A_{ik} \in \mathbb{R}_+$ is the initial stock of fuel in plant i at the beginning of cycle k, that is to say right after cycle $k-1$
- $r_{ik} \in \mathbb{R}_+$ is the reload of plant i in cycle k.

Mixed integer program

$$R_{ik}^{\min} \cdot \left(1 - d_{ikH}\right) \leq r_{ik} \leq R_{ik}^{\max} \cdot \left(1 - d_{ikH}\right) \quad \forall i,k \tag{A.1}$$

$$x_{i,-1}^{A} = X_{i}^{0} \quad \forall i \tag{A.2}$$

$$x_{ik}^{B} \geq x_{i,k-1}^{A} - \sum_{h=0}^{H-1} d_{ikh} \cdot \mathrm{P}_{ih}^{\max} + \sum_{h=0}^{H-1-L_{ik}^{O}} d_{i,k-1,h} \cdot \mathrm{P}_{i,h+L_{i,k-1}^{O}}^{\max} - d_{ikH} \cdot M + d_{i,k-1,H} \cdot M \quad \forall i,k \tag{A.3}$$

$$x_{ik}^{A} = R_{ik}^{r} \cdot (x_{ik}^{B} - B_{i,k-1}) + r_{ik} + B_{i,k-1} \quad \forall i,k \tag{A.4}$$

$$x_{ik}^{B} \leq A_{i,k}^{\max} \quad \forall i,k \tag{A.5}$$

$$x_{ik}^{A} \leq S_{i,k}^{\max} \quad \forall i,k \tag{A.6}$$

$$\sum_{h=0}^{H} d_{ikh} = 1 \quad \forall i,k \tag{A.7}$$

$$\sum_{k=0}^{K-1} d_{ikh} \leq 1 \quad \forall i,h \tag{A.8}$$

$$d_{ikh} + \sum_{h'=h}^{\min(H-1,h+L_{ik}^{O}+\mathrm{Se}_{f})} d_{i'k'h'} \leq 1 \quad \forall f,i,i',k,h : f \in \mathrm{CT}_{14} \wedge i,i' \in A_{f} \wedge i \neq i' \tag{A.9}$$

$$d_{ikh} + \sum_{h'=h}^{\min(H-1,h+L_{ik}^{O}+\mathrm{Se}_{f}-1,F_{f})} \sum_{k'=0}^{K-1} d_{i'k'h'} \leq 1 \begin{cases} \forall f,i,i',k,h : f \in \mathrm{CT}_{15} \wedge i,i' \in A_{f} \\ \wedge\, i \neq i' \\ \wedge\, h = \max(0, I_{f} - L_{ik}^{O} + 1), \ldots, \\ \ldots, \min(F_{f}, H-1) \end{cases} \tag{A.10}$$

$$d_{ikh} + \sum_{h=h'}^{\min(H-1,h+\mathrm{Se}_{f})} \sum_{k'=0}^{K-1} d_{i'k'h'} \leq 1 \quad \forall f,i,i',h,k : f \in \mathrm{CT}_{16} \wedge i,i' \in A_{f} \wedge i \neq i' \tag{A.11}$$

$$d_{ikh} + \sum_{k'=0}^{K-1} \sum_{h'=\max(h,h+L_{ik}^{O}-\mathrm{Se}_{f}+1)}^{\min(H-1,h+L_{ik}^{O}+\mathrm{Se}_{f}-1)} d_{i'k'h'} \leq 1 \quad \forall f,i,i',k,h : f \in \mathrm{CT}_{17} \wedge i,i' \in A_{f} \wedge i \neq i' \tag{A.12}$$

$$d_{ikh} + \sum_{k'=0}^{K-1} \sum_{h'=\max(h,h+L_{ik}^{O}-\mathrm{Se}_{f}+1)}^{\min(H-1,h+L_{ik}^{O}+\mathrm{Se}_{f}-1)} d_{i'k'h'} \leq 1 \quad \forall f,i,i',k,h : f \in \mathrm{CT}_{18} \wedge i,i' \in A_{f} \wedge i \neq i' \tag{A.13}$$

$$\sum_{i\in A_f}\sum_{k=0}^{K-1}\sum_{e=0}^{\min(h-\mathrm{ST}_{ikf},\mathrm{TU}_{ikf})} d_{ik,h-\mathrm{ST}_{ikf}-e} \le Q_f \quad \forall f,h : f \in \mathrm{CT}_{19} \tag{A.14}$$

$$\sum_{i\in A_f(h)}\sum_{k=0}^{K-1}\sum_{h'=\max(0,h-L^O_{ik}+1)}^{\min(h,H-1)} d_{ikh'} \le E_{fh} \quad \forall f,h : f \in \mathrm{CT}_{20} \tag{A.15}$$

$$\sum_{i\in C_f}\sum_{k=0}^{K-1}\sum_{h'=\max(0,h-L^O_{ik}+1)}^{h} d_{ikh'}\cdot P^{\max}_{it} \le O^{\max}_f \quad \forall f,h,t : f \in \mathrm{CT}_{21} \wedge h \in \mathbb{W}_t \wedge t \in W_h \tag{A.16}$$

Constraints (A.1) correspond to [CT7] and imposes bounds on reload performed during cycle k of plant i. Constraints (A.2) correspond to [CT8] and defines the initial fuel stock of plant i.

Constraints (A.3) model the variation of fuel stock during a production campaign [CT9]. They work by imposing a lower level for stock at the end of cycle $k-1$, that is, before refueling at cycle k, on the basis of the decoupling decisions. As shown in figure A.1 the constraints say that, if cycle k and $k-1$ are executed, then the fuel stock before refueling k has to be greater than the fuel stock after refueling $k-1$ minus the maximum fuel consumption during cycle $k-1$. If cycle k is not executed but $k-1$ is, the constraints are dominated by the non negativity constraint for x^B_{ik}. With the typical input, constraints (A.3) also impose that if cycle $k-1$ is not executed then $d_{i,k-1,H}=1$ and therefore also k is not executed, in order to satisfy the bound $x^B_{ik} \le A^{\max}_{ik}$. Together with the lower and upper bound on stocks before (A.5) and after (A.6) refueling[CT11], the constraints are also used to favour the correct sequencing for outages as specified by [CT13] and the fact that if cycle $k-1$ is not executed then also k is not executed.

Constraints (A.4) are the fuel stock variation during an outage described by [CT10].

Constraints (A.7) require that each outage must be done within its allowed weeks, thus imposing [CT13] while (A.8) ensure that for each plant i, in each week h no more than one decoupling is performed. If an outage is not planned in the scheduling horizon than it is planned in week H.

Constraints (A.9) corresponds to [CT14]: they state that the outages of set A_m have to be spaced by at least Se_m weeks. If Se_m is negative it represents the maximum authorized overlapping period. As shown in figure A.2, if plant i goes offline in week h, than no other plant i' from the same set can go offline for Se_m weeks after plant i has finished its outage.

Constraints (A.10) specify the minimum spacing between outages of set A_m

during a specific period $[I_m, F_m]$. They correspond to [CT15]. The way they work is illustrated in figure A.3: if a plant i goes offline in a week h such that its outage enters the period $[I_m, F_m]$, then no other plant i' can start the outage for Se_m after the end of the outage of plant i.

Constraints (A.11) impose a minimum spacing between decoupling dates of all plants in set A_m. They correspond to [CT16]. As shown in figure A.4, if plant i decouples in week h, then any other plant i' in A_m cannot decouple before $h + Se_m$.

Constraints (A.12), corresponding to [CT17] specify a minimum spacing between dates of coupling. The way the constraints work is depicted in figure A.5: if a plant i couples in week h then no other plant i' from the same set A_m can couple before Se_m weeks.

Constraints (A.13) specify a minimum spacing between coupling and decoupling dates [CT18]. As shown in figure A.6, if a plant i decouples in week h, then any other plant i' in the same set A_m has to decouple Se_m weeks before i couples (week $h + L^O_{ik}$) or Se_m after.

Constraints (A.14) are about resources [CT19]. Consider any week h: then lesser then Q_m

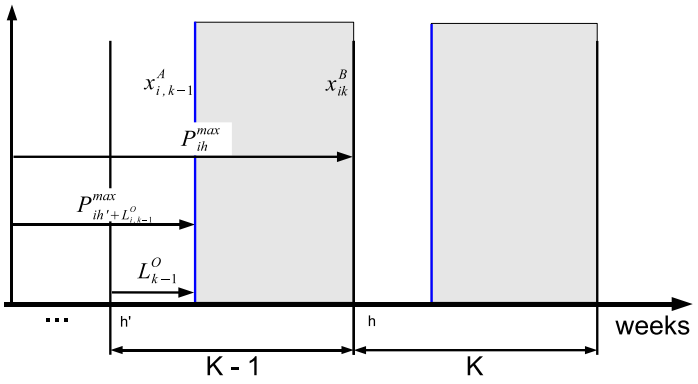


Figure A.1. Contraints (A.3): linking fuel stock and outage scheduling

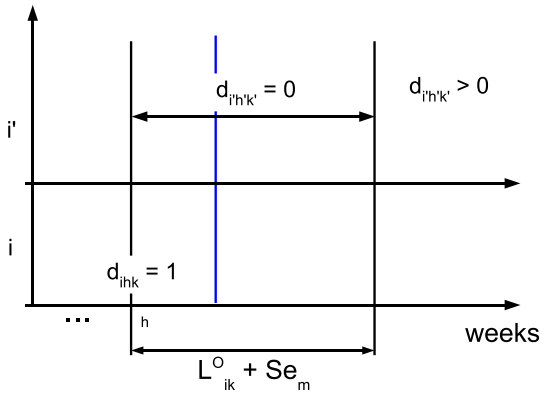


Figure A.2. Constraints (A.9): minimum spacing/maximum overlapping between outages

A.2 Details of the Weekly Production Planner module

A schedule d obtained by HOSG or by LS only satisfies the feasibility of outage scheduling conditions but completely disregards demand and costs. The step performed by WPP aims at determining a rough approximation of the plant production, refueling and consequent costs due to a fixed outage schedule. In particular, WPP solves a linear programming model, which explicitly considers all the scenarios but takes planning decisions still on a weekly base, and determines a correspondent feasible refueling.

In the linear programming model adopted in WPP some simplifications are introduced in order to reduce the problem dimension. In particular, similarly to the MIP model used in HOGS, only two stock level variables are considered for each cycle; in addition, all T1 plants are assumed as aggregated in a single “macro” plant, whose production power cost is computed as average of the costs of those T1 plants with lower cost that would be able to satisfy the average weekly demand with their upper bound production. Note that this cost estimation is more realistic than an overall average as it is assumed that "cheaper" T1 plants should be used first to satisfy the demand.

Then, in WPP all the technical constraints from [CT1] to [CT12] are included at least in an approximated way, with the exception of the imposition of power [CT6]profile that are still disregarded. In particular, this is straightforwardly done for [CT1] to [CT5], being fixed the outage schedule, and [CT7]

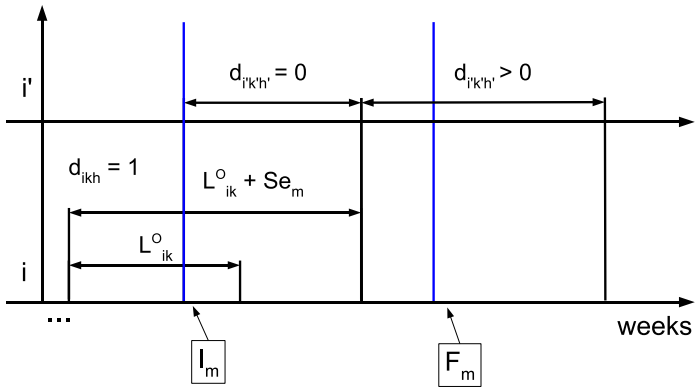


Figure A.3. Constraints (A.10): minimum spacing/maximum overlapping between outages during a specific period $[I_m, F_m]$

and [CT8]. Conditions [CT12] on maximum allowed modulation are imposed in an approximated way by estimating for each production campaign (fixed in input by d) the time interval (weeks) during which the T2 plants production can be performed at the upper bound level as well as the week after which in a production campaign T2 plants production should be zero due to zero stock level. Constraints on modulation and power output for T2 plants are then imposed accordingly.

Details about the model follow.

Input The WPP is fed in input the following data

- the decoupling weeks calculated by MIP, see the appendix section A.1, in form of binary variable d_{ikh}
- C_{jts}, the proportional production cost for plant j in scenario s in timestep t
- D_{ts}, the demand in each timestep and in each scenario

Sets and indices

- $j \in \mathbb{J} \triangleq \{0, \ldots, J-1\}$ is the index of type 1 power plants
- $s \in \mathbb{S} \triangleq \{0, \ldots, S-1\}$ is the index of scenarios

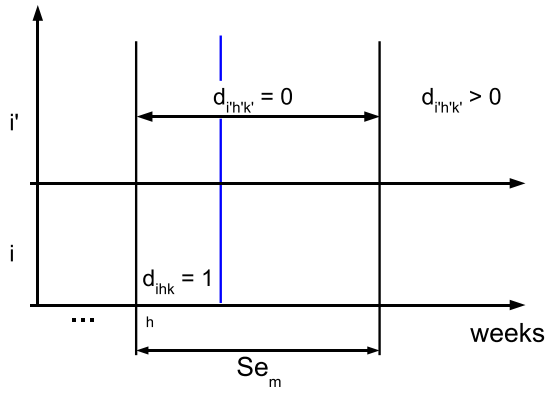


Figure A.4. Constraints (A.11): minimum spacing between decoupling dates

The other indices are the same specified for the MIP model, see the appendix section A.1

Data preparation

- the demand is aggregated on weeks: $D_{hs} \triangleq \sum_{t \in W_h} D_{ts}$, where D_{ts} is the demand for timestep t in scenario s

- the average proportional production cost for plant j in scenario s during week h is C_{jhs} and is calculated as the sum of the proportional production costs in the timesteps of the week: $C_{jhs} = \frac{1}{|W_h|} \sum_{t \in W_h} C_{jts}$

- The weeks of decoupling W_{ik}^D are simply calculated from d_{ikh} according to algorithm 3.

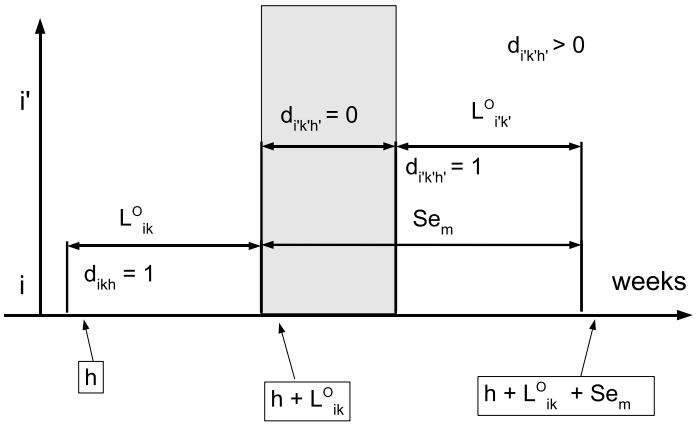


Figure A.5. Constraints (A.12): minimum spacing between coupling dates

Algorithm 3 Outage week calculation

```
for all i ∈ 𝕀 do
    k ← 0
    for all h ∈ ℍ do
        if ( d_ikh > 0 ) then
            W^D_ik ← h
            k ← k + 1
        end if
    end for
end for
```

- $P^{\min}_{jhs} \triangleq \sum_{t \in W_h} P^{\min}_{jts}$ is the minimum power output of plant j during week h in scenario s
- $P^{\max}_{jhs} \triangleq \sum_{t \in W_h} P^{\max}_{jts}$ is the maximum power output of plant j during week h in scenario s
- $P1^{\min}_{hs} \triangleq \sum_{j \in \mathbb{J}} \sum_{t \in W_h} P^{\min}_{jhs}$ is the minimum power output for the "macro" plant
- $P1^{\max}_{hs} \triangleq \sum_{j \in \mathbb{J}} \sum_{t \in W_h} P^{\max}_{jhs}$ is the maximum power output for the "macro" plant

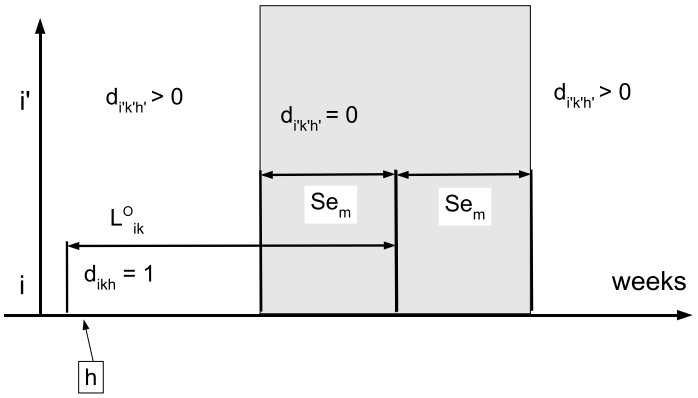


Figure A.6. Constraints (A.13): minimum spacing between coupling and decoupling dates

- The cost of the "macro" T1 plant is calculated according to the following procedure:

  1. $\tilde{p}_{jhs}$, the power output of each type 1 plant j in scenario s in week h is initialized to the minimum output $P^{\min}_{jhs}$ and, for each scenario, for each week, type 1 plants are sorted in increasing order of cost of production C_{jsh}
  2. in order to satisfy the demand using only T1 plants, for all weeks, for all scenarios, $\tilde{p}_{jhs} = P^{\max}_{jhs}$ for the next plant with the cheapest cost C_{jhs}. This step is iterated until the demand is completely satisfied. Note that with the analysed datasets it is always possible to satisfy the demand using only T1 plants. Of course, if any week h and scenario s exist, such that the total power output exceeds the demand, then the power of last added plant j_l is reduced.
  3. the equivalent cost of production C^1_{hs} is calculated according to the following definition:

$$C^1_{hs} \triangleq \frac{\sum_{j\in\mathbb{J}} C_{jhs} \cdot \tilde{p}_{jhs}}{L^{\mathbb{H}}_h} \tag{A.17}$$

- $P^{\max}_{ih} \triangleq \sum_{t\in W_h} P^{\max}_{it}$ is the maximum power output for type 2 plant i

- constraints [CT12] limit the maximum modulation when the fuel stock is above a threshold B_{ik}. To avoid the use of an excessive number of binary variables, the following simplified strategy has been adopted:

  1. the stock of fuel, in the first week of each cycle of each plant, is initialized at the minimum refuel plus the maximum modulation. Under these conditions, with the hypothesis of keeping the power output at the upper bound, it is easy to calculate the week when the fuel stock of current plant goes under the threshold BO_{ik}. That week is called $N_{ik}^{\min}$. It is, in the worst case, the week up to which it is certainly possible to keep production at the maximum level.
  2. The fuel stock at the first week of the cycle of each plant is initialized as $x_{ik}^{B} = \min\left(A_{ik}^{\max} + R_{ik}^{\max}, S_{ik}^{\max}\right) + M_{ik}^{\max}$ and the power output is kept at the maximum allowed. $N_{ik}^{\max}$ is the week when the fuel stock goes to zero.

Constants

- $L_h^{\mathbb{H}}$ is the duration of week h
- C_{hs}^{1} is the proportional equivalent cost of production for "macro" plant T1 during weeh h in scenario s
- C_{iH} is the proportional cost of last refuel
- C_{ik} is the proportional cost of fuel during cycle k
- D_{hs} is the aggregated demand on week h in scenario s
- $P1_{hs}^{\min}$ is the minimum power output for the "macro" plant
- $P1_{hs}^{\max}$ is the maximum power output for the "macro" plant
- W_{ik}^{D}, the decoupling week for plant i in cycle k
- L_{ik}^{O} is the length of outage k for plant i
- $R_{ik}^{\min}$ and $R_{ik}^{\max}$ are respectively the minimum and maximum reload for plant i in cycle k
- X_i^0 is the initial fuel stock of plant i

- k_i^* is the last scheduled cycle for plant i
- R_{ik}^r is the refueling coefficient
- $S_{ik}^{\max}$ and $A_{ik}^{\max}$ are the maximum bounds on stock of fuel respectively during production campaign and at the time of outage of cycle k
- $P_{ih}^{\max}$, the maximum power output of plant i during week h
- $M_{ik}^{\max}$ is the maximum modulation of plant i during cycle k

Decision Variables

- p_{hs}^1 is the power output of "macro" type 1 plant in week h in scenario s
- p_{ihs}^2 is the power output of type 2 power plant i in week h in scenario s
- $x_{ik}^B \in \mathbb{R}_+$ is the remaining stock of fuel in plant i at the end of cycle $k-1$, that is to say right before cycle k
- $x_{ik}^A \in \mathbb{R}_+$ is the initial stock of fuel in plant i at the beginning of cycle k, that is to say right after cycle $k-1$
- $r_{ik} \in \mathbb{R}_+$ is the refueling of plant i in cycle k
- x_{is}^f is the final fuel stock in plant i in scenario s

Linear program

$$\min \frac{1}{|S|} \sum_{s\in\mathbb{S}} \sum_{h\in\mathbb{H}} \left(p_{hs}^1 \cdot L_h^{\mathbb{H}} \cdot C_{hs}^1 - \sum_{i\in\mathbb{I}} C_{iH} \cdot x_{is}^f \right) + \sum_{i\in\mathbb{I}} \sum_{k\in\mathbb{K}} C_{ik} \cdot r_{ik} \tag{A.18}$$

subject to:

$$p_{hs}^1 + \sum_{i\in\mathbb{I}} p_{ihs}^2 = D_{hs} \quad \forall h,s \tag{A.19}$$

$$P1_{hs}^{\min} \le p_{hs}^1 \le P1_{hs}^{\max} \quad \forall h,s \tag{A.20}$$

$$p_{ihs}^2 = 0 \quad \forall i,k,h : h \in \left[W_{ik}^D, W_{ik}^D + L_{ik}^O\right] \tag{A.21}$$

$$p_{ihs}^2 \le P_{ih}^{\max} \quad \forall i,k,h : h \notin \left[W_{ik}^D, W_{ik}^D + L_{ik}^O\right] \tag{A.22}$$

$$R_{ik}^{min} \le r_{ik} \le R_{ik}^{\max} \quad \forall i,k : W_{ik}^D \neq -1 \tag{A.23}$$

$$x_{i,-1}^A = X_i^0 \quad \forall i \tag{A.24}$$

$$x_i^f = x_{i,k_i^*}^B \quad \forall i \tag{A.25}$$

$$x_{iks}^B = x_{i,k-1,s}^A - \sum_{h=W_{i,k-1}^D+L_{i,k-1}^O}^{W_{ik}^D} p_{ihs}^2 \cdot L_h^{\mathbb{H}} \quad \forall i,k,s \tag{A.26}$$

$$x_{iks}^A = R_{ik}^r \cdot \left(x_{iks}^B - B_{i,k-1}\right) + r_{ik} + B_{ik} \quad \forall i,k,s \tag{A.27}$$

$$x_{iks}^B \leq A_{ik}^{\max} \quad \forall i,k,s \tag{A.28}$$

$$x_{iks}^A \leq S_{ik}^{\max} \quad \forall i,k,s \tag{A.29}$$

$$\sum_{h=W_{ik}^D}^{N_{ik}^{\min}} \left[P_{ih}^{\max} - p_{ihs}^2 \cdot L_h^{\mathbb{H}}\right] \leq M_{ik}^{\max} \quad \forall i,k,s \tag{A.30}$$

$$p_{ihs}^2 = 0 \quad \forall i,k,h,s : N_{ik}^{\max} \leq h \leq W_{ik}^D + L_{ik}^O \tag{A.31}$$

Constraints (A.19) couple load and production [CT1]. Constraints (A.20) bound the production of macro plant [CT2]. Constraints (A.21) and (A.22) bound power output of T2 plants when the power profile is not imposed, as required by [CT3], [CT4] and [CT5]. Constraints (A.23) are bounds on reload [CT7]. Constraints (A.24) set the initial fuel stock [CT8], while (A.25) sets the final fuel stock. Constraints (A.26) models the fuel consumption due to production [CT9]. Constraints (A.27) is the fuel stock variation during an outage [CT10]. Constraints (A.28) and (A.29) bound the fuel stock [CT11]. Constraints (A.30) and (A.31) approximate the constraints on maximum modulation [CT12]. The bounds $N_{ik}^{\min}$ and $N_{ik}^{\max}$ are respectively the last week where the production can certainly be kept at the upper level and the first week when the production must be set to zero. Their calculation is explained above in section A.2. Note that in the intermediate weeks $h \in (N_{ik}^{\min}, N_{ik}^{\max})$ the power output is quite free as it is bounded only by the maximum and minium level but neither the maximum modulation constraints [CT12] nor the power profile are applied [CT6].

Output The output of WPP are refueling levels r_{ik} and also a set of reference levels mod_{ih} for T2 plants modulation in each week. These levels are obtained in a post processing phase according to the following definition:

$$\text{mod}_{iks} = \min\left[\left(\sum_{h\in\left[W_{ik}^D, W_{ik}^D+L_{ik}^O\right]} P_{ih}^{\max} - p_{ikh}^2\right) \cdot L_h^{\mathbb{H}},\ M_{i,k}^{\max}\right] \tag{A.32}$$

A.3 Details of the Timestep Production Planner module

The purpose of TSPP is to completely determine a detailed production plan for T1 and T2 plants, with reference to an input schedule for outages. TSPP considers as input a given outage schedule d and takes advantage of the WPP optimization using the refueling r and the levels of modulation as reference. TSPP is an iterative procedure based on a linear programming model which decomposes the overall problem in the sequential optimization of one scenario at a time. Therefore, all the schedule, refueling and modulation data are used as a way for coordinating as much as possible the planning over the set of scenarios.

Note that considering the above input and the modulation level as fixed, for each plant and scenario the stock levels in timesteps are directly computed. Therefore, no stock level variables are used in the LP model adopted in TSPP. [CT6] are modelled as upper and lower bound on $P2_{ish}$ variables that are forced to follow the imposed decreasing power profile. However, before starting the iterative scenario optimization a refueling feasibility check procedure (which basically verifies [CT7] and [CT11]) is used in order to verify at timestep level the correctness of the input r with the given outage schedule and modulation references. In case of infeasibility such procedure appropriately reduces the refueling values. The linear model used in the TSPP iterations basically includes the [CT1]–[CT5] constraints as all the other constraints should be satisfied by the previous HOSG, LS and WPP phases or by the refueling feasibility procedure. TSPP then determines for each single scenario the power production for T1 and T2 plants. In the TSPP model, deviational variables are introduced to allow variations (specifically, over or under achievements) with respect to the reference modulation levels. Therefore, the LP optimization objective is the minimization of the T1 plant production cost referred to a single scenario with the addition of a penalization for positive deviations. If in the solution for a scenario at least one deviational variable assumes positive value then this means that for such a scenario the modulation reference must be updated and the optimization for this scenario is re-executed. At the end of an iteration (i.e., after solving the LP for all the scenarios) if any modulation reference is changed, then the refueling feasibility is checked again and in case refueling levels are revised a new iteration is started.

Note that, as the violation of stock level bounds before refueling is the main source of infeasibility for the imposed refueling level, a reduction of refueling is generally expected to overcome the problem. In addition, if in a feasible outage schedule the production campaigns have a not too short duration (as it should be in the real application context), the minimum refueling level should allow

determining an overall feasible power production plan. Hence, in the same operating conditions the iterative procedure executed in TSPP is expected to converge to a feasible overall solution. The experiments support this thesis.

Procedure Outline Algorithm 4 presents an outline of the entire block.

Algorithm 4 Timestep Production Planner

```
1: correct stock
2: for all scenarios s do
3:     repeat
4:         repeat
5:             populate and solve LP
6:             for all i, k do
7:                 if ( Δ⁺_ik > 0 ) then
8:                     mod_ik ← mod_ik + Δ⁺_ik
9:                 else if ( Δ⁻_ik > 0 ) then
10:                    mod_ik ← mod_ik − Δ⁻_ik
11:                end if
12:            end for
13:        until Δ⁺_ik = 0 ∧ Δ⁻_ik = 0
14:    until no stock corrections are necessary
15: end for
```

The correction of the stock is executed at the beginning as an initialization step, and then is called again at the end of each loop. The procedure receives in input the outage dates, the modulation (the first time it is given exactly the levels calculated by WPP) and the refuelings. Its behaviour is outlined in procedure 5, which calls procedures 6 and 7 to verify - and if necessary to repair - the value of fuel stock respectively before and after refueling.

Algorithm 5 Fuel Stock Correction

modified $\leftarrow$ false
for all $i \in \mathbb{I}$ **do**
    restart $\leftarrow$ false
    for all $s \in \mathbb{S}$ **do**
        $x \leftarrow X_i^0$
        $t \leftarrow 0$
        for all $k \in \mathbb{K}$ **do**
            $M_{ik} \leftarrow \min\left(\text{mod}_{iks}, M_{ik}^{\max}\right)$
            if ($x > B_{ik}$) **then**
                while $t < \text{end}_{ik} \wedge x_{its} > B_{ik}$ **do**
                    $x \leftarrow x - P_{ik}^{\max} \cdot L_t^{\mathbb{T}}$
                    $t \leftarrow t + 1$
                end while
                $x \leftarrow x + M_{ik}$
            end if
            while $t < \text{end}_{ik}$ **do**
                $p \leftarrow \text{prof}(x_{its}) \cdot P_{ik}^{\max}$
                if ($x \leq p \cdot L_t^{\mathbb{T}}$) **then**
                    $p \leftarrow 0$
                end if
                $x \leftarrow x - p \cdot L_t^{\mathbb{T}}$
                $t \leftarrow t + 1$
            end while
            if ($k < K \wedge W_{ik}^D < H$) **then**
                { modified, restart } $\leftarrow$ repairBefore(...)
                $x \leftarrow R_{ik}^r \cdot \left(x - B_{ik}\right) + r_{ik} + B_{ik}$
                $t \leftarrow \text{end}_{ik}$
                {modified, restart } $\leftarrow$ repairAfter(...)
            end if
        end for
    end for
end for
return modified

Algorithm 6 Repair Stock Before Refueling

if $x > A_{ik}^{\max}$ **then**
  repeat
    if ($r_{i,k-1} > R_{i,k-1}^{\min}$) **then**
      $r_{i,k-1} \leftarrow \min\left(r_{i,k-1} - \left(x - A_{i,k-1}^{\max}\right), R_{i,k-1}^{\min}\right)$
    else
      $k \leftarrow k - 1$
    end if
  until $k <= 0$
  modified $\leftarrow$ true
  restart $\leftarrow$ true
end if
return {modified, restart}

Algorithm 7 Repair Stock After Refueling

if (restart = false $\wedge$ $x > S_{ik}^{max}$) **then**
  if ($k > 0 \wedge r_{i,k-1} > R_{i,k-1}^{\min}$) **then**
    $r_{i,k-1} \leftarrow r_{i,k-1} - \tau\left(x - S_{ik}^{\max}\right)$
  end if
  if ($r_{ik} > R_{ik}^{\min}$) **then**
    $r_{ik} \leftarrow r_{ik} - \tau\left(x - S_{ik}^{\max}\right)$
  end if
  if ($k > 0 \wedge r_{i,k-1} < R_{i,k-1}^{\min}$) **then**
    $r_{i,k-1} \leftarrow R_{i,k-1}^{\min}$
  end if
  $r_{ik} \leftarrow \max\left(R_{ik}^{\min}, r_{ik}\right)$
  modified $\leftarrow$ true
  restart $\leftarrow$ true
end if
return {modified, restart}

The parameter $\tau \in (0, 1]$ used in algorithm 7, has been set to $\frac{1}{10}$ in our experiments.

Sets and indices

- $s \in \mathbb{S}$ is the current scenario being optimized.

- $\mathbb{T}^O_{ik}$ is the set of timesteps in the outage of cycle k of plant i
- $\mathbb{T}^P_{ik}$ is the set of timesteps in the production campaign of cycle k of plant i

The sets and indices not defined here have already been defined in the previous sections.

Constants

- C_{jts} is the proportional production cost for plant j in time step t in scenario s
- $L^{\mathbb{T}}_t$ is he length of timestep t
- M is a big-M, a constant greater than any other constant in the model
- D_{ts} is the demand in timestep t in scenario s
- $P^{\min}_{jts}$ and $P^{\max}_{jts}$ are the bound on production of T1 plant j in timestep t in scenario s_c
- $P^{\max}_{ik}$ is the maximum power output of T2 plant i in cycle k
- x_{its} is the fuel stock of plant i in timestep t in current scenario
- $\text{prof}(x)$ is a scale factor that, multiplied by the maximum power output, allows to define a power profile. It is assumed that when the stock is small enough then the profile is zero: $\text{prof}(x) = 0 \;\; \forall x \leq \epsilon_x$

Decision Variables

- $p^{1s}_{jt} \in \mathbb{R}_+$, the production of type 1 power plant j in timestep t in scenario s
- $p^{2s}_{it} \in \mathbb{R}_+$, the production of type 2 power plant i in timestep t in scenario s
- $\Delta^+_{ik} \in \mathbb{R}_+$, the deviational variable for positive violation of the reference modulation for plant i in scenario k
- $\Delta^-_{ik} \in \mathbb{R}_+$, the deviational variable for negative violation of the reference modulation for plant i in scenario k

Linear program

$$\min \frac{1}{S} \sum_{j} \sum_{t} \left[C_{jts} \cdot L_t^{\mathbb{T}} \cdot p_{jt}^{1s} \right] + M \cdot \sum_{i} \sum_{k} \left[\Delta_{ik}^{+} + \Delta_{ik}^{-} \right] \tag{A.33}$$

subject to:

$$\sum_{j \in \mathbb{J}} p_{jt}^{1s} + \sum_{i \in \mathbb{I}} p_{it}^{2s} = D_{ts} \quad \forall t \tag{A.34}$$

$$P_{jts}^{\min} \leq p_{jt}^{1s} \leq P_{jts}^{\max} \quad \forall t, j \tag{A.35}$$

$$p_{it}^{2s} = 0 \quad \forall i, k, t : t \in \mathbb{T}_{ik}^{O} \tag{A.36}$$

$$p_{it}^{2s} \leq P_{it}^{\max} \quad \forall t, i, k : t \in \mathbb{T}_{ik}^{P} \tag{A.37}$$

$$\sum_{t \in \mathbb{T}_{ik}^{P}} p_{it}^{2s} \cdot L_t^{\mathbb{T}} + \Delta_{ik}^{+} - \Delta_{ik}^{-} = \sum_{t \in \mathbb{T}_{ik}^{P}} P_{ik}^{\max} \cdot L_t^{\mathbb{T}} - \mathrm{mod}_{ik}^{\max} \quad \forall i, k \tag{A.38}$$

$$\Delta^{+} \leq \max \left(M_{ik}^{max} - \mathrm{mod}_{ik}^{\max}, 0 \right) \quad \forall i, k \tag{A.39}$$

$$\Delta^{-} \leq M_{ik}^{max} \quad \forall i, k \tag{A.40}$$

$$p_{it}^{2s} = \mathrm{prof}\left(x_{its} \right) \cdot P_{ik}^{\max} \quad \forall i, k, t : t \in \mathbb{T}_{ik}^{P} \wedge x_{it} < B_{ik} \tag{A.41}$$

The optimization objective (A.33) is the minimization of the T1 plants production cost referred to the current scenario s_c being optimized, plus a penalization for reference modulation violations.

Bibliography

Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L. M., Poss, M. and Requejo, C. [2013]. The robust vehicle routing problem with time windows, *Computers & Operations Research* **40**(3): 856–866.

Ahmad, A. and Kothari, D. P. [2000]. A practical model for generator maintenance scheduling with transmission constraints, *Electric Power Components and Systems* **28**: 501–513.

Anghinolfi, D., Gambardella, L. M., Montemanni, R., Nattero, C., Paolucci, M. and Toklu, N. E. [2011a]. Addendum to: "A matheuristic algorithm for a large-scale energy management problem", `http://www.idsia.ch/~roberto/roadef_addendum.pdf`.

Anghinolfi, D., Gambardella, L. M., Montemanni, R., Nattero, C., Paolucci, M. and Toklu, N. E. [2011b]. An optimization algorithm for a large-scale energy management problem, *Proceedings of CAR 2011 - The Annual International Conference on Control, Automation and Robotics*, pp. C22–C27.

Anghinolfi, D., Gambardella, L. M., Montemanni, R., Nattero, C., Paolucci, M. and Toklu, N. E. [2012]. *Lecture Notes in Computer Science*, Vol. 7116, Springer, Heidelberg, chapter "A matheuristic algorithm for a large-scale energy management problem", pp. 173–181.

Artigues, C., Leus, R. and Nobibon, F. T. [2013]. Robust optimization for resource-constrained project scheduling with uncertain activity durations, *Flexible Services and Manufacturing Journal* **25**(1-2): 175–205.

Baldacci, R., Christofides, N. and Mingozzi, A. [2008]. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts, *Mathematical Programming* **115**(2): 351–385.

Baldacci, R., Toth, P. and Vigo, D. [2010]. Exact algorithms for routing problems under vehicle capacity constraints, *Annals of Operations Research* **175**(1): 213–245.

Barbarosoğlu, G. and Özgür, D. [1999]. A tabu search algorithm for the vehicle routing problem, *Computers & Operations Research* **26**(3): 255–270.

Ben-Tal, A. and Nemirovski, A. [1997]. Robust truss topology design via semidefinite programming, *SIAM Journal on Optimization* **7**(4): 991–1016.

Ben-Tal, A. and Nemirovski, A. [1999]. Robust solutions of uncertain linear programs, *Operations Research Letters* **25**(1): 1–13.

Ben-Tal, A. and Nemirovski, A. [2000]. Robust solutions of linear programming problems contaminated with uncertain data, *Mathematical Programming* **88**(3): 411–424.

Bertsimas, D. and Nohadani, O. [2010]. Robust optimization with simulated annealing, *Journal of Global Optimization* **48**(2): 323–334.

Bertsimas, D. and Sim, M. [2003]. Robust discrete optimization and network flows, *Mathematical Programming* **98**(1): 49–71.

Bertsimas, D. and Sim, M. [2004a]. The price of robustness, *Operations Research* **52**(1): 35–53.

Bertsimas, D. and Sim, M. [2004b]. Robust discrete optimization under ellipsoidal uncertainty sets, *Technical report*, Sloan School of Management, Massachusetts Institute of Technology.

Birge, J. R. and Louveaux, F. [1997]. *Introduction to stochastic programming*, Springer Verlag.

Bock, F. [1958]. An algorithm for solving "traveling-salesman" and related network optimization problems, unpublished, presented at the 14th ORSA National Meeting.

Campbell, A., Gendreau, M. and Thomas, B. [2011]. The orienteering problem with stochastic travel and service times, *Annals of Operations Research* **186**(1): 61–81.

Croes, G. A. [1958]. A method for solving traveling-salesman problems, *Operations Research* **6**(6): 791–812.

Dantzig, G. B. [1963]. Linear programming and its extensions. Princeton University Press, Princeton, NJ.

Dantzig, G. B. and Ramser, J. H. [1959]. The truck dispatching problem, *Management Science* **6**(1): 80–91.

Data Instances of ROADEF/EURO 2010 Challenge [2010]. `http://challenge.roadef.org/2010/instances.en.htm`.

Deb, K., Gupta, S., Daum, D., Branke, J., Mall, A. and Padmanabhan, D. [2009]. Reliability-based optimization using evolutionary algorithms, *IEEE Transacations on Evolutionary Computation* **13**(5): 1054–1074.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. [2002]. A fast and elitist multi-objective genetic algorithm: Nsga-ii, *IEEE Transaction on Evolutionary Computation* **6**(2): 181–197.

Dopazo, J. F. and Merrill, H. M. [1975]. Optimal generator maintenance scheduling using integer programming, *IEEE Transactions on Power Apparatus and Systems* **PAS-94**(5): 1537–1545.

Dorigo, M. [1992]. *Learning and Natural Algorithms*, PhD thesis, Politecnico di Milano.

Dorigo, M., Maniezzo, V. and Colorni, A. [1991]. Positive feedback as a search strategy, *Technical Report 91-016*, Dipartimento di Elettronica, Politecnico di Milano.

Easa, S. [1985]. Trade-off of gradation and cost requirements in aggregate blending, *Journal of Cement, Concrete, and Aggregates* **7**(1): 29–36.

Easa, S. M. and Can, E. K. [1985a]. Optimization model for aggregate blending, *Journal of Construction Engineering and Management* **111**(3): 216–230.

Easa, S. M. and Can, E. K. [1985b]. Stochastic priority model for aggregate blending, *Journal of Construction Engineering and Management* **111**(4): 358–373.

El Ghaoui, L. and Lebret, H. [1997]. Robust solutions to least-squares problems with uncertain data, *SIAM Journal on Matrix Analysis and Applications* **18**(4): 1035–1064.

El Ghaoui, L., Oustry, F. and Lebret, H. [1998]. Robust solutions to uncertain semidefinite programs, *SIAM Journal on Optimization* **9**(1): 33–52.

Foong, W. K., Maier, H. R., Simpson, A. R. and Stolp, S. [2008]. Ant colony optimization for power plant maintenance scheduling optimization, *Annals of Operations Research* **159**(1): 433–450.

Gambardella, L. M., Taillard, É. and Agazzi, G. [1999]. *New Ideas in Optimization*, McGraw-Hill, chapter "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows", pp. 63–76.

Gendreau, M., Hertz, A. and Laporte, G. [1994]. A tabu search heuristic for the vehicle routing problem, *Management science* **40**(10): 1276–1290.

Gil, E., Bustos, J. and Rudnick, H. [2003]. Short-term hydrothermal generation scheduling model using a genetic algorithm, *IEEE Transactions on Power Systems* **18**(4): 1256–1264.

Glover, F. and Laguna, M. [1997]. *Tabu Search*, Kluwer Academic Publishers Norwell, MA, USA.

GNU Project [2014]. GLPK (GNU Linear Programming Kit), `http://www.gnu.org/software/glpk/`. Online; March-2014.

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

Golden, B. L., Raghavan, S. and Wasil, E. A. [2008]. *The vehicle routing problem: latest advances and new challenges*, Vol. 43, Springer.

Gunawan, S. and Azarm, S. [2005]. Multi-objective robust optimization using a sensitivity region concept, *Structural and Multidisciplinary Optimization* **29**(1): 50–60.

Guo, S. and Yang, O. W. W. [2007]. Energy-aware multicasting in wireless ad hoc networks: A survey and discussion, *Computer Communications* **30**(9): 2129–2148.

Gurobi [2014]. Gurobi Optimization, `http://www.gurobi.com`. Online; March-2014.

Hartmann, S. and Briskorn, D. [2010]. A survey of variants and extensions of the resource-constrained project scheduling problem, *European Journal of Operational Research* **207**(1): 1–14.

Herroelen, W. and Leus, R. [2005]. Project scheduling under uncertainty: Survey and research potentials, *European Journal of Operational Research* **165**(2): 289–306.

Holland, J. H. [1975]. *Adaptation in natural and artificial systems*, University of Michigan press, Ann Arbor.

IBM CPLEX [2014]. CPLEX Optimizer, `http://www.cplex.com`. Online; March-2014.

Johnson, D. S. and McGeoch, L. A. [1997]. The traveling salesman problem: A case study in local optimization, *Local search in combinatorial optimization* **1**: 215–310.

Karaşan, O. E., Pınar, M. and Yaman, H. [2001]. The robust shortest path problem with interval data, *Technical report*, Department of Industrial Engineering, Bilkent University.

Kirkpatrick, S., Gelatt, C. G. and Vecchi, M. P. [1983]. Optimization by simulated annealing, *Science* **220**(4598): 671–680.

Kouvelis, P. and Yu, G. [1997]. *Robust discrete optimization and its applications*, Kluwer Academic Publishers, Norwell, MA.

Laporte, G. [1992]. The vehicle routing problem: An overview of exact and approximate algorithms, *European Journal of Operational Research* **59**(3): 345–358.

Lee, C., Lee, K. and Park, S. [2012]. Robust vehicle routing problem with deadlines and travel time/demand uncertainty, *Journal of the Operational Research Society* **63**(9): 1294–1306.

Lee, D. [1973]. Review of aggregate blending techniques, *Highway Research Record* **441**: 111–127.

Lee, S. M. and Olson, D. L. [1983]. Chance constrained aggregate blending, *Journal of Construction Engineering and Management* **109**(1): 39–47.

Leggieri, V., Nobili, P. and Triki, C. [2008]. Minimum power multicasting problem in wireless networks, *Mathematical Methods of Operations Research* **68**: 295–311.

Li, M., Azarm, S. and Aute, V. [2005]. A multi-objective genetic algorithm for robust design optimization, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation (GECCO'05)*, ACM, New York, NY, USA, pp. 771–778.

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

Maniezzo, V., Stützle, T. and Voß, S. [2009]. *Matheuristics: hybridizing metaheuristics and mathematical programming*, Vol. 10, Springer.

Mladenović, N. and Hansen, P. [1997]. Variable neighborhood search, *Computers & Operations Research* **24**(11): 1097–1100.

Mojana, M., Montemanni, R., Di Caro, G. and Gambardella, L. M. [2011]. An algorithm combining linear programming and an ant system for the sequential ordering problem, *Proceedings of ATAI 2011 - The Second Annual International Conference on Advanced Topics in Artificial Intelligence*, pp. 80–85.

Montemanni, R. [2007]. A mixed integer programming formulation for the total flow time single machine robust scheduling problem with interval data, *Journal of Mathematical Modelling and Algorithms* **6**(2): 287–296.

Montemanni, R. and Gambardella, L. M. [2005]. The robust shortest path problem with interval data via benders decomposition, *4OR: A Quarterly Journal of Operations Research* **3**(4): 315–328.

Montemanni, R., Gambardella, L. M. and Das, A. K. [2005]. The minimum power broadcast problem in wireless networks: a simulated annealing approach, *IEEE Wireless Communications and Networking Conference 2005*, Vol. 4, pp. 2057–2062.

Montemanni, R. and Mahdabi, P. [2011]. A linear programming-based evolutionary algorithm for the minimum power broadcast problem in wireless sensor networks, *Journal of Mathematical Modelling and Algorithms* **10**(2): 145–162.

Montemanni, R., Toklu, N. E., Toklu, Ş. Ç. and Toklu, Y. C. [2012]. Aggregate blending via robust linear programming, *Journal of Construction Engineering and Management* **138**(2): 188–196.

Nemirovski, A. [2009]. Lectures on robust convex optimization, *class notes, Georgia Institute of Technology, Fall* .

NEO Networking and Emerging Optimization [2012]. Vrp instances, `http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances`. Online; accessed 30-October-2012.

Neumann, D. L. [1964]. Mathematical methods for blending aggregates, *Journal of the Construction Division* **90**(CO2): 1–13.

Nguyen, T. V. L. and Montemanni, R. [to appear]. Mathematical programming models for home health care service optimization, *International Journal of Operational Research* .

Nickel, S., Schröder, M. and Steeg, J. [2009]. *Planning for home health care services*, Fraunhofer ITWM.

Porcheron, M., Gorge, A., Juan, O., Simovic, T. and Dereu, G. [2010]. Challenge ROADEF/EURO 2010: A large-scale energy management problem with varied constraints, *Technical report*, Électricité de France (EDF) R&D.

Raidl, G. R. and Puchinger, J. [2008]. *Hybrid Metaheuristics*, Vol. 114 of *Studies in Computational Intelligence*, Springer Berlin Heidelberg, chapter "Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization", pp. 31–62.

Rappaport, T. S. [1996]. *Wireless communications: principles and practice*, Prentice Hall.

Results of the ROADEF/EURO 2010 Challenge [2010]. `http://challenge.roadef.org/2010/result.en.htm`.

Ritter, J. B. and Shaffer, L. R. [1961]. Blending natural earth deposits for least cost, *Journal of the Construction Division* **87**(CO1): 39–61.

Sargent, C. [1960]. Economic combinations of aggregates for various types of concrete, *HRB Bulletin* **275**: 1–17.

Soyster, A. L. [1973]. Convex programming with set-inclusive constraints and applications to inexact linear programming, *Operations Research* **21**(5): 1154–1157.

Toklu, N. E., Gambardella, L. M. and Montemanni, R. [2014]. A multiple ant colony system for a vehicle routing problem with time windows and uncertain travel times, *Journal of Traffic and Logistics Engineering* **2**(1): 52–58.

Toklu, N. E., Gambardella, L. M. and Montemanni, R. [to appear]. Vehicle routing problem with uncertain costs via a multiple ant colony system, *In CMCGS 2014 - 3rd Annual International Conference on Computational Mathematics, Computational Geometry and Statistics*.

Toklu, N. E. and Montemanni, R. [2011a]. *Lecture Notes in Management Science*, Vol. 3, Tadbir OR, Vancouver, chapter "A Robust Approach for a Minimum Power Broadcasting Problem in Wireless Sensor Networks", pp. 223–232.

Toklu, N. E. and Montemanni, R. [2011b]. A three-stage robust approach for minimum power multicasting in wireless sensor networks, *Proceedings of MobiCONA 2011 - The Annual International Conference on Mobile Communications, Networking and Applications*, pp. M42–M47.

Toklu, N. E. and Montemanni, R. [2012a]. Minimum power multicasting on wireless networks: a shared incumbent environment approach, *Proceedings of MobiCONA 2012 - The 2nd Annual International Conference on Mobile Communications, Networking and Applications*.

Toklu, N. E. and Montemanni, R. [2012b]. Robust multicasting on stochastic wireless actuator networks: an algorithmic approach, *Journal of Applied Operational Research* **4**(3): 110–124.

Toklu, N. E., Montemanni, R., Di Caro, G. and Gambardella, L. M. [2012]. A shared incumbent environment for the minimum power broadcasting problem in wireless networks, *International Proceedings of Computer Science and Information Technology*, Vol. 27, pp. 158–162.

Toklu, N. E., Montemanni, R. and Gambardella, L. M. [2013a]. An ant colony system for the capacitated vehicle routing problem with uncertain travel costs, *IEEE Symposium on Swarm Intelligence*, pp. 32–39.

Toklu, N. E., Montemanni, R. and Gambardella, L. M. [2013b]. A robust multiple ant colony system for the capacitated vehicle routing problem, *IEEE International Conference on Systems, Man, and Cybernetics 2013*, pp. 1871–1876.

Toklu, N. E., Montemanni, R. and Gambardella, L. M. [to appear]. *Metaheuristics in Uncertain Environments*, Parallel and Distributed Computing Series, Wiley, chapter "Embedding Robust Optimization Techniques into Metaheuristic Algorithms".

Toklu, Y. C. [2002]. Aggregate blending problem–an arena of applications of optimization methods, *ICCCBE-IX The Ninth International Conference on Computing in Civil and Building Engineering* pp. 3–5.

Toklu, Y. C. [2005]. Aggregate blending using genetic algorithms, *Computer-Aided Civil and Infrastructure Engineering* **20**: 450–460.

Toth, P. and Vigo, D. (eds) [2001]. *The vehicle routing problem*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Tubacanon, M. T., Abuldhan, P. and Chen, S. S. Y. [1980]. A probabilistic programming model for blending aggregates, *Applied Mathematical Modelling* **4**: 257–260.

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

Venter, L. [2010]. *Metaheuristics for petrochemical blending problems*, Master's thesis, Stellenbosch University, South Africa.

Wieselthier, J. E., Nguyen, G. D. and Ephremides, A. [2000]. On the construction of energy-efficient broadcast and multicast trees in wireless networks, *INFOCOM 2000 - Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 2, pp. 585–594.

Wieselthier, J. E., Nguyen, G. D. and Ephremides, A. [2001]. Algorithms for energy-efficient multicasting in static ad hoc wireless networks, *Mobile Networks and Applications* **6**(3): 251–263.

Wieselthier, J. E., Nguyen, G. D. and Ephremides, A. [2002]. Energy-efficient broadcast and multicast trees in wireless networks, *Mobile Networks and Applications* **7**(6): 481–492.

Xhafa, F., Carretero, J., Dorronsoro, B. and Alba, E. [2009]. A tabu search algorithm for scheduling independent jobs in computational grids, *Computing and Informatics* **28**(2): 1001–1014.

Yang, X. [2011]. Metaheuristic optimization, *Scholarpedia* **6**(8): 11472. revision 91488.

Zitzler, E., Laumanns, M. and Thiele, L. [2001]. SPEA2: Improving the strength pareto evolutionary algorithm, *Technical report*, Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK).