Efficient combinatorial optimization algorithms for logistic 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
Vasileios Papapanagiotou

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

September 2018

Dissertation Committee

Prof. Evanthia Papadopoulou
University of Lugano

Prof. Fernando Pedone
University of Lugano

Prof. Agostino Bruzzone
University of Genoa

Prof. Bernard Ries
University of Fribourg

Dissertation accepted on 4th September 2018

Research Advisor	Co-Advisor
Prof. Luca Maria Gambardella	**Prof. Roberto Montemanni**

Academic Advisor

Prof. Jürgen Schmidhuber

PhD Program Director	PhD Program Director
Prof. Walter Binder	**Prof. Olaf Schenk**

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.

Vasileios Papapanagiotou
Lugano, 4th September 2018

To my beloved

Abstract

The field of logistics and combinatorial optimization features a wealth of NP-hard problems that are of great practical importance. For this reason it is important that we have efficient algorithms to provide optimal or near-optimal solutions. In this work, we study, compare and develop *Sampling-Based Metaheuristics* and *Exact Methods* for logistic problems that are important for their applications in vehicle routing and scheduling. More specifically, we study two *Stochastic Combinatorial Optimization Problems (SCOPs)* and finally a *Combinatorial Optimization Problem* using methods related to the field of *Metaheuristics*, *Monte Carlo Sampling*, *Experimental Algorithmics* and *Exact Algorithms*. For the *SCOPs* studied, we emphasize studying the impact of *approximating the objective function* to the quality of the final solution found.

We begin by examining Solution Methods for the Orienteering Problem with Stochastic Travel and Service Times (OPSTS). We introduce the state-of-the-art before our contributions and proceed to examining our suggested improvements. The core of our improvements stem from the approximation of the objective function using a combination of Monte Carlo sampling and Analytical methods. We present four new Evaluators (approximations) and discuss their advantages and disadvantages. We then demonstrate experimentally the advantages of the Evaluators over the previous state-of-the-art and explore their trade-offs. We continue by generating large reference datasets and embedding our Evaluators in two Metaheuristics that we use to find realistic near-optimal solutions to OPSTS. We demonstrate that our results are statistically significantly better than the previous state-of-the-art.

In the next chapter, we present the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands inspired by an environmental use case. We propose four different solution approaches based on different approximations of the objective function and use the Ant Colony Metaheuristic to find solutions for the problem. We discuss the trade-offs of each proposed solution and finally argue about its potentially important environmental application.

Finally, focus on exact methods for the Sequential Ordering Problem (SOP).

Firstly, we make an extensive experimental comparison of two exact algorithms existing in the literature from different domains (cargo and transportation and the other compilers). From the experimental comparison and application of the algorithms in new contexts we were able to close nine previously open instances in the literature and improve seventeen more. It also led to insights for the improvement of one of the methods (The Branch-and-Bound Approach - B&B). We proceed with the presentation of the improved version that led to the closing of eight more instances and speeding up the previous version of the B&B algorithm by 4%-98%.

Acknowledgements

At this point I would like to thank everyone who helped me through my doctoral studies culminating in this document. Shame on me for those people that I forgot to mention.

First of all, I would like to thank my Research Advisor Prof. Luca Maria Gambardella, Research Co-Advisor Prof. Roberto Montemanni and Academic Advisor Prof. Jürgen Schmidhuber all from the Dalle Molle Institute for Artificial Intelligence (IDSIA), Switzerland. Your example, efforts, advice, encouragement and above all mentorship were essential for me to arrive at this point.

I would also like to thank my dissertation committee (in alphabetical order): Prof. Agostino Bruzzone from University of Genoa, Prof. Evanthia Papadopoulou from Università Della Svizzera Italiana (USI), Prof. Fernando Pedone from Università Della Svizzera Italiana (USI) and Prof. Bernard Ries from University of Fribourg, all of them have already provided valuable feedback, ideas and advice related to my research.

Many thanks are given to the different PhD Program Directors at the Università Della Svizzera Italiana (USI), Switzerland during my study years (in alphabetical order): Prof. Walter Binder, Prof. Michael Bronstein, Prof. Antonio Carzaniga, Prof. Igor Pivkin and Prof. Stefan Wolf.

At this point, I would like to thank some people that contributed to my research in one way or another (again in alphabetical order): Cassio de Campos, Panagiotis Cheilaris, Panagiotis Farantatos, Salvatore Ingala, Akhmedov Murodzhon, Viet Thien Li Nguyen, Matteo Salani, Georgios Stamoulis, Nihat Engin Toklu, Vasileios Triglianos, Corrado Valeri, and Dennis Weyland.

A big thanks also to Dimitrios Floros for all his support throughout my studies.

Last but not least I would like to thank my parents Georgios and Theodora and my sister Nicoletta. They are the giants whose shoulders I stood on to become the person I am. Thank you for all your care and sacrifices to afford me the education I have: at home, at school, at University.

Contents

Figures

Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, we introduce the research areas relevant to the thesis, the problems studied and the prerequisite knowledge for the remaining part of the thesis.

1.1 Classification of the Research Area

In recent years, there has been an increasing interest for Stochastic Combinatorial Optimization Problems (SCOPs). In contrast to the traditional approach of Combinatorial Optimization Problems [Papadimitriou and Steiglitz, 1982; Lovasz, 1998], that consist of finding an optimal object from a finite set of objects given a deterministic objective and deterministic constraints, such as Vehicle Routing [Toth and Vigo, 2001], SCOPs encompass stochastic information about the problem. This results to more realistic models as in the reality many events or quantities can be approximated probabilistically more accurately (e.g. travel times) due to unpredictable factors such as traffic or weather conditions.

One of the classes of problems of SCOPs that is central to the thesis is the *Stochastic Vehicle Routing Problems*. Vehicle Routing Problems deal with the transport of goods between depots and customers by a fleet of vehicles. The overall goal is to fulfill the transportation requirements in the most efficient way regarding objectives as e.g. costs or travel times, while respecting the operational constraints, such as capacities of vehicles, time window constraints, working time constraints and maximum route lengths. While in the classic scenario the model contains only precise information, Stochastic Vehicle Routing Problems are modeled using stochastic data. A common way to model these stochastic information is to use probability distributions, which are either known or estimated from historical data. Widely used stochastic elements in the context of Vehicle Routing Problems are the presence of customers, demands of customers and travel

times. Typically the objective function is a stochastic variable, e.g. the expected travel time or the expected costs. It is also possible to formulate constraints using stochastic data, which leads to so called chance-constrained problems. For example, one constrained could assure that each customer is visited within a predefined time window with a certain probability. Examples of Stochastic Vehicle Routing Problems are the *Orienteering Problem with Stochastic Travel and Service Times*[Campbell et al., 2011] and the *Probabilistic Travelling Salesman Problem* [Bertsimas and Howell, 1993]. While Stochastic Vehicle Routing Problems can be used to obtain more realistic models for real world problems, they are usually harder to solve than their non-stochastic counterparts. One reason for this inherent hardness is that Stochastic Vehicle Routing Problems are structurally different from classic Vehicle Routing Problems and therefore well established methods cannot be used in this context. The other reason is that the objective function for many Stochastic Vehicle Routing Problems is computationally very expensive. In some cases no polynomial time algorithms for the evaluation of a solution are known and sometimes there is even no closed form expression for the objective function available. Additionally, most of the Stochastic Vehicle Routing Problems are NP-hard and exact mathematical approaches can only be applied to very small instances. Therefore heuristics and metaheuristics are of great interest for obtaining good or even optimal solutions for those problems. In the cases where the objective function is computationally very expensive or not available as a closed form expression, the combination of metaheuristics with Monte Carlo Sampling for the evaluation of solutions, currently belongs to the most promising approaches. Subsequently we discuss this concept more in detail.

One of the widely used concepts used in the thesis is the concept of metaheuristics. *Metaheuristics* are general purpose methods for solving hard optimization problems. They operate on the solution space using a single solution or a population of solutions. The quality of a solution is measured by an objective function. Iteratively, new solutions are generated and replace the current solution or solutions in the current population according to a specific procedure and considering the qualities of the solutions involved in this process. Although metaheuristics obtain good solutions efficiently for a wide variety of problems, usually no a-priori guarantees for the quality of the solutions are given. Many of those methods are exploiting analogies, e.g. from the nature or physical systems, for the optimization process. Examples of the most successful metaheuristics include Local Search, Tabu Search, Simulated Annealing, Ant Colony Optimization, Evolutionary Algorithms and Particle Swarm Optimization. For an extended review and a detailed comparison of various metaheuristics, we refer to [Bianchi et al., 2009; Blum and Roli, 2003; Gendreau and Potvin, 2005; Talbi, 2009; Luke,

2013].

The main metaheuristics used in this thesis are the *Random Search Metaheuristic*, the *Variable Neighborhood Search Metaheuristic* and the *Ant Colony Optimization Metaheuristic*. Useful resources for the *Random Search Metaheuristic* are: [Spall, 2005], [Solis and Wets, 1981], [Brownlee, 2011], for the *Variable Neighborhood Search Metaheuristic*: [Mladenović and Hansen, 1997], [Sevkli and Sevilgen, 2006] and [Campbell et al., 2011] and for the *Ant Colony Optimization Metaheuristic*: [Dorigo, 1992], [Dorigo et al., 1991] and [Gambardella and Dorigo, 1999].

Now let us examine how those metaheuristics could be used in combination with Monte Carlo Sampling to solve Stochastic Vehicle Routing Problems. In the standard case, we have given a problem with stochastic input data, non-stochastic constraints and an objective function which is the expectation of a certain random variable. We then distinguish three different scenarios regarding the complexity of the objective function.

1. A closed-form expression for the objective function is available and can be computed efficiently.

2. A closed-form expression for the objective function is available, but the evaluation of a solution is computationally too expensive.

3. There is no closed-form expression for the objective function available.

It will be shown how Monte Carlo sampling is useful in all of the above cases. But first let us define Monte Carlo Sampling. Let X be the solution space and assume that the stochastic data are given by the probability space (Ω, Σ, P). Let x be a solution and let ω be a random variable according to the given probability space. $F : X \times \Omega \rightarrow R$ represents the solution costs for a given solution and a given realization of the random variable. Now we can define the objective function $f : X \rightarrow \mathbb{R}$ as $f(x) = \mathbb{E}(F(x, \omega))$. Now, if we were to take an n-sample of the sample space (n realizations of the random variable ω: $\omega_1, \omega_2, \ldots, \omega_n$) and we computed over the sample, then we would have the Monte Carlo approximation $\widetilde{f}_n : X \rightarrow \mathbb{R}$ of $\mathbb{E}(f(x, \omega))$:

$$\widetilde{f}_n(x) = \frac{1}{n} \sum_{i=1}^{n} F(x, \omega_i)$$

We can see that for arbitrarily large values of n:

$$\mathbb{E}(\widetilde{f}_n(x)) = \mathbb{E}(\frac{1}{n} \sum_{i=1}^{n} F(x, \omega_i)) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}(F(x, \omega_i)) = \mathbb{E}(F(x, \omega)) = f(x)$$

Since the expected value of the Monte Carlo approximation of f is f itself, then this approximation is an unbiased estimator of f.

For problems belonging to problem class (1) we have the following situation. A closed-form expression for the objective function is available and can be computed efficiently. This means that we can apply metaheuritics in a straightforward way using the exact objective function. Although this leads to efficient heuristics, metaheuristics using the Monte Carlo Sampling approximation of the objective function can lead to significant improvements. In this thesis, the Orienteering Problem with Stochastic Travel and Service Times, presented in Chapter 2, belongs in this class. There is a closed-form approximation of the objective function that can be computed in polynomial time. However, as it will be shown in the relevant chapter, Monte Carlo sampling methods alone or in combination with deterministic and analytical methods when embedded in metaheuristics can obtain same quality solutions with significantly less computational time or significantly better solutions in the same time. A similar situation is true for the *Probabilistic Traveling Salesman Problem (PTSP)* [Weyland, Bianchi and Gambardella, 2009], [Weyland, Montemanni and Gambardella, 2009].

For problems belonging to problem class (2) a closed-form expression for the exact objective function is available but cannot be computed efficiently. Applying metaheuristics for those problems using the exact objective function is not feasible, since the overall computational time is too large. In contrast to the exact objective function, the Monte Carlo Sampling approximation of the objective function can be computed efficiently in this case. Metaheuristics using this approximation can be used to solve those problems efficiently. A recent example in this problem class is the *Probabilistic Traveling Salesman Problem with Deadlines (PTSPD)*. There is no efficient method for the computation of the objective function available and heuristics using the best known method for the computation of the objective function lead to non feasible computational times. On the other hand, metaheuristics based on the Monte Carlo Sampling approximation require only a fraction of the computational time and lead to high quality solutions in a reasonable amount of runtime [Weyland et al., 2013].

In the third problem class there is no closed-form expression for the exact objective function available. That means we cannot use heuristics to solve those kind of problems in a straightforward way. Fortunately, it is possible to approximate the objective function using Monte Carlo Sampling. In this way metaheuristics based on the Monte Carlo Sampling approximation can be used to tackle this problem and in many cases this is the only existing alternative for tackling problems belonging to this class. The *2-stage Capacitated Vehicle Routing Problem with Stochastic Demands* that we explore in Chapter 3 belongs in this class.

Another example belonging to the third class that deals with complex constraints on delivery timeframe, accessibilities and stop frequency by applying a methodology combining Artificial Intelligence and Discrete event Stochastic Simulation can be found on [Bruzzone and Longo, 2014]. The paper provides an application methodology called MARLIN that consists of defining constraints, defining scenarios and factors and finally analyzing and comparing the results from the simulation of the scenarios. The MARLIN methodology has been used in real-world supply chains e.g. for delivering fresh fish to super markets in Northern Italy.

For the analysis of the performance of our approaches we borrow methods from Statistics such as hypothesis testing [Tanis, 2008] and *Experimental Algorithmics*

Finally, we explore and improve *Exact Algorithms*[Woeginger, 2003] using techniques such as Branch-and-Bound [Jamal et al., 2017] and Mixed-Integer Linear Programming for the *Sequential Ordering Problem* [Montemanni et al., 2013]

1.2 Main Optimization Problems in the Thesis

1.2.1 Introduction

In this section we introduce formally the logistic problems that are discussed in the thesis, namely the Orienteering Problem with Stochastic Travel and Service Times, the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands and the Sequential Ordering Problem. The same notation may be used with different meaning in each problem according to the definitions given in the corresponding sections.

1.2.2 The Orienteering Problem with Stochastic Travel and Service Times

The OPSTS was first introduced in [Campbell et al., 2011]. In this problem there is a starting point that we call the depot. A vehicle goes out of that depot and has to serve some customers. The vehicle will end its route at a destination node and does not have to return to the depot. There is a global deadline and in most cases it is such that it is not possible to visit all the customers before the deadline. For this reason, a subset of customers has to be selected to be served. After selecting this subset, the vehicle tries to serve the customers in the subset. For each service

before the deadline it earns a reward, otherwise a penalty is incurred. What makes the OPSTS different than the traditional Orienteering Problem is the fact that the travel and service times are stochastic and we know beforehand their probability distribution. For example in figure 1.1, the company (visualized by a truck) will visit the customers $1, 4, 3, 7$ and leave out customers $5, 2, 6$ because of the global deadline, 10 in this case. The deterministic travel times can be seen on each arc and in parentheses we can see the actual travel times for this instance. Because of the stochasticity of the travel times, the total travel time exceeds the deadline and the company will have to pay a penalty. As it will be obvious from subsequent sections the deadline is an important component of the structure of the problem.

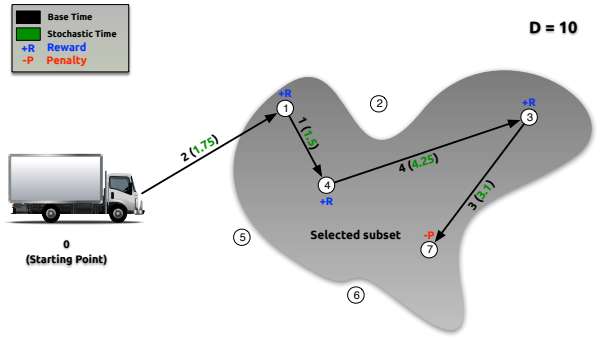


Figure 1.1. In this instance of OPSTS the truck will visit nodes 1,4,3,7. The deadline is 10. The deterministic times are in black and the actual travel times are in the parentheses

1.2.2.1 Literature Review

The Orienteering problem originates from the sports game of orienteering where players begin from a certain point and they have a deadline before which they collect different rewards from different checkpoints and return to the starting point [Tsiligirides, 1984],[Chao et al., 1996]. Therefore, the Orienteering Problem combines vertices selection and finding a path that connects all the vertices exactly once apart from the starting vertex which is visited twice (this is also called a Hamiltonian tour). In order to get as much reward as possible the correct selection of vertices and the shortest Hamiltonian tour between them have to be found. Please note that in the variant examined in this thesis (OPSTS) apart from the stochasticity in travel and service times we do not consider the arc from the last vertex to the depot. For more information, please see Section 1.2.2.2.

The Orienteering problem is also called the bank robber problem [Arkin et al., 1998], the selective travelling salesman problem [Gendreau et al., 1998; Laporte and Martello, 1990; Thomadsen and Stidsen, 2003] and the maximum collection problem [Butt and Cavalier, 1994; Kataoka and Morito, 1988]. Another related problem is the Travelling Salesman Problem (TSP) with profits [Feillet et al., 2005].

In the deterministic version of the orienteering problem we have a set of N vertices i and fixed starting and ending vertices (1 and N). There is a deadline T_{max} before which a path between the starting and ending vertices has to be determined. The vertices can be visited at most once and the scores are additive. The problem can be defined on a graph $G = (V,A)$ with $V = \{v_1, \ldots, v_N\}$ being the vertex set and A the arc set. Each vertex has non-negative score (reward) S_i, and each arc $a_{ij} \in A$ has a travel time t_{ij}. A Hamiltonian path $G'(\subset G)$ is a set containing the vertices selected to be visited (starts from v_1 and ends at v_N and has length less than or equal to D). The objective is to find the Hamiltonian path over $G' \subset G$ which is the set that respects the aforementioned constraints so that the reward collected is maximized. In the case that we want a tour instead of a path, we can add a dummy arc between the start and end vertex. If we let $x_{ij} \in \{0,1\} \forall i \in 1, \ldots, N$ indicate if a visit to i is followed by a visit to j (0 means no, 1 means yes) and u_i the position of the vertex in the path, then the orienteering problem can be formulated as a mathematical program as follows:

$$\text{maximize} \left[\sum_{i=2}^{N-1} \sum_{j=2}^{N} S_i x_{ij} \right] \tag{1.1}$$

subject to

$$\sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1, \tag{1.2}$$

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1, \forall k = 2, \ldots, N-1 \tag{1.3}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} (t_{ij} x_{ij}) \leq T_{max} \tag{1.4}$$

$$2 \leq u_i \leq N; \forall i = 2, \ldots, N, \tag{1.5}$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}); \forall i, j = 2, \ldots, N, \tag{1.6}$$

$$x_{ij} \in \{0,1\}; \forall i, j = 1, \ldots, N \tag{1.7}$$

The objective stated in Equation (1.1) is to maximize the total score from

the visited nodes while guaranteeing that the path starts at vertex 1 and ends at vertex N (1.2). Constraints (1.3) ensure the connectivity of the path while constraint (1.4) ensures that the time budget is not violated. Inequalities (1.5), (1.6) eliminate subtours according the the Miller-Tucker-Zemlin (MTZ) Travelling Salesman formulation found in [Miller et al., 1960]. It must be noted that the greatest differences of this formulation from a TSP one is in (1.3) where the sums must be ≤ 1 implying that not all nodes are selected and similarly the MTZ constraint in (1.6) is modified to account for nodes not selected in the Orienteering Problem.

A very good overview of the deterministic orienteering problem and its variants as well as the mathematical formulation presented can be found in [Vansteenwegen et al., 2011].

In the thesis, we examine a variant of the orienteering problem where the travel and service times are stochastic, for a detailed definition Section 1.2.2.2. To the best of our knowledge, apart from [Campbell et al., 2011] and our work [Papapanagiotou et al., 2013],[Papapanagiotou et al., 2014], [Papapanagiotou et al., 2016c], [Papapanagiotou et al., 2015b], [Papapanagiotou et al., 2015a], [Papapanagiotou et al., 2016a] and [Papapanagiotou et al., 2016b] there is limited other literature on the topic. Stochastic variants of the travelling salesman problem are closely related to the Orienteering Problem with Stochastic Travel and Service Times. One of the most closely related problems is the probabilistic traveling salesman problem with stochastic travel and service times (TCTSP). [Teng et al., 2004] introduce and solve TCTSP with discrete travel and service time distributions. They use an L-shaped algorithm and they manage to solve problems with up to 35 customers.

In [Gambardella et al., 2011] an ant colony approach coupled with local searches is presented for solving the Probabilistic Travelling Salesman Problem (PTSP) and in [Montemanni and Gambardella, 2009] an Ant Colony System is developed for the team orienteering problem with time windows; both papers have similarities with the OPSTS.

Another related problem to OPSTS is the Selective Travelling Salesperson Problem (SSTP) first presented in [Tang and Miller-Hooks, 2005]. The difference with OPSTS is that SSTP has chance constraints for the deadlines instead of imposing penalties for unfulfilled ones. The focus of the experiments in that paper is the performance of the algorithms, and only tight deadlines are taken into account. Another stochastic version of the orienteering problem that has been considered is the orienteering problem with stochastic profits for each customer. In this problem the objective is to maximize the probability to reach a certain level of profit before the deadline. An exact algorithm is proposed in

[İlhan et al., 2008].

Additionally, there are papers dealing with various vehicle routing problems with random travel times and time constraints. These papers have significant differences in their objectives with OPSTS and their methods cannot be directly applied to OPSTS. An example is the Capacitated Vehicle Routing Problem with stochastic travel times where the objective is to maximize the probability that all vehicle tours are completed before a strict deadline. In [Kenyon and Morton, 2003] the problem is tackled using a Monte Carlo procedure to create solutions and select the best. [Jula et al., 2006] consider a vehicle routing problem with stochastic travel times and time windows. They show how to compute the first and second moments of the arrival time distributions and propose a dynamic programming approach that solves up to 80 customers with tight time windows. [Russell and Urban, 2008] consider a vehicle routing problem with stochastic travel times with soft time windows. The travel times follow the Erlang distribution and the Taguchi loss function is used to compute the penalties for time windows violation.

1.2.2.2 Problem Definition

In the rest of this section we give the formal definition of the Orienteering Problem with Stochastic Travel and Service Times. Let $N = \{1, \ldots, n\}$ be a set of n customers and 1 being the depot. We define a subset $M \subseteq N$ as the set of customers selected to be served. The customers in M have to be served before a global deadline D, otherwise the server gets a penalty from each customer in M served after the deadline D. The graph of the customers is assumed to be fully connected and therefore there is an arc (i, j) for all $i, j \in M$. The server gets a reward r_i for each customer $i \in M$ served before the deadline, and for each customer $j \in M$ served after the deadline a penalty e_i is incurred. To represent the travel time we define a $X_{i,j}$ as a non-negative random variable from node i to node j and for the service time we define a non-negative random variable S_i for each customer i. We make the assumption that the probability distributions of both $X_{i,j}$ and X_i are known $\forall i, j$ and they are the same. In [Campbell et al., 2011], the probability distribution of the random variables follow a Γ distribution. Let the random variable A_i be the arrival time at customer i and $\bar{A}_i$ a realization of A_i. We now represent the reward earned at customer i when arriving to i at time $\bar{A}_i$ as $R(\bar{A}_i)$. According to the previous definitions $R(\bar{A}_i) = r_i$ for $\bar{A}_i \leq D$, otherwise $R(\bar{A}_i) = -e_i$ (for $\bar{A}_i$).

Let τ be a tour of the customers, defined as a sequence of customers of M. The objective function of the problem is defined as the expected reward of the

tour τ:

$$\max \upsilon(\tau) = \sum_{i \in \tau} [P(A_i \leq D) r_i - (1 - P(A_i \leq D)) e_i] \tag{1.8}$$

1.2.2.3 Computing the objective function for the Γ distribution

Equation (1.8) can be computed easily if we can compute the cumulative distribution function (CDF) of all the A_i. In statistics, we call the cumulative distribution function $F_X(x)$ of a real-valued random variable X the probability that X will take a value less than or equal to some x ($P(X \leq x)$). Therefore $P(A_i \leq D)$ in terms of a CDF can be computed as $F_{A_i}(D)$. Thus, the important thing is to know the CDF of A_i. A_i is the arrival time in node i and therefore it is the sum of all the travel times till node i. Then $A_i = \sum_{k=0}^{i} X_i$. It is known that if Y_i follows a $\Gamma(k_i, \theta)$ distribution for $i = 1, 2, \cdots, N$ and all Y_i are independent then $\sum_{i=1}^{N}(Y_i) \sim \Gamma(\sum_{i=1}^{N}(k_i, \theta)$ and since the CDF of the Γ function is known, we can compute the CDF of A_i, see also [Johnson et al., 2002]. If we assume that the CDF of Γ with scale parameter k_i is F_{k_i} then the objective function (1.8) can be computed as $\sum_{i \in \tau} [F_{k_i}(D) r_i - (1 - F_{k_i}(D)) e_i]$.

1.2.2.4 A mathematical formulation for OPSTS

In this section we show a mathematical formulation for OPSTS. The formulation is based on the combination of the deterministic and stochastic versions of the problem, see [Campbell et al., 2011] and [Vansteenwegen et al., 2011]. The formulation for OPSTS differs from the one presented in (1.1) - (1.7) in the following points. Firstly, the objective function is the Equation (1.8). Additionally, the violation of the deadline in Inequality (1.4) does not exist since in the OPSTS formulation the constraint is soft and is penalized for its violation (which is obvious in the objective function).

1.2.2.5 Exact Solution approach with Dynamic Programming

For certain time distributions an exact solution of the OPSTS can be obtained using dynamic programming. The approach is motivated by [Kao, 1978] and it is fully developed in [Campbell et al., 2011]. This solution method works only when the travel and service time distributions differ in a single parameter and the sum of this parameter can characterize the convolution of the arrival time distributions. Such functions, for example, are the Γ (as discussed in Section

1.2.2.3) and the Poisson (with common scale parameter), the normal (if the variance is a constant multiple of the mean) and the binomial and negative binomial (with common parameter p), for more information see [Johnson et al., 2002]. Let us name the parameter k_{ij} as the characterizing parameter (e.g. scale) for each arc (i, j) and k_i as the service time parameter. For the distributions explicitly mentioned earlier the convolution of travel and service time distributions is characterized by the sum of the characterizing parameters.

Let us assume that the travel and service time distributions are independent and identically distributed and differ in their parameters as noted above. We denote with $i \in N$ the last visited node and with K the nodes visited from the depot up to i. Let m be the value of the characterizing parameter for the arrival time distribution at i. The tuple (i, K, m) can fully characterize the state of the algorithm. The actions from any state are to travel to a not-yet visited node $\in N \backslash K$ or to the end node. Traveling from node i with parameter k to node j results in a new state $(j, K \cup j, k + k_{ij} + k_j)$. As noted previously the expected reward for the j state is $R(i, K, m, j) = \sum_{i \in \tau}[F_{m'}(D)r_i - (1 - F_{m'}(D))e_i]$, $F_{m'}$ is the cumulative distribution function with $m' = k + k_{ij} + k_i$. Ending the tour yields no reward.

The possible (i, K, m) states do not repeat and are acyclic and thus we can obtain the following functional equation:

$$f(j, K', m') = \max_{(i,K,m):K=K'\backslash j, m'=k+k_{ij}+k_i} \{f(i, K, m) + R(i, K, m, j)\} \tag{1.9}$$

In [Denardo and Fox, 1979] it is shown how (1.9) can be solved recursively. For further acceleration, pruning techniques can be applied using the reaching algorithm described in [Denardo, 2012]. As initialization of the algorithm we set the value of each state $f(i, K, m)$ is set to $-\infty$ and the initial (i, K, m) state is set to $(0, \emptyset, 0)$. When node $j \notin K$ is visited from node i, the state changes to $f(j, K \cup j, k + k_{ij} + k_i) = \max\{f(j, K \cup j, k + k_{ij} + k_i), R(i, K, m, j)\}$.

Here we can make some observations for the implementation of the dynamic programming solution. Firstly, for any 2 states (i, K, m) and (i, K, m') such that $(m \leq m')$ (the total distance of m' is longer but the tour τ is the same), we have that $P(A_j \leq D) \geq P(A'_j \leq D)\ \forall j \in \tau$. Therefore $E[R(A_j)] \geq E[R(A'_j)]$ $\forall j \in \tau$ see [Puterman, 2014]. Thus, in this case $f(i, K, m) \geq f(i, K, m')$ and so (i, K, m) dominates (i, K, m') and we prune (i, K, m') from the states needed to be considered. For further enhancements in the implementation see [Feillet et al., 2004].

1.2.3 The 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands

In this section we define a Capacitated Vehicle Routing Problem with Probabilistic Demand Increases inspired by realistic cases. This problem was motivated by and also helped in the study and mitigation of the "Green Bullwhip Effect (GBWE)" which is a kind of "Bullwhip Effect". The description is based on our published work [Toklu et al., 2013], [Toklu et al., 2014] and [Klumpp et al., 2014].

This problem is motivated by the case of a company that needs to serve multiple different customers with uncertain demands and has to decide the routes and number of vehicles needed to serve the customers. The objective is to minimize the total distance of the routes chosen.

The solutions proposed because of the differences in the distances traveled and the number of vehicles used, they have different environmental and financial impacts.

Furthermore, this problem considers only probabilistic increases in the demands of the customers. In reality, increases in the demand are many times caused by what is known as the "bullwhip effect".

The "bullwhip effect" has been documented many times in the literature, some representative references are [Forrester, 1961], [Lee et al., 1997]. In short, the "bullwhip effect" also known as the "Forrester effect" refers to the fact that even small increases in demand can cause larger and larger swings in inventory as one examines further back the supply chain.

The problem of serving a number of transportation requests with a fleet of vehicles is known as the Vehicle Routing Problem (VRP). The objective is to find a way to satisfy all the transportation requests by using the given fleet of vehicles at minimum cost. The proposed solution should define which vehicle handles which request and in which sequence so that all the transportation requests can be feasibly satisfied.

One of the most famous variants of VRP is the Capacitated Vehicle Routing Problem (CVRP). In the classical version of the problem of CVRP, we have one depot and a fleet with identical vehicles and capacity constraints. The objective is to find routes of vehicles with minimal travel costs. We assume that the capacity of one (single) vehicle is always greater than the demand of a single customer.

A related problem to the one presented in this paper is the Capacitated Vehicle Routing Problem with Stochastic Demands (VRPSD). In VRPSD, instead of a constant demand, a probability distribution is specified for the demand of each customer and the assumption is that the demands are independent. Before the beginning of the tour, a feasible solution is found called *a priori* tour. In the clas-

sical version of VRPSD each vehicle of the fleet while executing the a priori tour, has to choose whether to proceed to the next customer or return to the depot for restocking.

In our work, a variant of VRPSD is considered. In this variant, demands can be increased with some probability. Furthermore, the vehicles are filled to their capacity and always proceed to serve the customers without deviating from their a priori tour. If a vehicle gets empty without having served the demands of the customers in their entirety, then a new vehicle begins from the depot to serve the remaining demands of the customers, by following a newly computed tour. This procedure can be repeated until all the demands of the customers are met.

1.2.3.1 Literature Review

The Vehicle Routing Problem with Stochastic Demands has been researched before in a number of papers but to the extend of our knowledge with different assumptions from our variant. Whenever not mentioned in the related work below, it is assumed that the VRPSD solved is the classical one.

In [Bertsimas, 1992], the difference in assumptions is that the vehicle returns periodically to the depot to empty its current load. Uncertainty is handled by building an a priori sequence among all customers of minimal expected length. Also simple heuristics are proposed and theoretical investigations are performed. In [Teodorović and Pavković, 1992] simulated annealing is used to solve VRPSD. In [Gendreau et al., 1995] an exact algorithm for the VRPSD is proposed by formulating it as a two stage problem and solving the second stochastic integer program using an integer L-shaped method. In [Bianchi et al., 2006] different hybrid metaheuristics are analyzed in terms of performance and compared to state of the art algorithms. This paper also uses the notion of 'preventive restocking'. Preventive restocking means that the vehicle chooses to go to the depot for restocking even though it is not empty and can satisfy the next customer. In [Tripathi et al., 2009] an Ant Colony Optimization is proposed called "neighborhood-search embedded Adaptive Ant Algorithm (ns-AAA)" in order to solve VRPSD. The VRPSD solved in this paper has the assumptions of the classical VRPSD and also uses preventive restocking. In [Tan et al., 2007] a multiobjective version of VRPSD is solved by means of evolutionary methods. The algorithm finds tradeoff solutions of complete routing schedules with minimum travel distance, driver remuneration and number of vehicles, with constraints such as time windows and vehicle capacity. In [Erera et al., 2010] duration constraints are imposed on the expected delivery costs and this affects the structure of the set of a priori tours.

Part of the work was intended to be used in Green Logistics. In [Klumpp,

2011], the existing knowledge about the bullwhip effect and green logistics is described and a volatility simulation analysis of specific and relevant green logistics instruments is performed to the whole supply chain. For a more comprehensive coverage of logistics trends one can consult [Klumpp et al., 2013]. Concerning the "bullwhip effect", its impact, quantification and simulation, one car refer to [Forrester, 1961], [Metters, 1997] or more recent sources like [Özelkan and Lim, 2008] and [Jaksic and Rusjan, 2008].

1.2.3.2 Problem definition

Let $G = (L, A)$ be a graph, L a set of locations and A a set of arcs connecting the L vertices. It is assumed that the depot is always number 0 in the set of locations and the graph is complete. Also V is the set of vehicles available and c_{ij} is the cost of traveling from location i to location j.

To deal with probabilistic increases in the demands, we generate various scenarios and in each of them the demands are perturbed in many different ways. The probability distributions used to perturb the demands are described in Chapter 3. S is the set of scenarios considered, that are sampled using Monte Carlo sampling, and d_i^s the demand in location i in scenario $s \in S$.

The objective of this problem is to find a feasible solution x that minimizes the total travel distance of the routes performed by the fleet and therefore the total carbon dioxide emission. We let x^v be the route decided for vehicle $v \in V$ in x, $|x^v|$ the length of the route x^v, and x_k^v the k-th visited place of the vehicle v in x. $\underline{d}_{x_k^v}$ is the actual demand revealed in location k visited by the vehicle v using the route x^v and Q is the capacity of the vehicle.

In case a vehicle runs out of capacity, a new one begins from the depot to satisfy the remaining demands of the clients.

For the computation of the final cost we first evaluate the "base cost" which is the cost ignoring uncertainty. Then we simulate the route for all scenarios generated, which take into account uncertainty and return the average of the cost computed. We call this procedure the "fix function" and it is computed in the objective function by the term $\sum_{s \in S}(\frac{F_s(x)}{|S|})$. $F_s(x)$ represents the additional cost in scenario s, due to stochasticity.

To sum up, our problem can be informally expressed as:

$$\text{minimize} \left[\sum_{v \in V} \sum_{k=1}^{|x^v|-1} c_{x_k^v, x_{k+1}^v} + \sum_{s \in S} \left(\frac{F_z(x)}{|S|} \right) \right] \tag{1.10}$$

$$\text{subject to } x_1^v = x_{|x^v|}^v = 0 \qquad \forall v \in V \tag{1.11}$$

$$x_k^v \neq x_{x'}^{v'} \qquad \forall v, v' \in V$$

$$\forall k, k' \in \{2, ..., |x^v| - 1\}$$

$$k \neq k' \text{ if } v = v' \tag{1.12}$$

$$x_k^v \in (L \backslash \{0\}) \qquad \forall v \in V, \forall k \in \{2, ..., |x^v| - 1\} \tag{1.13}$$

$$\sum_{k \in \{2, ..., |x^v|-1\}} \underline{d}_{x_k^v} \leq Q \qquad \forall V \in V \tag{1.14}$$

$$x^v \text{ is a route} \tag{1.15}$$

where (1.11) says that the starting and ending location of a tour is the depot; (1.12) says that the same customer must not be visited twice; (1.13) says that the non-depot locations visited by a vehicle must be valid customers; (1.14) says that the total demands on a route of a vehicle must not exceed the vehicle capacity.

The fix function $F_z(x)$, as a part of a very accurate objective function, can be explained as simulating the solution x over scenario s, getting a list of unsatisfied customers, finding tours for the extra vehicle(s) by using an exact method to revisit the unsatisfied customers, and finally returning the total cost of these tours. However, in our case for reasons that will be explained in Section 3.2.4, to solve the $SubVRP$ we use the Nearest Neighbor Heuristic (NNH).

The algorithmic representation can be seen in Algorithm 1. The nested loop finds any customers that are still unserved with complexity $O(V \cdot |x^v|)$. Then the unserved customers are served by using either Nearest Neighbourhood Search (NNH) with complexity $O(n^2)$ where n the number of unsatisfied customers, or using an exact method with complexity that depends on the solver.

1.2.4 The Sequential Ordering Problem

The Sequential Ordering Problem (SOP) is a combinatorial optimization problem that may be briefly described as follows. Given a weighted directed graph, a set of precedence constraints between vertex pairs and a starting vertex, the objective is to find a minimum-cost Hamiltonian path. A path is feasible if it fulfills the precedence constraints. For example, in Figure 1.2, we see an instance of SOP. It is a fully connected directed graph and the coloured edges mark a feasible solution. They form a Hamiltonian path that respects the precedents constraints.

Algorithm 1 The algorithmic representation of the function $F_z(x)$

function $F_z(x)$ **is:**
$unsatisfied\_customers \leftarrow []$
for all $v \in V$ **do**
  $serving\_capacity \leftarrow vehicle\_capacity$
  for all $k \in \{2, ..., |x^v| - 1\}$ **do**
    if $serving\_capacity \geq d_i^s$ **then**
      $serving\_capacity \leftarrow serving\_capacity - d_i^s$
    else
      add k to $unsatisfied\_customers$
      $missing_k \leftarrow d_i^s - serving\_capacity$
      $serving\_capacity \leftarrow 0$
    end if
  end for
end for
$SubGraph \leftarrow \{0\} \cup unsatisfied\_customers$
$SubVRP \leftarrow$ VRP problem on $SubGraph$ where
    the demands are given by $missing_k \ \forall k \in SubGraph$
$y \leftarrow$ solve $SubVRP$ using an exact method (in our case NNH)
return travel cost of y

Here the precedence constraints are marked with dotted arrows and dictate that node 3 should be visited before node 2 (but it does not matter if it is exactly before) and node 2 should be visited before node 4.

A SOP without precedence constraints is equivalent to an Asymmetric Travelling Salesman Problem (ATSP), and when the latter has symmetric costs, it is referred to as the Travelling Salesman Problem (TSP). Therefore, the SOP is a generalization of the TSP, and since the TSP is a well-known NP-hard problem, the SOP must be NP-hard. In Section 1.2.4.1, literature review and some existing applications will be discussed.

1.2.4.1 Literature Review

The problem has been initially formulated by [Escudero, 1988] as the underlying model for a production planning system. Escudero also presents an inexact algorithm that exploits the properties of the ATSP poly-tope. A similar approach is taken by [Ascheuer et al., 1993], who show how to generate valid cuts using a polynomial-time separation algorithm. Another cutting-plane approach based

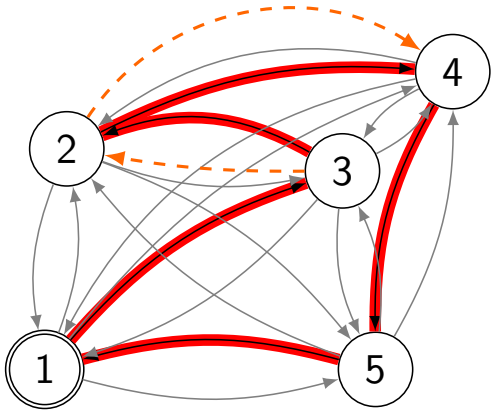


Figure 1.2. The colored edges represent a feasible solution for this instance of SOP, it is a Hamiltonian path in the fully connected graph that respects the precedence constraints marked with the dotted arrows

on a Lagrangian relaxation was presented by [Escudero et al., 1994]. [Hernàdvölgyi, 2003], [Hernàdvölgyi, 2004] generated good lower bounds using a technique that reduces the instance size. [Balas et al., 1995] presents a deeper analysis of the impact of precedence constraints on the ATSP convex hull.

Metaheuristic methods have also been proposed for solving the SOP. [Chen and Smith, 1996] and [Moon et al., 2002] propose genetic algorithms. [Pulleyblank and Timlin, 1991] use a Voronoi quantized crossover that adopts a complete graph representation. [Gambardella and Dorigo, 2000] describe the HAS-SOP algorithm, which couples an ant-colony system with a 3-opt local search. The effectiveness of this algorithm appears to be dependent on the density of the precedence constraints. [Montemanni et al., 2007], [Montemanni et al., 2008], [Montemanni et al., 2009] show how to exploit this property by artificially manipulating the density of precedence constraints. Experiments comparing the previous two algorithms on a set of real problems arising in quay crane assignment have been reported by [Montemanni et al., 2008b]. [Anghinolfi et al., 2009], [Anghinolfi et al., n.d.] propose a Particle Swarm Optimization (PSO) approach to solving the SOP. A meta-heuristic approach combining a Mixed Integer Linear Programming (MILP) solver and an Ant System has been described by [Mojana et al., to appear].

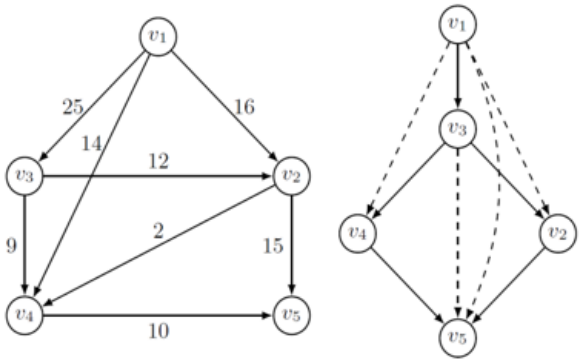


Figure 1.3. An example a SOP instance, cost graph (left), precedence graph (right)

1.2.4.2 Problem Definition

In this section we provide the formal definition of the Sequential Ordering Problem (SOP). Let $D = (V,A)$ be a complete weighted directed graph where V is the set of vertices and $A = (i,j)|i,j \in V$ the set of arcs. All the arcs $(i,j) \in A$ have an associated cost $c_{ij} \in N_0^+$. We denote as $v_1 \in V$, the first and last vertex of the solution path. Moreover, a precedence digraph $P = (V,R)$ is also defined with the same vertex set V as D. A precedence constraint is defined as an arc $(i,j) \in R$ where vertex i has to precede vertex j in any feasible path and P must be acyclic. A feasible solution to SOP is a Hamiltonian path (a path that includes each vertex exactly once) and respects the precedence constraints. The cost C of path S is defined as $C(S) = \sum_{(i,j)\in S} c_{ij}$. Note that the precedence constraints satisfy the transitivity property, that is if $(i,j) \in R$ and $(j,k) \in R$ then $(i,k) \in R$. The objective is to find a feasible solution with minimal cost.

Figure 1.3 shows an example of a SOP instance. The graph on the left is the cost graph, while the graph on the right is the precedence graph. The start vertex is u_1 and the end vertex u_5. For example, edge (u_1, u_3) in the precedence graph imposes the constraint that u_1 must be visited before u_3. Transitive precedence constraints are shown using dashed lines in the figure.

An alternative definition that is useful sometimes is to tackle the problem as a scheduling problem with precedences, also known as *open* ATSP with precedences. The difference is that the starting vertex is cloned to a dummy vertex $|V|+1$ and precedence constraints are imposed between each existing vertex and the new dummy vertex and the feasible solutions have to start at the start vertex and end at the dummy vertex, respecting the previous constraints as well i.e. visiting all the vertices and fulfilling precedence constraints.

1.3 Outline

In this section, we give a brief outline of the content of the rest of the thesis and its main results.

All in all, the main work of the thesis consists of Chapters 2, 3, 4. Each chapter presents our study and results of one problem, namely the *Orienteering Problem with Stochastic Travel and Service Times*, the *2-stage Capacitated Vehicle Routing Problem with Stochastic Demands* and the *Sequential Ordering Problem*.

In Chapter 2 we study the Orienteering Problem with Stochastic Travel and Service Times problem and the benefits of improving the state-of-the-art. We then proceed by presenting our contributions. Firstly, we introduce four new evaluators that approximate the objective function. Then we study experimentally the performance and accuracy of the evaluators from many different perspectives and find the trade-offs of each one as well as make recommendations for their use. All the new evaluators improve the initial state-of-the-art in the existing datasets of the literature. Additionally, we create and propose larger datasets that we use for further experiments. The experiments concern the behaviour of metaheuristics (Random Search Metaheuristic, Variable Neighborhood Search Metaheuristic) when we embed the different evaluators. We demonstrate the benefits of using the new proposed evaluators and give recommendations about their usage in the context of metaheuristics and also compare and contrast the metaheuristics.

In Chapter 3 we study the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands, a problem that we introduced inspired by an environmental application. We propose four different solution approaches that we analyze and discuss experimentally. The metaheuristic used is the Ant Colony Metaheuristic. Subsequently, we discuss the results and the potential environmental application of the problem.

In Chapter 4 we study exact algorithms for the Sequential Ordering Problem. We begin by testing two different existing algorithms. One was targeting applications in cargo and transportation problems and the other minimizing instruction power switching in compilers. We test them in all domains. From the comparison many improvements in the existing literature came about including closing 9 new instances previously open. Furthermore, the comparison gave insights for an improved algorithm that we developed and present in the rest of the chapter. We validated our results with datasets in the literature.

Please note that this thesis is based on the following publications. Chapter 2 is based on the journal publications: [Papapanagiotou et al., 2014], [Papapanagiotou et al., 2015a] and the conference publications: [Papapanagiotou et al.,

2013], [Papapanagiotou et al., 2016c], [Papapanagiotou et al., 2015b], [Papapanagiotou et al., 2016a], [Papapanagiotou et al., 2016b]. Chapter 3 on the journal publication: [Toklu et al., 2014] and the conference publications: [Toklu et al., 2013] and [Klumpp et al., 2014]. Chapter 4 is based on the conference publications: [Papapanagiotou et al., 2015b], [Jamal et al., 2017]. For all the publications in Chapter 2 and the first in Chapter 4 the author is the main contributor under the guidance and supervision of the corresponding co-authors, while for the publications of Chapter 3 and the second publication of Chapter 4 the author offered substantial contribution.

Chapter 2

Solution Methods for the Orienteering Problem with Stochastic Travel and Service Times

2.1 Introduction

In Stochastic Combinatorial Optimization Problems (SCOPs) the computation of the objective function is often the bottleneck of the computation. This happens because the objective function evaluator is called very frequently by the optimizing functions. This is especially true for metaheuristics, which are our focus. Additionally, the objective function of SCOPs is usually computationally expensive, or even NP-hard to compute as in the case of the Probabilistic Travelling Salesman Problem with Deadlines [Weyland et al., 2012]. Therefore, the study of good approximation methods for the objective function values are essential for obtaining good results especially for bigger datasets. In this section we begin the study of sampling for approximating the objective function of the OPSTS. Firstly, we will study an analytical approximation to the objective function (The Analytical Evaluator) and then we will present a Monte Carlo sampling approximation. The material of this chapter is based on our work published in [Papapanagiotou et al., 2013],[Papapanagiotou et al., 2014], [Papapanagiotou et al., 2016c], [Papapanagiotou et al., 2015b], [Papapanagiotou et al., 2015a], [Papapanagiotou et al., 2016a] and [Papapanagiotou et al., 2016b].

2.2 Objective Function Evaluators

2.2.1 The Analytical Evaluator

The evaluator that is used as a reference evaluator for the objective function of OPSTS is derived analytically from the expression of the objective function (1.8) and it is the one used in [Campbell et al., 2011].

In Section 1.2.2.3 we made the same assumptions as in [Campbell et al., 2011], namely that the travel and service time distributions only differ in a single parameter and the sum of this parameter can characterize the convolution of the arrival time distributions. Also, we assume that the arrival random variables are independently and identically distributed and that the sum of the arrival random variables can be approximated with the same distribution. Furthermore, as in [Campbell et al., 2011] we assume that the distribution of arrival and service times is Γ. The reason for that is because the Γ distribution arises naturally in processes where the waiting times are relevant and it can be conceptualized as a waiting time between Poisson distributed events. Such distribution respects all the previous assumptions mentioned (for more information see [Tanis, 2008]). As shown in Section 1.2.2.3, for the Γ distribution the objective function u of the tour of customers τ becomes:

$$\max v(\tau) = \sum_{i \in \tau} [F_{k_i}(D) r_i - (1 - F_{k_i}(D)) e_i] \tag{2.1}$$

Algorithm 2 shows in detail the implementation of the Analytical Evaluator. The variable $solution$ is a vector representing the indexes of the proposed path of the solution e.g. $[0, 2, 5, \ldots, N]^T$; $|solution|$ represents the size of the solution; D the deadline of the our current instance; k is the shape parameter of the Γ [Tanis, 2008] random variable and also represents the deterministic arrival time; $distance$ is the distance; r_i, e_i, the reward and penalties at node i and v the final objective value. The complexity of the algorithm is $O(|solution|)$ with F_k having a high running time cost.

2.2.2 The Monte Carlo Evaluator (MC)

The Monte Carlo Evaluator is based on Monte Carlo sampling which is essentially a sampling methodology that uses the sample means to estimate the actual population means. In simple terms and in its simplest form we run an experiment that returns success (one) or failure (zero) n times (number of *histories* or *trials*). We then take the ratio of successes $|s|$ to the number of trials p ($\frac{|s|}{p}$). If

Algorithm 2 The algorithmic representation of the function $Analytical$

1: **function** $Analytical(solution, D)$:
2: $k \leftarrow 0$
3: $\upsilon \leftarrow 0$
4: **for all** $i \in [2, \cdots, |solution|]$ **do**
5: $\quad k \leftarrow k + distance(solution[i-1], solution[i])$
6: $\quad \upsilon \leftarrow \upsilon + F_k(D)r_i - (1 - F_k(D))e_i]$
7: **end for**
8: **return** υ

p is sufficiently large then $\frac{|s|}{p}$ gives a good approximation of the expected value of our experiment (the population means). For more information we refer the reader to [Dunn and Shultis, 2011].

The crucial quantity we need to compute the objective function of the OPSTS as seen in Equation (1.8) is $P(A_i \leq D)$. This evaluator achieves it by using Monte Carlo sampling. Firstly, we generate many different fully connected graphs with different arrival times ($\bar{A}_i$) for every node, generated according to the probability distribution that the edge follows. When a solution is given the Monte Carlo Evaluator computes the deterministic objective value of the solution for each scenario and then takes the average value of all the computations. The bottleneck of MC is the generation of random numbers from a specific distribution. To accelerate the procedure the Monte Carlo Evaluator, many samples of travel times from every node to every other node are precomputed. Once we generate these samples we reuse them any time an objective function evaluation is required. Each sample in the precomputation is a matrix with cells that represent a realization of a random variable $X_{i,j}$ that represents the travel time. Such a precomputation matrix can be seen in Figure 2.1. After the precomputation phase, we compute the objective function as described previously. Algorithm 3 shows in detail how samples are precomputed. **RandomΓ**(k, θ) is a function that returns a value from a Γ distribution with parameters k, θ. $distance$ returns the distance between two solution nodes (in this example euclidean). The complexity of the precomputation is $O(n^2 s)$ where n is the number of nodes and s the number of samples. The most time consuming function is **RandomΓ**. Additionally, Algorithm 4 shows in detail how the Monte Carlo sampling is performed after having precomputed samples. The complexity is $O(|solution| \cdot |samples|)$ where $|solution|$ is the length of the solution and $|samples|$ the number of samples which is a constant number. Let us name $P(A_i \leq D) = p_i$. A small optimization that we do in Al-

gorithm 4 is that we compute the sum of all penalties which is a negative value and then we add it back. This is equivalent to our objective function because $\sum_{i\in\tau}[p_i r_i-(1-p_i)e_i] = \sum_{i\in\tau} p_i r_i - \sum_{i\in\tau} e_i + \sum_{i\in\tau} p_i e_i = \sum_{i\in\tau}[p_i(r_i+e_i)] - \sum_{i\in\tau} e_i$. This provides an additional speed benefit because we can precompute the sum of penalties and rewards (r_i+e_i) for every node and avoid a branch in the loop (for the nodes that are served after the deadline) and not examine at all the solution nodes that are after the deadline.

Algorithm 3 The algorithmic representation of the function $precomputeDistances$

```
1: function precomputeDistances(samples, solution)
2: for all sample ∈ [1,...,samples] do
3:    for all i ∈ [1,...,|solution|] do
4:       for all j ∈ [1,...,|solution|] do
5:          k ← distance(solution[i], solution[j])
6:          matrix[sample][i][j] ← RandomΓ(k, θ)
7:       end for
8:    end for
9: end for
10: return matrix
```

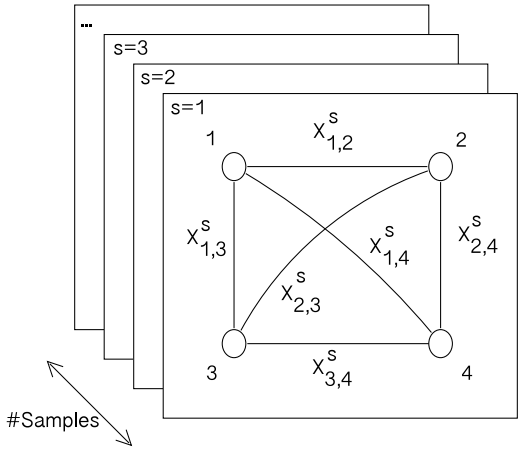


Figure 2.1. Precomputed Samples. Each sample is a fully connected graph with realizations of the arrival times according to their distribution

Algorithm 4 The algorithmic representation of the function MC

function $MC(solution, samples, D)$:
for all $i \in [1, \ldots, samples]$ **do**
  $i \leftarrow 0$
  $v \leftarrow \sum_{i \in \tau}(e_i)$
  $curTime \leftarrow 0$
  while $curTime \leq D$ **and** $i < |S|$ **do**
    $curTime+ = vector[sample][solution[i]][solution[i+1]]$
    $v \leftarrow v + r_{i+1} + e_{i+1}$
    $i \leftarrow i + 1$
  end while
end for
$v \leftarrow \frac{v}{|samples|}$
return u

2.2.2.1 Experimental Results

In this section, we present experiments to assess the usefulness of the Monte Carlo Evaluator (MC) in comparison with the Analytical Evaluator. In order for Monte Carlo sampling to be useful, it must be shown that it can provide an approximation with a reasonable amount of error in a fraction of the time of the Analytical and that the time gains increase over time or size so that our solution method is more scalable.

The implementation was done in C++ and instances ran on a 4-core Intel Core I7-3615QM 2.3GHz using Mac OSX 10.8 but only 1 core was used at a time. The available memory was 16 GB RAM. For the experiments presented in this section, we use the benchmark instances originally introduced in [Campbell et al., 2011]. Two of the datasets are based on the sets first appearing in [Tsiligirides, 1984]. For all the datasets it is assumed that the graph of the customers is fully connected and travel and service times are computed as described in Section 1.2.2.2. The penalty values, are generated as 10% of the corresponding reward so that they match exactly the ones used in [Campbell et al., 2011]. In these experiments, for distances, the Euclidean distance is used and for probability distribution for the travel and service times, the Γ distribution is used. The Γ distribution is computed using the boost math library, which uses Lanczos approximation for the computation [Boost, 2013]. The datasets represent 21, 32, 66 and 64 customers and they are named 221, 432, 566 and 664 respectively.

2.2.2.2 Number of samples in relation to error

To demonstrate the utility of the Monte Carlo Evaluator for evaluating the objective function of the OPSTS, first it needs to be shown that it produces a reasonably low error and consequently we can choose the number of samples that produce such an acceptable error value. For this reason, we run experiments where we measure the relative error of the Monte Carlo Evaluator in relation to the Analytical Evaluator, and we present this value in relation to the number of samples used in Monte Carlo sampling. We vary the number of samples from 100 to 1000 with step 100. The solutions evaluated are generated randomly and then they are optimized by a simple greedy algorithm. Optimization is used because random solutions can be of such a low quality that not even one client can be served without surpassing the deadline. Such cases would bias the comparison between the Monte Carlo approach and Analytical Evaluator when only reasonable solutions are encountered. The greedy algorithm optimizes the order of visiting nodes in a given solution using essentially a nearest neighbour heuristic (see Algorithm 5). The intuition behind this heuristic is that nodes that are closer and have the highest ratio $\frac{r_i}{p_i}$ are better. The complexity is $O(|solution|^2)$. The results of the experiments can be seen in Figures 2.2 and 2.3. The error is evaluated in comparison to the Analytical Evaluator. Solution Size is the number of nodes of each solution given for evaluation. It can be observed that as we use more samples, the error decreases. Furthermore, for the same number of samples, for larger solution sizes the higher relative error is obtained. This can be justified by the fact that the a small error in estimating the distance travelled can decide whether a reward or penalty is given and a wrong decision propagates to subsequent nodes (detailed explanation will follow in the Section 2.2.2.5). Additionally, in both figures it can be seen that the error is constantly below 1.4% even with only 100 samples. Therefore, since the algorithm gives a reasonable approximation of the objective function, it is worth it investigating it further.

2.2.2.3 When the Monte Carlo Evaluator is beneficial

In this section we examine why MC yields bigger time gains the more time it is running. MC sampling as it was described has an overhead which is the precomputation phase. The mean time for executing one evaluation in the course of a number of evaluations $\#Evaluations$ where $SetupTime$ is the precomputation overhead and $MCTime$ the running time of the MC evaluator (as seen in

Algorithm 5 The algorithmic representation of the function $GreedyOptimize$

```
1: function GreedyOptimize(solution):
2: for all i ∈ [0,...,size(solution)−1 do
3:    max ← −1
4:    index ← i + 1
5:    for all j ∈ [i + 1,...,size(solution)] do
6:       curDistance ← distance(solution_i, solution_j)
7:       curValue ← solution_j.reward
8:       if solution_j.penalty > 0 then
9:          curValue ← curValue / (curDistance · solution_j.penalty)
10:      else
11:         curValue ← curValue / curDistance
12:      end if
13:      if curValue > max then
14:         index ← j
15:         max ← curValue
16:      end if
17:   end for
18:   swap(i + 1, index)
19: end for
```

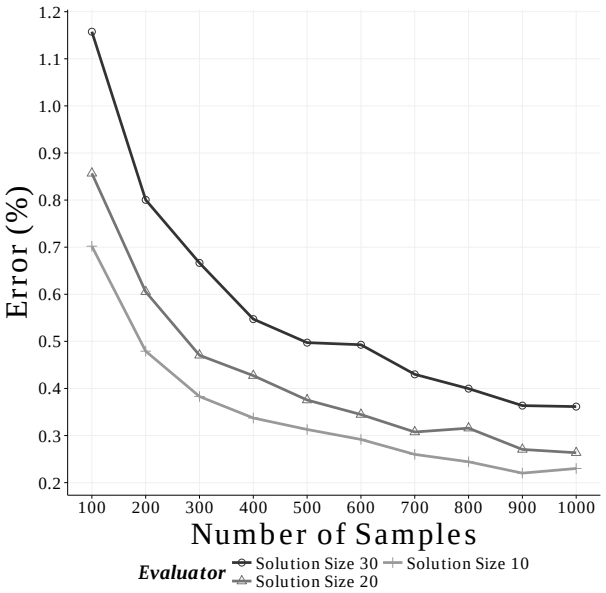


Figure 2.2. Monte Carlo Evaluator Error vs Number of Samples. Different sizes of solutions are considered. The deadline in this set is 50. The size of the dataset is 32 nodes (432 dataset)

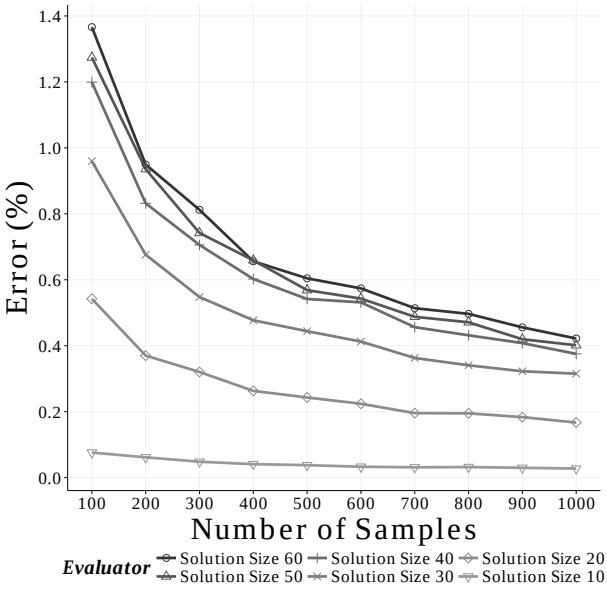


Figure 2.3. Monte Carlo Evaluator Error vs Number of Samples. Different sizes of solutions are considered. The deadline in this set is 50. The size of the dataset is 64 nodes (664 dataset)

Algorithm 4) is:

$$avg\_mc\_running\_time = \frac{mean(SetupTime)}{\#Evaluations} + mean(MCTime) \qquad (2.2)$$

We can observe that in the Equation (2.2) as the number of evaluations gets very large ($\#Evaluations \to \infty$), the fraction $\frac{mean(SetupTime)}{\#Evaluations}$ tends to 0, which means that the $mean(SetupTime)$ is no longer a significant term in the running time. In order to gain intuition in a realistic context, in Figure 2.4 we can see a plot of $\frac{MCTime}{AnalyticalTime}$ versus the number of Evaluations for solution sizes 30 and 60 using 100 and 200 samples for the Monte Carlo Evaluator respectively. $MCTime$ includes the precomputation time and obviously if the fraction is lower than 1, then the mean time for doing one evaluation with the Monte Carlo Evaluator is less than that for the Analytical Evaluator. In the figure we can observe that for solution size 30, if we run the objective function 2000 or more times we start having speed gains and for solution size 60 we need to run it more than 4000 times. Taking into account that in a typical run of a metaheuristic method (in which the Monte Carlo Evaluator will be embedded), performing more than 100000 evaluations is intuitively very common, the Monte Carlo Evaluator has the potential for a considerable speed gain.

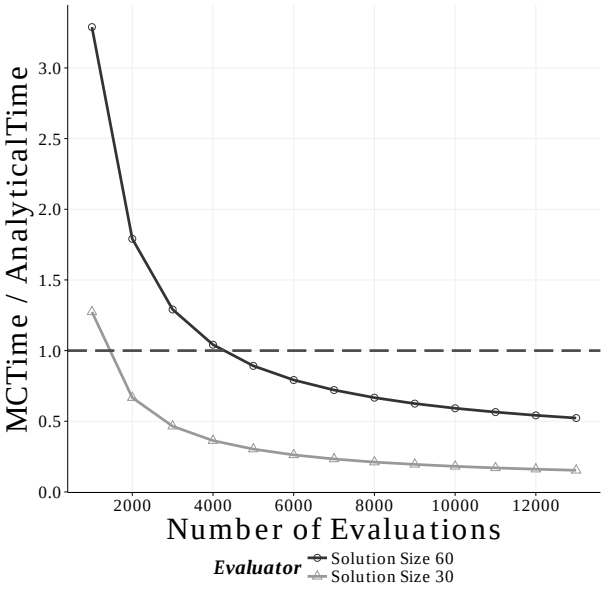


Figure 2.4. $\frac{MCTime}{AnalyticalTime}$ plot, when the plot line is lower than 1 it means that the Monte Carlo Evaluator has speed gains over the Analytical

2.2.2.4 Running Time Comparison

In Table 2.1 we present a running time comparison between the Analytical and MC evaluators. Times are measured in seconds and are taken as an average after running the respective evaluator 100 times. In the first column the number of samples used in the Monte Carlo Evaluator is reported and in the second all the relevant times of our experiment. More specifically $AnalyticalTime$ the average running time of the Analytical Evaluator, $SetupTime$ indicates the running time of creating the precomputation matrix and $MCTime$ the time of an average execution of the MC Evaluator.

The last column contains the result of a ratio. When it is smaller than 1, then the MC evaluator -without taking into account precomputation costs- is faster than the Analytical Evaluator. According to Equation (2.2) when the number of evaluations is large ($\#Evaluations \rightarrow +\infty$), only the cost from the Monte Carlo Evaluator (MC) is important. From the column $MCTime/AnalyticalTime$, we can observe that the time consumed for computing the objective function using MC is a small fraction of the time of the Analytical cost until a certain number of samples. Additionally, the error is in most cases sufficiently small even by using only 100 samples.

2.2.2.5 Monte Carlo Evaluator performance and implications

In the datasets, tested before MC sampling performs well and by tuning the number of samples, the error can be reduced to negligible levels. However, for some of the bigger instances, the errors produced would not reduce to negligible levels with the expected rate, because of unusually bigger errors. This happens because small estimation errors in travel times can decide whether a reward is awarded or a penalty is incurred. A wrong penalty given, will propagate until the node where the deadline really happened. All the intermediate nodes are estimated with a wrong value. If for example an evaluator considers falsely that a node is not visited on time on average then this node will contribute a penalty instead of a reward and the same is true for all the subsequent nodes until the first that is visited after the deadline on average in reality. In Figure 2.5 we can see how the error propagates. In this example “MC Deadline Node” is the node where the MC estimates the deadline will occur while it occurs on the “Deadline Node”. For the nodes from “MC Deadline Node” to “Deadline Node”, the evaluator wrongly assigns a penalty, resulting in computing an objective value of 0 in the end instead of 16.

To alleviate this problem, we examined where in the evaluation the error

Table 2.1. Comparison of running times of the *Analytical* and *MC* for Solution Size 64 and 100 evaluations

#Samples	Average *AnalyticalTime*	Average *SetupTime*	Average *MCTime*	$\frac{MCTime}{AnalyticalTime}$
Solution Size = 30				
100	2.974e-05	0.0760	5.062e-06	0.170
150	3.112e-05	0.113	6.546e-06	0.210
200	2.990e-05	0.152	1.663e-05	0.556
250	3.161e-05	0.190	1.796e-05	0.568
300	3.207e-05	0.226	2.148e-05	0.670
350	3.169e-05	0.264	1.998e-05	0.630
400	3.139e-05	0.302	2.778e-05	0.885
450	3.188e-05	0.339	2.934e-05	0.920
500	2.765e-05	0.377	4.131e-05	1.494
Solution Size = 60				
100	5.344e-05	0.076	6.564e-06	0.123
150	4.949e-05	0.114	7.657e-06	0.154
200	5.084e-05	0.152	1.487e-05	0.292
250	5.061e-05	0.190	1.595e-05	0.315
300	4.963e-05	0.229	2.454e-05	0.494
350	5.064e-05	0.265	1.981e-05	0.391
400	5.060e-05	0.304	2.664e-05	0.526
450	5.226e-05	0.340	3.400e-05	0.650
500	5.135e-05	0.387	4.684e-05	0.912
550	5.102e-05	0.420	6.412e-05	1.257

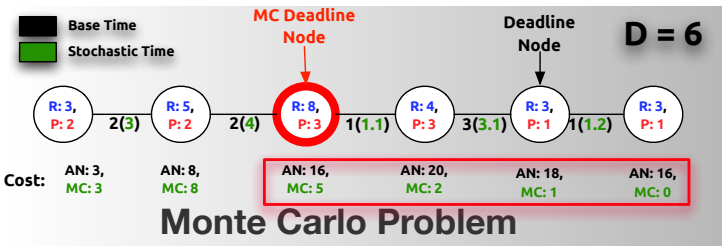


Figure 2.5. Example of propagation of errors when using Monte Carlo sampling. Monte Carlo Evaluator (MC) estimates wrongly the deadline node resulting in wrong objective value estimation

originates most frequently and started using more precise evaluators for these parts. The first such evaluator is examined in the next section.

2.2.3 The MC-Analytical-MC Evaluator (M-A-M)

As it was mentioned in the Sections 2.2.1 - 2.2.2, the Analytical Evaluator can be slow for big datasets and MC while it is faster than Analytical, for bigger datasets it may produce big errors. For that reason, we looked for ways to combine the advantages of both methods.

The Analytical Evaluator can be accelerated while keeping the error low by dividing the solution in parts and evaluating them with different evaluators of a solution. We consider that a subset of the nodes of the solution where the deadline is more likely to occur as critical nodes. We call the subset of these nodes "the Deadline Area (DA)". These nodes are visited on time in some scenarios and too late in others. This may cause propagation of errors. In Section 2.2.2.5, we discussed how using MC can result in big errors in certain datasets. In order to avoid these errors that can be big, we evaluate these nodes using the Analytical Evaluator. To define the DA we select a parameter$\alpha(0 < \alpha < 1)$ and then we include all the nodes with deterministic travel times between $[(1-\alpha)D, (1+\alpha)D]$. The remaining nodes are evaluated using MC. We noticed that when this evaluator is used inside an optimizer, there are usually very few nodes if any after the DA, since optimisers avoid generating solutions with many "penalized" nodes. A graphical representation of M-A-M can be seen in Figure 2.6. The implementation of the M-A-M evaluation can be seen in Algorithm 6. This algorithm combines the previous algorithms of Analytical and MC and has a complexity of $O(|samples| \cdot |solution|)$.

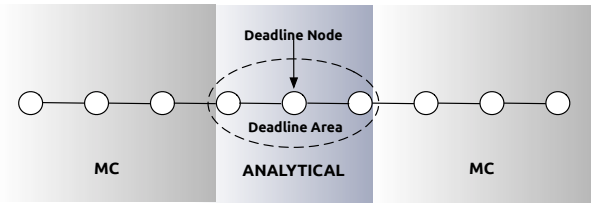


Figure 2.6. The evaluator M-A-M as applied in a solution

2.2.3.1 Samples vs Error for M-A-M

To demonstrate the usefulness of M-A-M, we first show that it produces a reasonably low error and then we choose the number of samples that produce an

Algorithm 6 The algorithmic representation of the function M-A-M

1: **function** M-A-M $(solution, samples, D, \alpha, \rho)$:
2: $DA\_start \leftarrow (1-\alpha) \cdot D$
3: $DA\_end \leftarrow (1+\alpha) \cdot D$
4: $\upsilon \leftarrow \upsilon + MC(solution[0..DA\_start], samples, D)$
5: $\upsilon \leftarrow \upsilon + Analytical(solution[DA\_start+1..DA\_end], D)$
6: $\upsilon \leftarrow \upsilon + MC(solution[DA\_end+1..|solution|], samples, D)$
7: **return** υ

acceptable error value. For this reason, we run experiments where we test both the MC and M-A-M against the Analytical Evaluator. We use 1000 random solutions that we evaluate with different number of samples starting from 1 sample to 300 samples with step 20. In practice, the solutions provided to the evaluation methods tested are of better quality yielding a smaller error. This is because in random solutions it can happen that not even one client can be served without surpassing the deadline. In this experiment, we chose the deadline to be 15 which is small even for our smallest dataset (21 customers) to demonstrate the usefulness of the method even with small deadlines, which tend to be error-prone. With small deadlines we tend to have more errors because of the stochasticity of travel times and the few nodes that can be visited before the deadline. An approximation error can make our objective function consider that the deadline happened before or after the node where it really happened causing errors as explained in Section 2.2.2.5. Taking into account that with small deadlines the solution size tends to be small, every error tends to have bigger impact in the final solution. The random solutions generated are based on the datasets introduced in Section 2.2.2.1 with 21, 32 and 64 clients respectively. The results of the experiments can be seen in Figure 2.7, Figure 2.8 and Figure 2.9. The error is the relative error in comparison with the Analytical Evaluator and it is in percentages. Different line shades and shapes show different evaluators or different configuration of the same evaluator. Because the Monte Carlo Evaluator has very different results in comparison with M-A-M, the y-axis is in logarithmic scale. Let us now give an example of what the percentages of deadline areas mean. For a deadline equal to 10 and a 10% DA, all the nodes with arrival times from 9 to 11 are included in the DA. As we will see in later sections, we try to minimize the DA because it requires more accurate and consequently time-consuming evaluators. We observe that the M-A-M evaluator has significantly lower error than MC for the same number of samples even with a small DA (e.g. 1%). Additionally the

error rate of M-A-M decreases roughly at the same pace as MC as we increase the number of samples and we can further decrease the error significantly by increasing the DA. In Figure 2.7 where there are only twenty one customers with a strict deadline (15) results can be noisy (see line for 5% DA) because a small calculation error can make the objective function assign the wrong penalty or reward and in this case even one wrong node can make a difference.

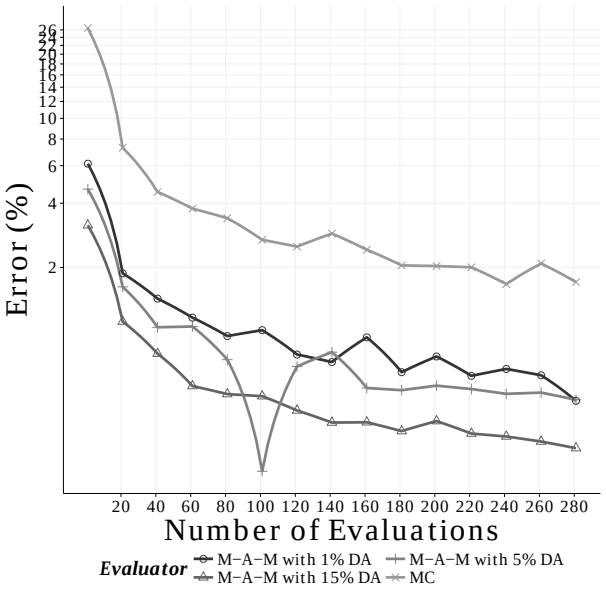


Figure 2.7. Error vs Number of Samples, 21 customers, Deadline = 15

2.2.3.2 When M-A-M is beneficial

The experiment here is analogous to the one in Section 2.2.2.3. When the y-axis is lower than 1 then the mean time for one M-A-M evaluation is less than when using the Analytical Evaluator. In Figure 2.10, Figure 2.11 we can observe that for 21 and 32 customers we have time gains even if we execute the objective function only 500 times. In Figure 2.12 we can observe that for 64 customers, we have time gains for more than 1000 evaluations.

2.2.3.3 Time Gains of M-A-M

In this experiment we examine the time gains that M-A-M and MC evaluators for different number of samples with respect to the Analytical Evaluator. Two different experiments are proposed. In the first we show the time gains according

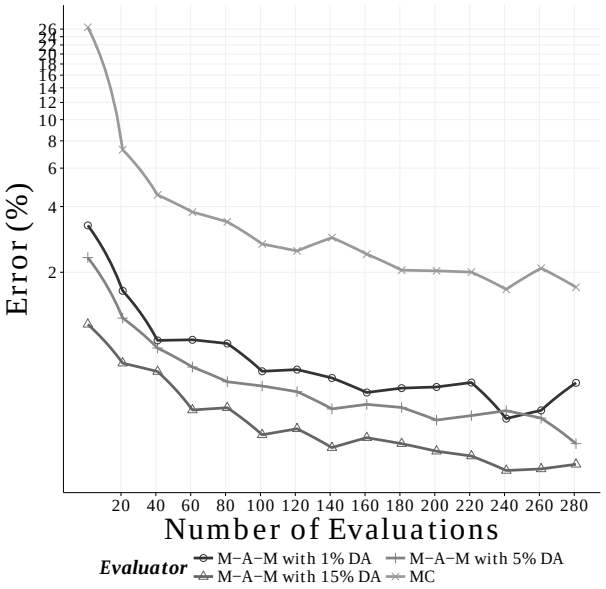


Figure 2.8. Error vs Number of Samples, 32 customers, Deadline = 15

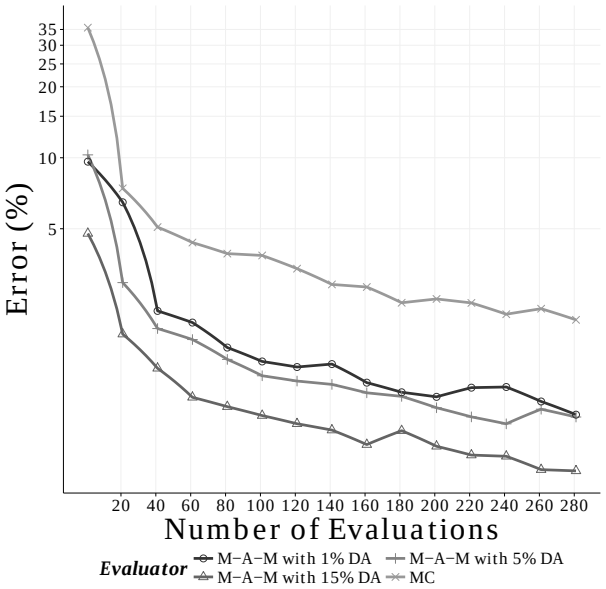


Figure 2.9. Error vs Number of Samples, 64 customers, Deadline = 15

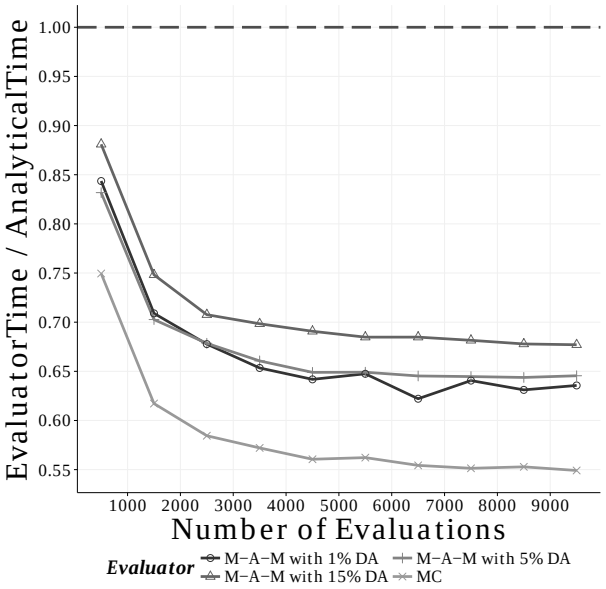


Figure 2.10. Benefit vs #Evaluations, 21 customers, 30 samples

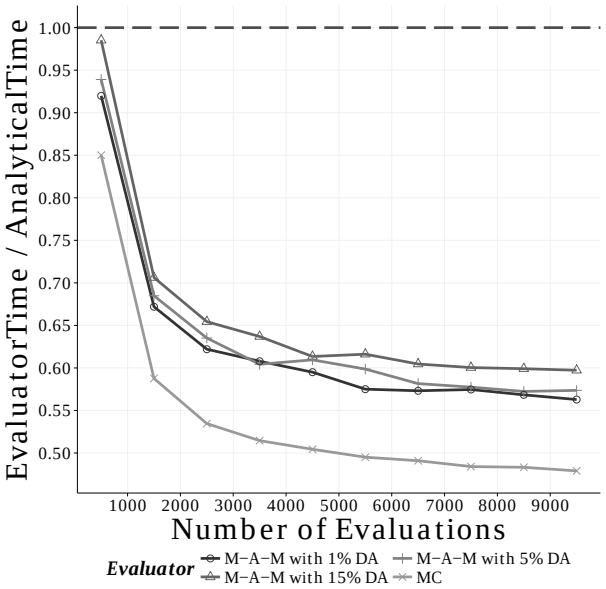


Figure 2.11. Benefit vs #Evaluations, 32 customers, 30 samples

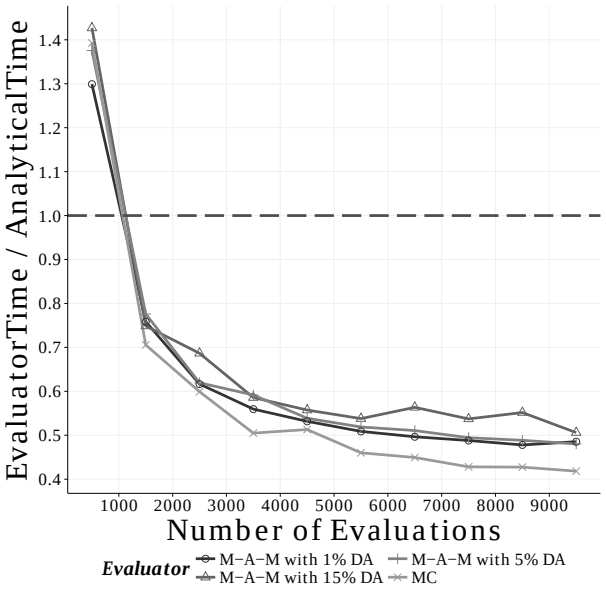


Figure 2.12. Benefit vs #Evaluations, 64 customers, 30 samples

to the number of samples (Figure 2.13, Figure 2.14, Figure 2.15). As in Section 2.2.3.2 different line shades and shapes show different evaluators or different configuration of the same evaluator. For this experiment the deadline is equal to 15 which is restricted for these datasets. For this restricted deadline we can observe from the Figures 2.13, 2.14, 2.15 that we have time gain until using 50 samples in almost all cases. From the Figures 2.7, 2.8, 2.9 we can see that for M-A-M the error is less than 1% (about 0.5%) for 50 samples. By using 30 samples we see that there is time gain in all cases and error less than 1% for 21 and 32 customers and less than 2% for 64 customers.

The next experiment for running time shows the time gain according to the deadline of the problem (Figures 2.16, 2.17, 2.18). As the deadline gets bigger we have larger time gains. This happens because as the deadline gets bigger it becomes less likely to reach the deadline in the proposed solutions and therefore the performance of M-A-M tends to the performance of MC. For this experiment 30 samples were used for the evaluation. The number of samples was selected taking into account the previous experiments in this section that show the efficacy of 30 samples both in terms of time gain and small error. It must be noted that the solutions examined are random and the performance of M-A-M and MC is worse than it would normally be when used within an optimization method as in the random solutions, some very bad solutions are included that make the algorithm

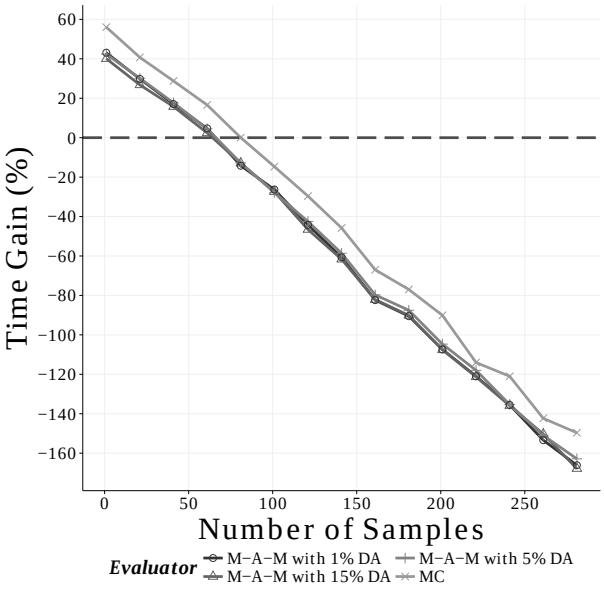


Figure 2.13. Time Gain vs #Samples, 21 customers, Deadline = 15

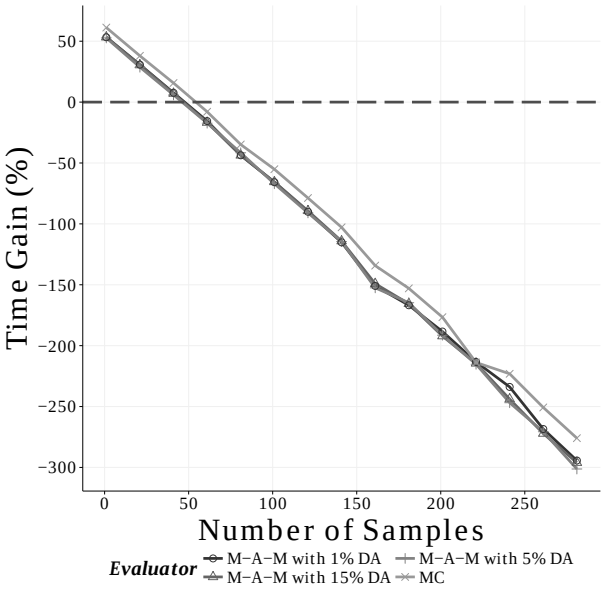


Figure 2.14. Time Gain vs #Samples, 32 customers, Deadline = 15

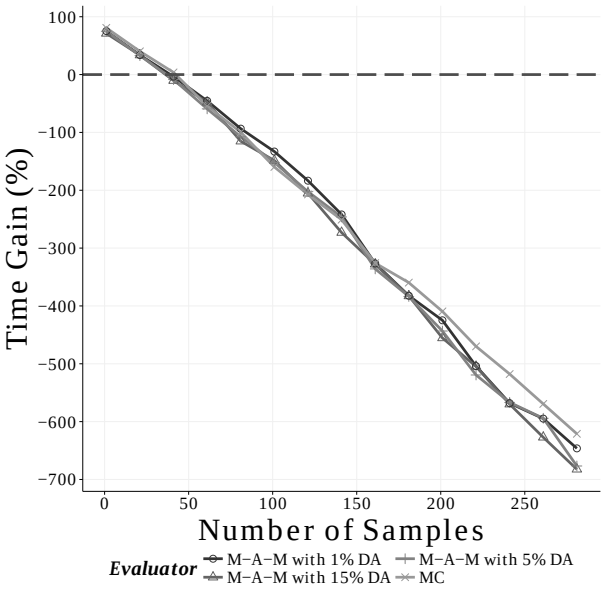


Figure 2.15. Time Gain vs #Samples, 64 customers, Deadline = 15

run slower and be more error prone. However, even under these circumstances we can observe that the M-A-M manages to keep the error rates low enough while still having competitive time gains with MC. From the experiments, we observe that a DA of 5% performs very well in terms of time gain while keeping the error very low, in all cases less than 2% and in most cases less than 1% and therefore it is the suggested evaluation configuration.

Error Thresholds	**1%**	**2%**	**3%**	**4%**
Time Gains	27.29%	30.94%	32.18%	33.05%
DA Ratio (α)	21%	13%	9%	5%

Table 2.2. Time Gains and DA ratios for different error thresholds for M-A-M. Errors are measured with reference to the Analytical Evaluator for the dataset of 66 customers used in the paper

2.2.3.4 Tuning the Deadline Area of M-A-M

In the previous Section by combining the knowledge from Sections 2.2.3.1, 2.2.3.2 and 2.2.3.3 we came to a reasonable recommendation for tuning the DA (parameter α) of M-A-M. In this Section we apply a more rigorous procedure

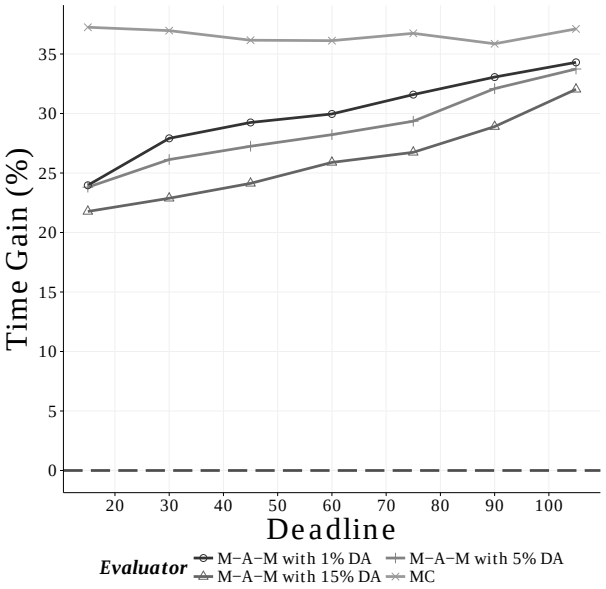


Figure 2.16. Time Gain vs Deadline, 21 customers, Deadline = 15

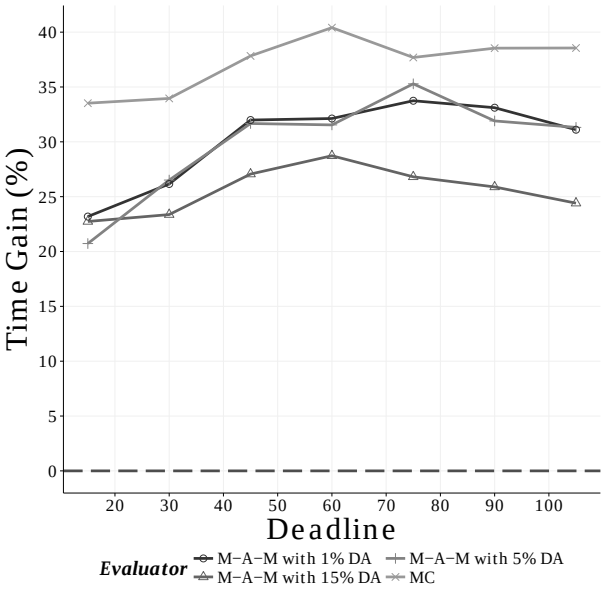


Figure 2.17. Time Gain vs Deadline, 32 customers, Deadline = 15

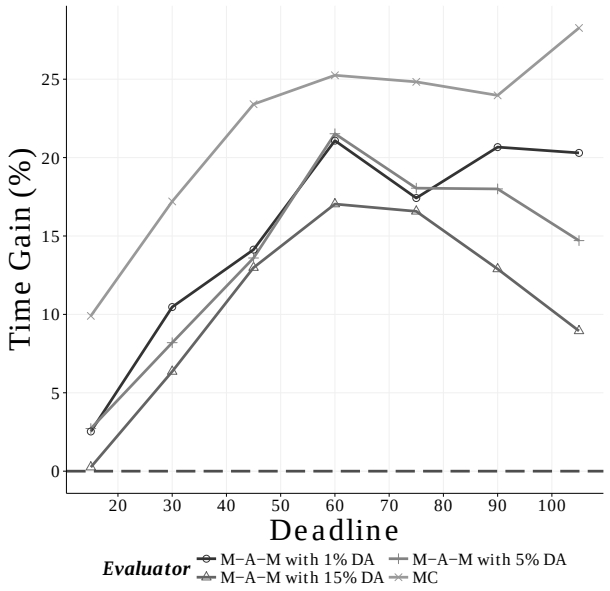
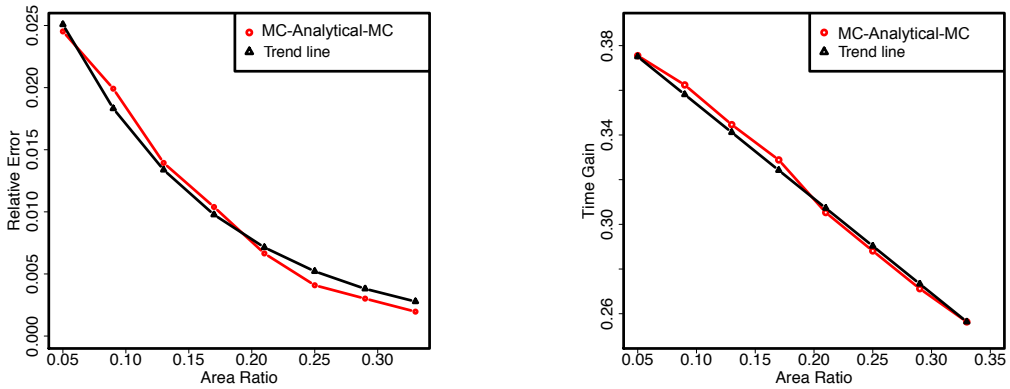


Figure 2.18. Time Gain vs Deadline, 64 customers, Deadline = 15

for tuning the deadline in more realistic situations, more specifically inside the VNS metaheuristic that will be explained in Section 2.4.2. We have shown that the Monte Carlo Evaluator (MC) is faster than the Analytical one and because the DA is evaluated by the Analytical Evaluator we want it to be as small as possible. Therefore, we want to find the minimum DA (value of parameter α) such that for every deadline it gives us a guaranteed threshold for the average error. In Table 2.3, we can see the results for the M-A-M method on a representative instance for different error thresholds. For example, if we want an error below 2% we would select a DA ratio of 13% and it would be 30.94% faster in the evaluation. Time gains and errors are measured with reference to the 'Analytical Evaluator'. Time gains and errors are averages over all the evaluations done in 30 runs of the VNS metaheuristic. The specific metaheuristic is explained in detail in Section 2.4.2.

By using the concept of the DA ratio we can influence the relative error per evaluation and the time gains of our algorithm. In Figure 2.19, we can see two graphs depicting DA ratio vs relative error and DA ratio vs time gains, respectively (dataset with 66 customers). We can observe that the error is decreasing with a higher rate than time gain as we increase the DA ratio. This fact adds to the usefulness of the pursuit for further improvement of hybrid methods of evaluation (like M-A-M). To confirm our observations, we fitted exponential curves ($e^{k \cdot x + \beta}$) to model the error curves and lines to model the time gain curves. The



(a) Residual standard error: 0.0025 (b) Multiple R-squared: 0.99

Figure 2.19. Influence of area ratio α to relative error and time gain for deadline 40, for method M-A-M, dataset with 66 customers. Errors and time gains are measured with reference to the Analytical Evaluator. The trend lines (in black) are the fitted curves and model the trend. The fitted exponential curve is $y = e^{-3.56 \cdot x - 6.77}$ and the fitted line is $y = -0.57 \cdot x + 0.45$

average residual standard error for the error curves was 0.0025 and multiple R^2 for the time gain curves was 0.99. This means that in the first case that there exist exponential curves that approximate the actual error curves with negligible error and in the second case, lines can replace the time gain curves and they would explain 99% of the variance of time gains. The implication of that is that by increasing the DA we can trade-off a small loss of time gain (linear) for a large decrease in error (exponential).

Error Thresholds	**1%**	**2%**	**3%**	**4%**
Time Gains	27.29%	30.94%	32.18%	33.05%
DA Ratio (α)	21%	13%	9%	5%

Table 2.3. Time Gains and DA ratios for different error thresholds for M-A-M. Errors are measured with reference to the Analytical Evaluator for the dataset of 66 customers used in the paper

2.2.4 Reward-MC-Analytical-MC-Penalty Evaluator (R-M-A-M-P)

The R-M-A-M-P has been designed to enhance the speed of M-A-M (introduced in Section 2.2.3) further by assuming that nodes very far from the DA with very high probability either receive a reward or induce penalty. Therefore, in a similar manner with the DA, we define the Reward Area and penalty area where the nodes get reward and penalty respectively without any computation. To define the R-M-A-M-P areas we need as in M-A-M, the α and $\rho(\alpha \leq \rho \leq 1)$. The nodes with expected visit time in the interval $[0,(1-\rho)D)$ are in the Reward Area and get an automatic reward, the nodes in $[(1+\rho)D,+\infty]$ (where $+\infty$ is the last node of the solution) are in the PENALTY area and automatically get a penalty. As in M-A-M the nodes with expected visit time in $[(1-\alpha)D,(1+\alpha)D]$ which we name as DA are evaluated using the Analytical Evaluator and the rest of the nodes are evaluated using Monte Carlo sampling. Inside an external algorithm this method tends to have a very small to non-existent second MC and PENALTY areas. R-M-A-M-P provides us with fine-grained control and in the majority of the times can perform better than M-A-M. In Algorithm 7 we see the implementation of R-M-A-M-P in pseudocode. This algorithm combines the previous algorithms of Analytical and MC as well as $RewardCost$ and $PenaltyCost$ and has a complexity of $O(|samples| \cdot |solution|)$.

A graphical representation of R-M-A-M-P can be seen in Figure 2.20.

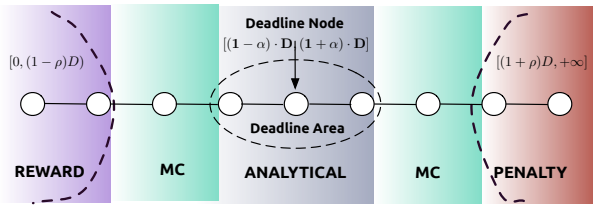


Figure 2.20. The evaluator $Reward - MC - Analytical - MC - PenaltyEvaluator$ as applied in a solution

2.2.5 The Reward-Analytical-Penalty Evaluator (R-A-P)

This evaluator can act as a baseline evaluator as no stochasticity is involved. Similarly to M-A-M we define a parameter $\alpha(0 \leq \alpha \leq 1)$ and the nodes visited at expected time in $[(1-\alpha)D,(1+\alpha)D]$ of the solution under consideration are evaluated using the Analytical Evaluator. Before the DA the nodes of the solution

Algorithm 7 The algorithmic representation of the function R-M-A-M-P

1: **function** R-M-A-M-P($solution, samples, D, \alpha, \rho$):
2: $DA\_start \leftarrow (1-\alpha) \cdot D$
3: $DA\_end \leftarrow (1+\alpha) \cdot D$
4: $MCAreaBefore \leftarrow (1-\rho) \cdot D$
5: $MCAreaAfter \leftarrow (1+\rho) \cdot D$
6: $\upsilon \leftarrow RewardCost(solution[0..MCAreaBefore], D)$
7: $\upsilon \leftarrow \upsilon + MC(solution[MCAreaBefore..DA\_start], samples, D)$
8: $\upsilon \leftarrow \upsilon + Analytical(solution[DA\_start..DA\_end], D)$
9: $\upsilon \leftarrow \upsilon + MC(solution[deadlineAreaAfter..MCAreaBefore], samples, D)$
10: $\upsilon \leftarrow \upsilon + PenaltyCost(solution[MCAreaAfter..|solution|], D)$
11: **return** υ

are in the Reward Area and the ones after the deadline are in the penalty area. Although this evaluator was defined as a baseline evaluator for experimental reasons, it was found that it performs surprisingly well in the cases where the deadline occurs rarely and a small Analytical part is enough to keep the error low (see Section 2.3.1). The implementation of its building blocks can be seen in Algorithms 8 and 9 and the actual R-A-P method is formalized in Algorithm 10. A graphical representation of the $R-A-P$ can be seen in Figure 2.21. R-A-P algorithm combines Analytical, $RewardCost$ and $PenaltyCost$ and has a complexity of $O(|samples| \cdot |solution|)$.

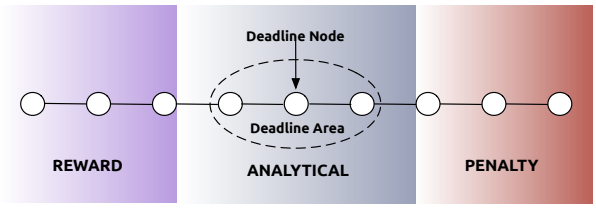


Figure 2.21. The evaluator $R-A-P$ as applied in a solution

2.3 Performance of the Evaluators

In the following Section we will examine the performance of the defined evaluators in datasets of 21, 32 and 64 customers as introduced in the previous sections. In order to compare the usefulness and the performance of the evaluators, we

Algorithm 8 The algorithmic representation of the function $RewardCost$

```
1: function RewardCost(solution, D):
2: υ ← 0
3: for all i ∈ [1, ··· , |solution|] do
4:     υ ← υ + r_i
5: end for
6: return υ
```

Algorithm 9 The algorithmic representation of the function $PenaltyCost$

```
1: function PenaltyCost (solution, D):
2: υ ← 0
3: for all i ∈ [1, ··· , |solution|] do
4:     υ ← υ − e_i
5: end for
6: return υ
```

use two main metrics: speedup and relative error. Both metrics relate to the performance of the evaluator relatively to the Analytical Evaluator. We define speedup as follows:

$$Speedup = \frac{t_{Analytical}}{t_{Evaluator}} \tag{2.3}$$

where $t_{Analytical}$ is the total runtime of the Analytical Evaluator and $t_{Evaluator}$ the runtime of the evaluator examined respectively.

If for example the result of speedup is 1.5, it means that the evaluator we examine is 1.5 times faster than the Analytical Evaluator while evaluating the same solution. We define the relative error as follows:

$$Relerror = \left| \frac{u_{Analytical} - u_{Evaluator}}{u_{Analytical}} \right| \tag{2.4}$$

where $u_{Analytical}$ is the value from the analytical objective function evaluator and $u_{Evaluator}$ is the value returned by the examined objective function evaluator.

2.3.1 Tuning and comparison

We get measurements for different parameter configurations. First we specify the DA ratio range (DA range) and the Reward-Penalty area ratio range (RP range). Here the DA range is $[0.1, 0.7]$ with step 0.1 and RP range is $[0.7725, 0.99]$ with

Algorithm 10 The algorithmic representation of the function R-A-P

1: **function** R-A-P($solution, samples, D, \alpha, \rho$):
2: $DA\_start \leftarrow (1-\alpha) \cdot D$
3: $DA\_end \leftarrow (1+\alpha) \cdot D$
4: $\upsilon \leftarrow \upsilon + RewardCost(solution[0..DA\_start], samples, D)$
5: $\upsilon \leftarrow \upsilon + Analytical(solution[DA\_start..DA\_end], D)$
6: $\upsilon \leftarrow \upsilon + PenaltyCost(solution[DA\_end..|solution|], samples, D)$
7: **return** υ

step 0.0725. These intervals are considered safe and should cover all the meaningful configurations. Then we run the Variable Neighborhood Search Metaheuristic (see Section 2.4.2) which will be using the Analytical Evaluator and take 15000 different solutions produced. Each of these solutions is then evaluated by each evaluator 30 times and we obtain the mean value of the two metrics Speedup and Relerror for each evaluator. We repeat this evaluation procedure for each combination (configuration) of DA ratio and reward-penalty ratio in the DA and RP range. For each method we select the configuration with the best average speedup with the constraint that the average relative error over all deadlines is below a reasonable threshold (*best avg methodology*); or for each method we eliminate all entries that do not comply with a given error threshold and for each deadline we select the configuration that yields the best speedup (*best methodology*). Both methodologies will be explained in more details in below.

In our experiments, we use the same datasets introduced in Section 2.2.2.1. Here we present only the most interesting results. To see the full results of these and other 3 datasets please visit [Papapanagiotou, 2016]. In the experiments that follow, we want to compare the performance of the different hybrid evaluators. As it was discussed in Section 2.3 the metrics that are of importance are speedup and relative error (Equation (2.4) and Equation (2.3)). The desirable outcome is to select an evaluator with minimum relative error and maximum speedup. Optimizing for the two objectives simultaneously can be difficult since to reduce the relative error some speed is usually traded-off. However, considering that we want the objective function to be used inside a metaheuristic optimizer, it is often more important to have a faster metaheuristic so that more comparisons can be made for a given unit of time than having the minimum error possible. Some evidence of this phenomenon can be found in Section 2.2.3.4. Thus, in our experiments we fix a threshold for error and then compare which evaluator reaches the best speedup for this threshold. We use two ways of fixing the thresh-

old for the error. The first method is called *best avg* in the graphs and selects the configuration of which the average speedup over all examined deadlines is maximum and the average relative error over all examined deadlines is less than the error threshold. The second method is named *best* in the graphs and selects the best speedup by deadline with the constraint that the relative error is less than the error threshold. We present two types of graphs, one that shows the best performance of the method (speedup) given the above by deadline and a boxplot that shows the speedup performance of each method over all deadlines. For four error thresholds (0.5%, 1%, 2%, 5%) we plot both types of graphs for both methodologies (best avg and best) and present the best results for each threshold. In general, presenting best avg should be preferred if the results are not much different than best, since it imposes smaller overhead in its computation. As it is clarified in a previous section, both metrics are relative to the Analytical Evaluator and one useful observation is that all the methods are statistically significantly faster than the Analytical even with the strictest error thresholds.

2.3.1.1 Dataset 221

The DA range and RP range is the one defined in Section 2.3.1. For an error threshold of 0.5% (Figure 2.22) for the best methodology we can observe that the R-A-P method is more performant than the others for every deadline. For an error of 1% (Figure 2.23), R-M-A-M-P is best on average for the best methodology. For an error of 2% (Figure 2.24) and the best avg methodology, the MC-Analytical-MC Evaluator is has the best speedup on average. For an error of 5% (Figure 2.25) MC is significantly better than the other methods but an error of 5% usually is too high and can impact the metaheuristic in many negative ways. For small datasets such as this one R-M-A-M-P is the best for 1% error and performs close to the best in the other cases except for the case where we accept large errors (5%)

2.3.1.2 Dataset 432

In this dataset,we observe that for error thresholds of 0.5% and 2% R-M-A-M-P is better on average (Figure 2.26 and Figure 2.28). For 2% error threshold both M-A-M and R-M-A-M-P are significantly better than R-A-P (Figure 2.28. For error threshold of 1% M-A-M is better on average and significantly faster than R-A-P (Figure 2.27). For error threshold of 5% MC is statistically significantly better than any other method (Figure 2.29) Again, for this small dataset for small error tolerance R-M-A-M-P is the best or close to the best evaluator but for higher error

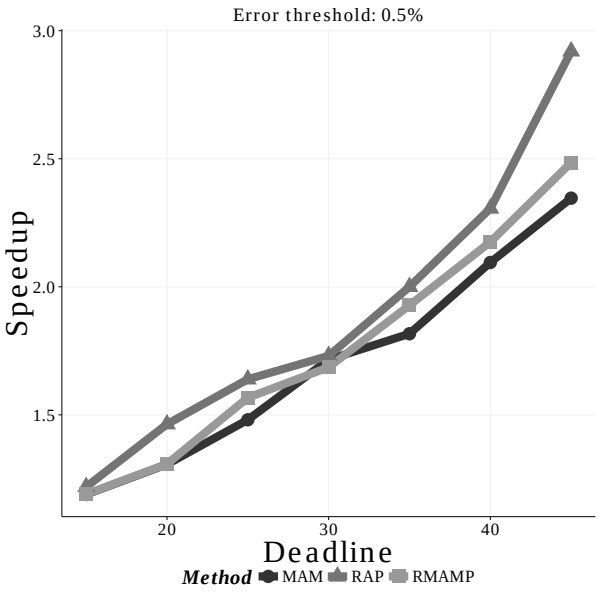


Figure 2.22. Deadline vs Speedup for dataset 221 (21 customers), tuning by each deadline separately (best). Error threshold 0.5%

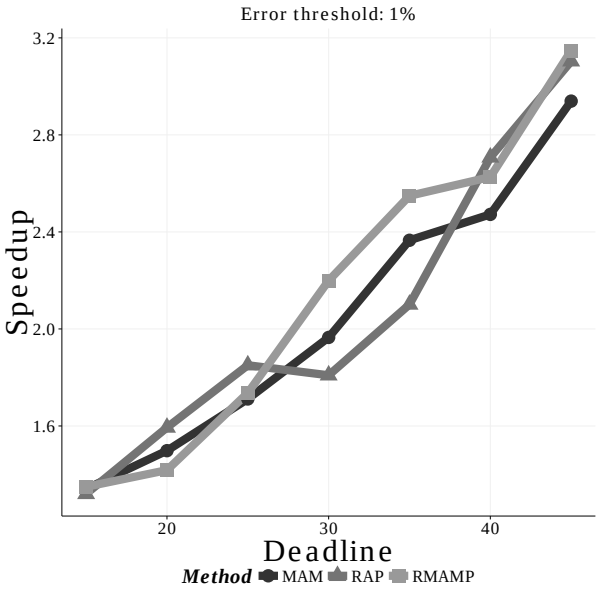


Figure 2.23. Deadline vs Speedup for dataset 221 (21 customers), tuning by each deadline separately (best). Error threshold 1%

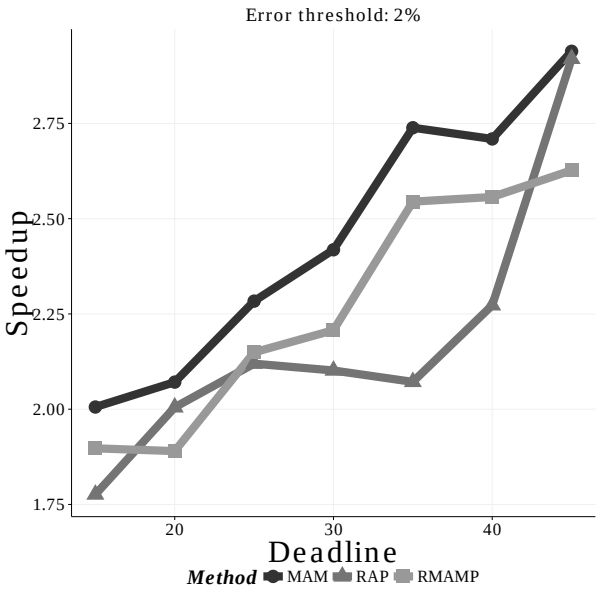


Figure 2.24. Deadline vs Speedup for dataset 221 (21 customers), tuning by according to average relative error and speedup (best avg). Error threshold 2%

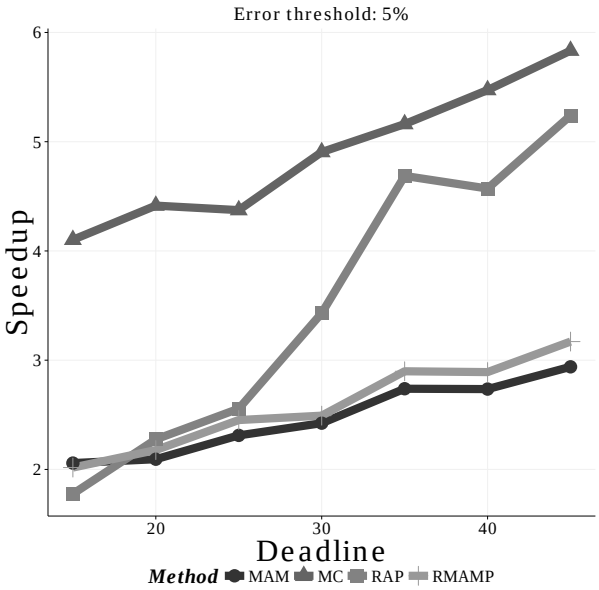


Figure 2.25. Deadline vs Speedup for dataset 221 (21 customers), tuning by each deadline separately (best). Error threshold 5%

tolerance the overhead of splitting the solution and using different evaluators for each part makes it relatively slower.

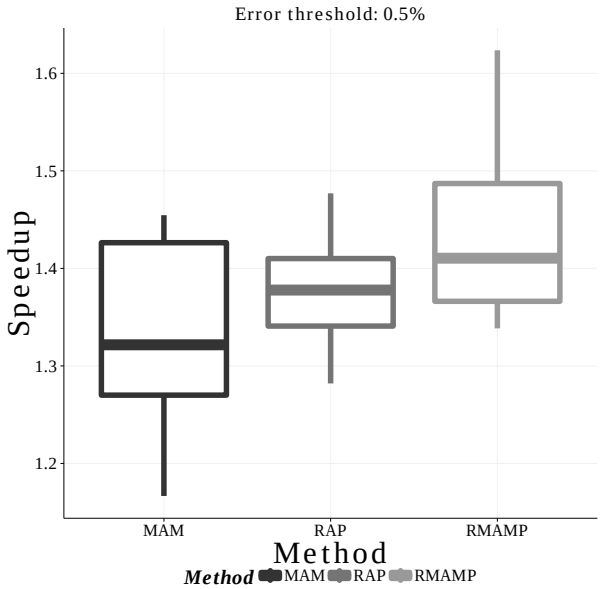


Figure 2.26. Method vs Speedup for dataset 432 (32 customers), tuning by according to average relative error and speedup (best avg). Error threshold 0.5%

2.3.1.3 Dataset 664

We observe that from a certain deadline onwards, R-A-P is much faster than the other methods. This occurs because in this instance, the Analytical part of R-A-P is enough to keep the error very low and from a certain deadline upwards, the deadline is rarely occurring needing a very small Analytical part to achieve good results. This dataset is a good use case for the utility of a method like R-A-P which does not include any sampling. We also observe that in some graphs the Monte Carlo Evaluator is present but with missing points (Figure 2.30, Figure 2.31, Figure 2.32, Figure 2.33). This means that for the specific deadlines there was no result below the selected error threshold. Therefore, when the deadline and the structure of the solution (travel time $X_{i,j}$ between nodes) is such that most of the time the selected clients are served without the deadline occurring, $R-A-P$ outperforms other evaluators.

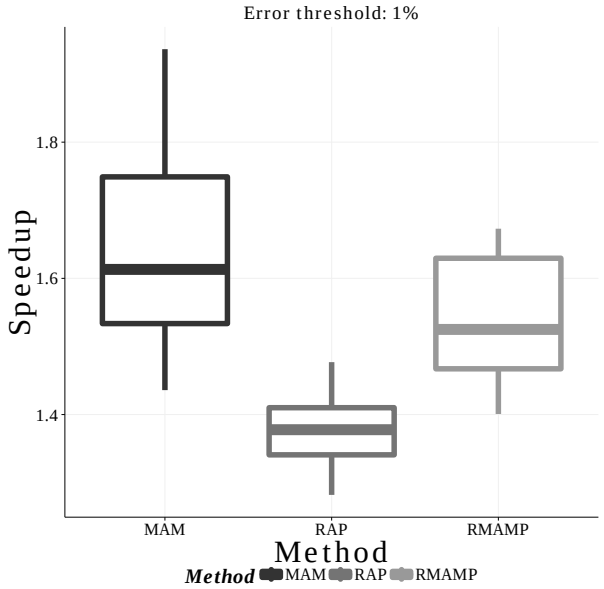


Figure 2.27. Method vs Speedup for dataset 432 (32 customers), tuning by according to average relative error and speedup (best avg). Error threshold 1%

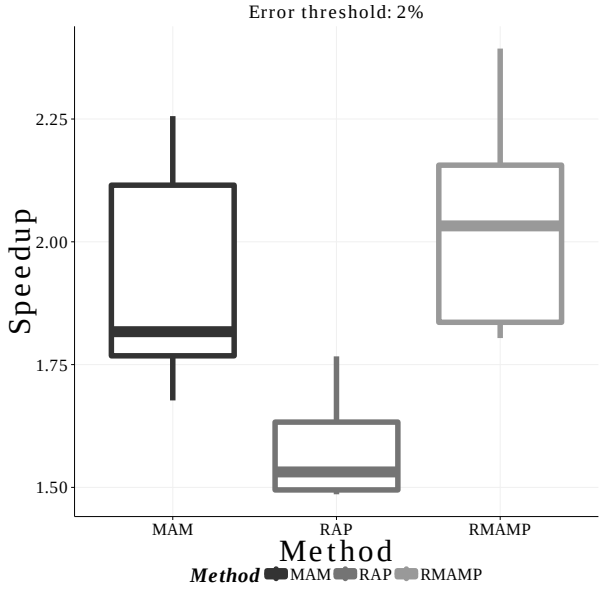


Figure 2.28. Method vs Speedup for dataset 432 (32 customers), tuning by according to average relative error and speedup (best avg). Error threshold 2%

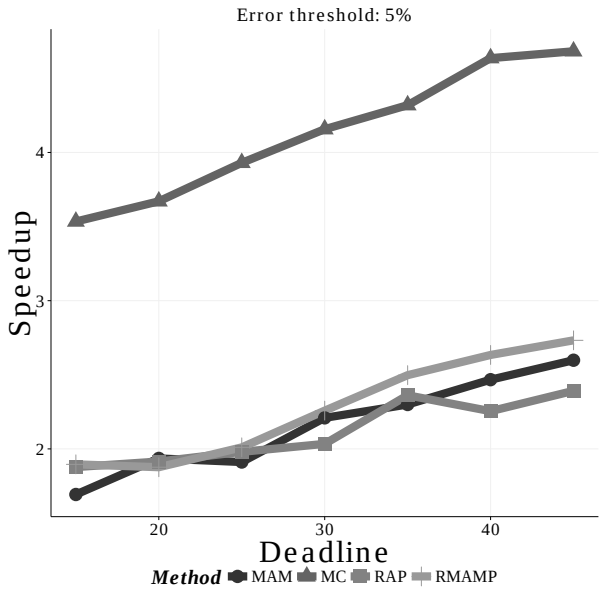


Figure 2.29. Deadline vs Speedup for dataset 221 (21 customers), tuning by each deadline separately (best). Error threshold 5%

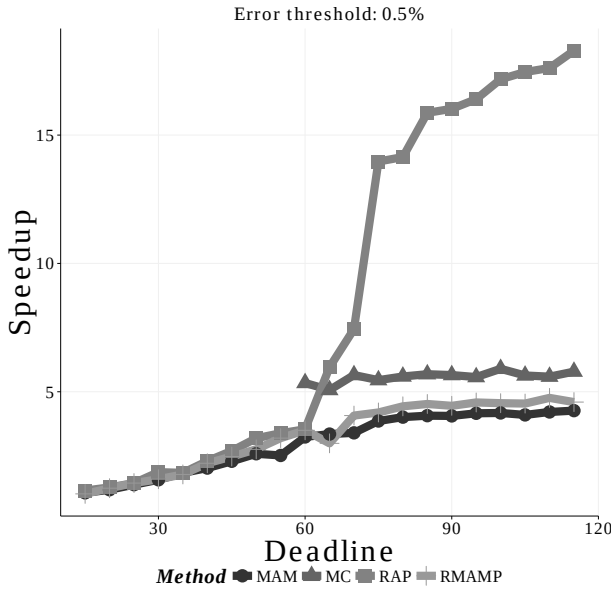


Figure 2.30. Deadline vs Speedup for dataset 664 (64 customers), tuning by each deadline separately (best). Error threshold 0.5%

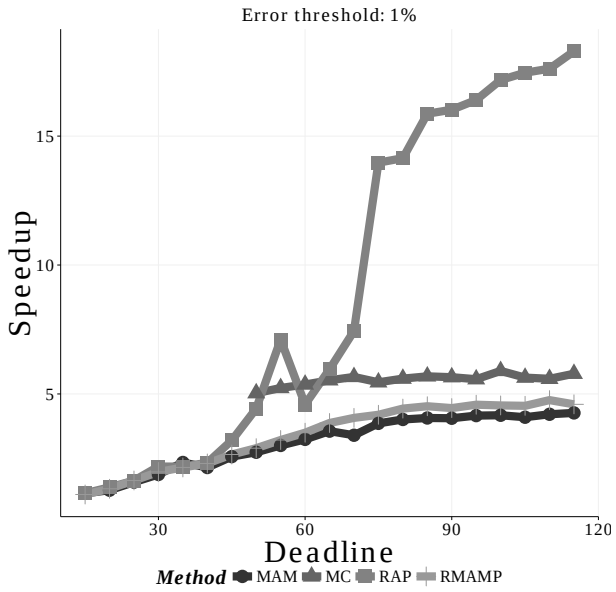


Figure 2.31. Deadline vs Speedup for dataset 664 (64 customers), tuning by each deadline separately (best). Error threshold 1%

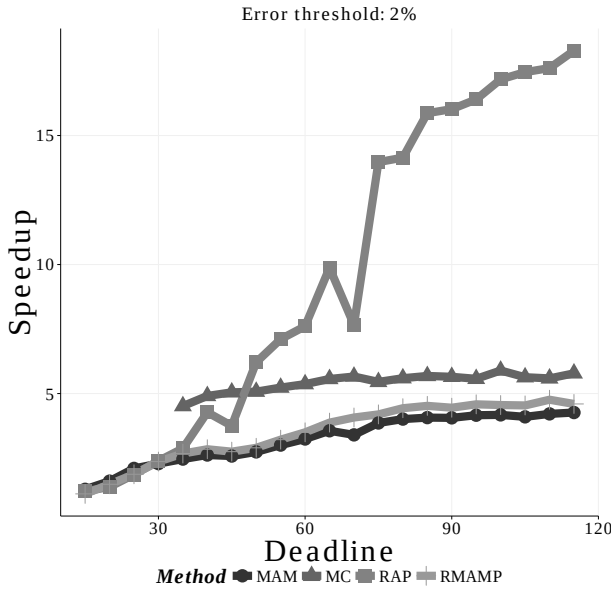


Figure 2.32. Deadline vs Speedup for dataset 664 (64 customers), tuning by each deadline separately (best). Error threshold 2%

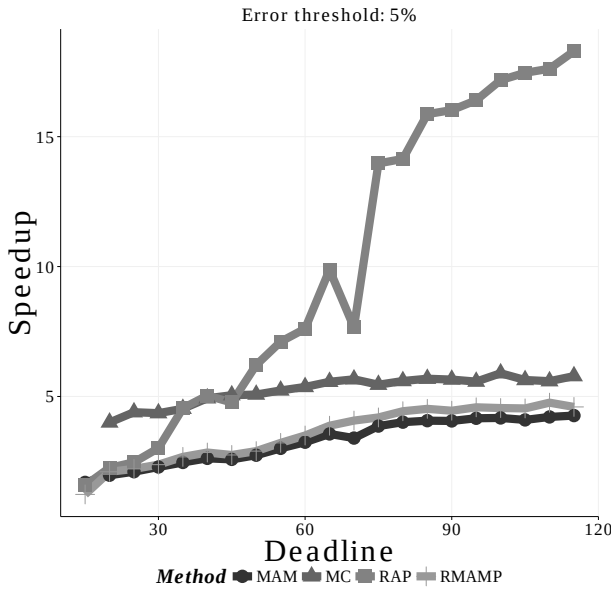


Figure 2.33. Deadline vs Speedup for dataset 664 (64 customers), tuning by each deadline separately (best). Error threshold 5%

2.3.1.4 General Remarks and Guidelines on Tuning

Using the representative results presented and also the full results [Papapanagiotou, 2016], one can see that all methods are useful and significantly faster than the Analytical Evaluator. According to the needs of the application we showed two methodologies on how to select the best-suited method. R-M-A-M-P usually yields consistent results and because it provides fine-grained control for balancing error and speedup, it is in many cases the method of choice for high performance with low error. M-A-M is more accurate but slower than MC and it beats R-M-A-M-P in limited cases where the M-A-M part of the solution is enough to yield good results and the overhead to divide the solution in more parts (five in the case of R-M-A-M-P, three in the case of M-A-M) makes R-M-A-M-P less efficient. R-A-P is very efficient when tuned by each deadline separately in datasets where a small Analytical part is enough to reduce the error of evaluation to minimal levels. Usually, this happens in datasets where the deadline is large enough and occurs rarely, In this case, most of the solution is evaluated using the deterministic Reward part, which is very fast. MC is usually the method of choice when large error is acceptable and the deadline occurs often. In most of these cases MC is significantly better than the other methods, however, usually the error threshold in these cases is unacceptable for use in a metaheuristic.

2.3.2 Generation of large datasets

The datasets that have been examined till now were small and it has been quite trivial to find and evaluate solutions for them. Therefore using these datasets is difficult to reveal the full potential of the proposed faster evaluators. Additionally, in practice many times the algorithm has to be applied to bigger instances. The OPSTS occasionally needs to be applied to problems that have potentially thousands of nodes like production scheduling where we want to use machines with different operations, to choose their movements to produce as many products as possible while minimizing cost, one example of that is the steel rolling mill problem [Balas, 1995]. Therefore, it would be worthwhile to test the proposed algorithms in bigger datasets. To test our methodology we generate four datasets with thousands of nodes following a procedure similar to the one used to create the previous datasets that we used, see [Chao et al., 1996]. More specifically, we generate datasets of 1000, 1800, 2500, 4000 nodes as follows: We assume a square with a diagonal of 1000, 1000, 2500, 4500 units respectively. This is the maximum distance between nodes; we generate uniformly in both coordinates 1000, 1800, 2500, 4000 nodes correspondingly, with random rewards from 1 to 100; and finally penalties that are 0.1 of the reward of each node. The datasets are available in the online addendum [Papapanagiotou, 2016].

2.4 Metaheuristic Algorithms

In this section we present metaheuristics for providing solution for OPSTS.

Global Optimization Algorithms

The general global optimization problem is defined as:

$$\min_{x \in S} f(x) \tag{2.5}$$

where x is a vector of n variables of an n-dimensional feasible region S. We assume S to be non-empty. f is a real-valued function defined over S and the objective is to find the value $x \in S$ that minimizes f.

To find solutions for f we use optimization algorithms. Here we introduce the global optimization algorithms.

The characteristic of global optimization algorithms is that if we run them long enough (mathematically running time $t \to \infty$), they will eventually find the global optimum. This can be guaranteed by ensuring that at the limit, every location in the search space will be visited.

Metaheuristics

Metaheuristics represent a family of approximate optimization techniques that provide "acceptable" solutions in a reasonable time for hard and complex problems (usually NP-hard). They are a framework or heuristic for selecting or combining heuristics that may provide a sufficiently good solution to an optimization problem. Metaheuristics in general do not guarantee the optimality of the solutions they find nor define how close are the obtained solutions from the optimal ones. Some of them are proved to converge at the best solution given a long enough run (mathematically running time $t \rightarrow \infty$) and these ones can be considered as global optimization algorithms.

2.4.1 Random Search Metaheuristic (RS)

The RS is known for its computational efficiency and generality [Spall, 2005]. For the purposes of fair comparison, we devised a Random Search Metaheuristic for the OPSTS.The idea behind the RS is that we produce many random solutions and select the best one. RS is a global optimization algorithm

Another advantage of RS for our experiments is that it can be a very fair metaheuristic for comparing the evaluators that we embed, especially if we initialize it with the same random seed and obtain the same sequence of random solutions. By embedding different evaluators but with the same random solution pool each time, the fastest evaluator will be able to examine more of the solutions of the solutions pool. Also, a small error in evaluation counts to avoid picking suboptimal solutions as best. With a different metaheuristic we could not have this advantage since the different choices different choices influenced by the evaluators may favour constructing and examining a very different solution pool

In Algorithm 11 we can see that the RStakes as input the search space and the evaluator to be used and returns the best solution found. We consider that there exists a function Random_Solution that returns a random solution from the search space. Additionally, RS is good as a reference to judge the efficiency of other metaheuristics. Since it is simple and does not use additional complex heuristics, it is a logical choice for implementing it first and comparing each subsequent heuristic with it. For more information see [Solis and Wets, 1981], [Brownlee, 2011]. Finally, by changing the way a solution is generated (e.g. using local search heuristics) or the acceptance criterion (e.g. stochastically accept suboptimal solutions as best) we can produce different metaheuristics like the one presented in Section 2.4.2.

Algorithm 11 Random Search Metaheuristic

```
1: function RS (search_space, Evaluator):
2: max ← −∞
3: best_solution ← Random_Solution(search_space)
4: # Stopping Criterion
5: while there is time available do
6:    # Generator
7:    candidate_solution ← Random_Solution(search_space)
8:    # Evaluation
9:    candidate_value ← Evaluator(candidate_solution)
10:   # Acceptance Criterion
11:   if candidate_value ≥ max then
12:      max ← candidate_value
13:      best_solution ← candidate_solution
14:   end if
15: end while
16: return best_solution
```

2.4.2 Variable Neighborhood Search Metaheuristic (VNS)

We use the VNS as a metaheuristic to compare it with the seminal paper of OP-STS [Campbell et al., 2011]. In [Campbell et al., 2011], the VNS metaheuristic used only the Analytical Evaluator and here for the first time, we present its performance when we incorporate other evaluators. VNS was first proposed by [Mladenović and Hansen, 1997] and an application to the orienteering problem can be found here [Sevkli and Sevilgen, 2006]. The basic idea is to try to escape local minima by changing search neighbourhoods.

Algorithm 12 shows our implementation of the VNS algorithm based on the one presented in [Campbell et al., 2011]. VNS has two phases the shaking phase and the local search phase. When it is in the shaking phase, a given solution is perturbed according to the neighborhood given as input. In our implementation this is done by the *Shake* function that takes as input the current best solution, perturbs it according to the k neighborhood and returns the result. In the local search phase, the solution from the shaking phase is improved contributing to the exploitation of the search space. For the local search phase Variable Neighborhood Descent (VND) is used which can be seen in Algorithm 13. In VND, a specific neighbourhood is searched until no improving solution can be found and then it proceeds to do the same thing with every available neighborhood

until there are no more improvements while iterating over the neighborhoods. The $BestImproving$ function finds and returns the best solution in the neighborhood k of a given solution $solution$. The neighborhoods used are the ones used in [Campbell et al., 2011], first suggested by [Feillet et al., 2005] namely resequencing the route using 1-shift, replacing a customer on the route with one not on the route, adding a customer on the route, deleting a customer from the route and the ruin and recreate neighborhood from [Schrimpf et al., 2000]. More specifically the ruin and recreate heuristic, removes $\lfloor \frac{nk}{10} \rfloor$ nodes where n is the number of nodes of the current tour and k is the current iteration of VNS. We then add random nodes not previously on the tour. For more details one should consult [Campbell et al., 2011].

Algorithm 12 Variable Neighborhood Search Metaheuristic

```
1: function VNS (search_space, Evaluator):
2: solution ← Random_Solution(search_space)
3: # k is the neighborhood number
4: k ← 1
5: while there is time available do
6:    perturbed_solution ← Shake(solution, k)
7:    VND_solution ← VND(perturbed_solution)
8:    if Evaluator(solution) > Evaluator(VND_solution, Evaluator) then
9:       if k < number_of_neighborhoods then
10:         k ← k + 1
11:      else
12:         k ← 1
13:      end if
14:   end if
15: end while
```

2.4.2.1 Performance of R-M-A-M-P in VNS

R-M-A-M-P described in Section 2.2.4 is the most versatile evaluator (in terms of parameterisation) that we introduced and it seems to achieve very good performance if not the best in most datasets especially under error requirements lower than 2% and for datasets with more than 21 nodes. Therefore, it is our hypothesis that it would be the most promising first choice for testing it inside the metaheuristic that was the state of the art at the time of this study (VNS). To

Algorithm 13 Variable Neighborhood Descent

```
1: function VND (solution, Evaluator):
2: # k is the neighborhood number
3: k ← 1
4: current_solution ← solution
5: # Stop when there has been a local search in all
6: # neighborhoods without any further improvement
7: while current_solution <> solution do
8:    while k < number_of_neighborhoods do
9:       best_improv_solution ← BestImproving(solution, k)
10:      if Evaluator(solution) > Evaluator(best_improv_solution) then
11:         k ← k + 1
12:      else
13:         solution ← best_improv_solution
14:      end if
15:   end while
16:   current_solution ← solution
17:   k ← 1
18: end while
```

that end, we designed and executed experiments to assess any gains in performance. At a later section where we compare all the evaluators in metaheuristics (e.g. see Figure 2.40 and 2.41) our hypothesis was shown experimentally to be true in all the datasets tested.

Below we present a selection of results. The full results can be found at [Papapanagiotou, 2016].

2.4.2.2 Preliminary test: Speedup Estimation

In the following experiments, we try to answer if and how much better the VNS metaheuristic described in Section 2.4.2 becomes when we embed the R-M-A-M-P (VNS-RMAMP) into it, in comparison with the VNS with the original Analytical Evaluator (VNS-Analytical) used in [Campbell et al., 2011]. Due to space restrictions we show in this paper only representative results and the rest can be found in the online compendium [Papapanagiotou, 2016].

All the experiments have been run on a computer equipped with a Quad-Core AMD Opteron 2350 processor running at 2.0 GHz with 32GB of RAM. Only one core was used in each run. The parameters of R-M-A-M-P have been set as

$\alpha = 0.1$ and $\rho = 0.2$ according to some experiments available in the online compendium [Papapanagiotou, 2016]. In this section we study how much faster is the R-M-A-M-P than Analytical inside VNS. We measure the speed by the number of solution evaluations achieved in the same amount of time and then we report the speedup defined as $Speedup = \frac{\#R-M-A-M-P\_Evaluations}{\#Analytical\_Evaluations}$.

We can see the results for the datasets of 1000 and 4000 nodes in Figure 2.34 and Figure 2.35 respectively. We observe that R-M-A-M-P yields a speedup up to 544x. Similar results hold for the remaining datasets. In general, the more nodes the dataset has, the more speedup is gained. In the next sections, we will analyze how such a speedup impacts on the performance of a metaheuristic algorithm, implicitly taking into account also the precision loss of the sampling evaluator. In Section 2.3.1 it has been shown that by tuning the parameters α and ρ we can determine the trade-off of speedup and accuracy that we want.

In Section 2.2.3.4, we saw evidence that if error is lower than 2%, there is a good balance of speed and precision loss. In this thesis, the tuning is such that the precision loss is approximately 2%.

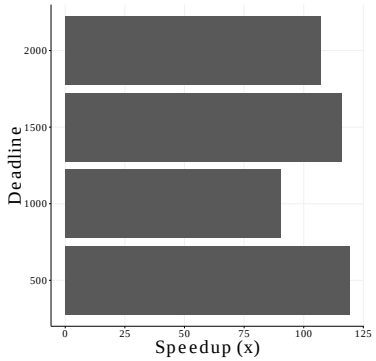


Figure 2.34. Here we see the speedup achieved by R-M-A-M-P in the 1000 dataset for a runtime of 2 minutes

2.4.2.3 Solution Quality Comparison

In the experiments we consider the solution quality reached at given runtimes as the most important performance indicator. For this purpose, the expected profit of each solution obtained is re-computed according to the Analytical Evaluator, in order to have fair comparisons. We sample the solution quality every 10 seconds from 0 minute to 2 minutes. In order to obtain representative results we run 30 times each experiment and we get the average point. Since each point is a

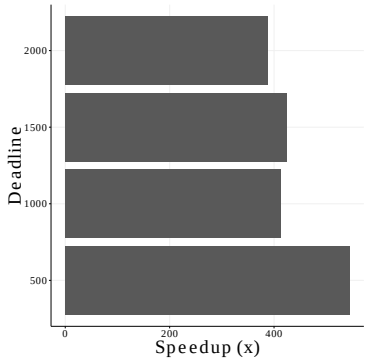


Figure 2.35. Here we see the speedup achieved by R-M-A-M-P in the 4000 dataset for a runtime of 2 minutes

sample mean and we compare sample means, we can use Student's t-test [Tanis, 2008] to compare the final values obtained by each method and see if any of the methods yields statistically significantly better results. In our experiments, we have the hypothesis that the Analytical is not worse than R-M-A-M-P and we try to find strong evidence against this hypothesis that is our *null hypothesis*. To do that we need to show that the *p-value* which is the output of a Student's t distribution is ≤ 0.05. The *p-value* is the probability that the result obtained is actually equal or more extreme than what was observed, assuming our model is true. A value lower than 0.05 is strong evidence against our null hypothesis [Tanis, 2008]. The final solutions reached by VNS-RMAMP are reevaluated by the Analytical Evaluator so that we have a fair comparison of the final values reached.

We compare the solution quality reached by the VNS metaheuristic when we have the R-M-A-M-P and Analytical evaluators embedded inside. In Figures 2.36 and 2.37 we see the objective values reached over time for the VNS metaheuristic over two different datasets, with different objective functions embedded, and with different deadlines considered. We observe that after 20 seconds (20000 ms) in every case, VNS-RMAMP reaches better values than VNS-Analytical. In Figure 2.38 we see the distribution of objective values reached after 2 minutes for the dataset with 4000 nodes. The result is from running it 30 times and each row represents a different deadline. We see that the mean value of R-M-A-M-P is better than Analytical in every case. Such a trend is true for all the datasets.

Furthermore, in Table 2.4 we can see the p-values for each deadline for the null hypothesis that VNS with R-M-A-M-P is not better than VNS with Analytical for the dataset with 1800 nodes. What we can observe is that the opposite is true

with statistical significance ($p < .05$) for all deadlines and for deadlines 1000, 1500 with great statistical significance ($p < .01$). This means that we have strong evidence to reject the null hypothesis and therefore we can say that R-M-A-M-P achieves statistically significantly better results. This is better illustrated in the boxplot in Figure 2.39 where the low and upper values of the whiskers are within ±1 Standard Error of the Mean (SEM)(see also [Tanis, 2008]). SEM shows with what precision we know the true mean of our value. Here, the true mean with approximately 70% chance is included in the whiskers. We can clearly see that the values reached by the VNS-RMAMP significantly outperform VNS-Analytical. Again, such a trend is visible for all the datasets considered, and the evidence is even more clear for larger datasets.

Additionally, in Table 2.5 and Table 2.6 we can see the objective value achieved in 2 minutes by VNS with Analytical and R-M-A-M-P and the improvement of R-M-A-M-P in percentage. Even for a low runtime of two minutes, VNS with R-M-A-M-P manages to find a better solution for every deadline from 2% to 128% better.

In Table 2.7 we see in even more detail the results of the methods by deadline. The statistics that we use in this table is the median and the Interquartile Range (IQR) (see also [Tanis, 2008]). If we sort the values (here of the objective values), the median is the value in the middle of the sorted values. If we then split the sorted values in four equal parts, we call these parts the quartiles. The difference between the third and the first quartiles is called the Interquartile Range (IQR). It is used to show how close are the values from the median and to find outliers that are thought to be points 1.5 IQR away from quartile 1 or 3. We can see the median and minimum values achieved by VNS with Analytical and R-M-A-M-P and also the IQR which is the difference between the 3rd and 1st quartiles. We can observe that the median values achieved by VNS-RMAMP are always better than VNS-Analytical and also that the minimum values achieved are in all but one case better. VNS-RMAMP in many cases has larger spread, shown by the IQR columns.

2.5 Comparison of metaheuristics

Previously, we have presented ways to tune the evaluators in isolation so that they have a small average error. In most datasets R-M-A-M-P was able to perform the best under the error constraint, while in one dataset where the deadline was not reached often, R-A-P performed the best (see Section 2.3).

In the experiments of this Section, we try to answer several questions so that

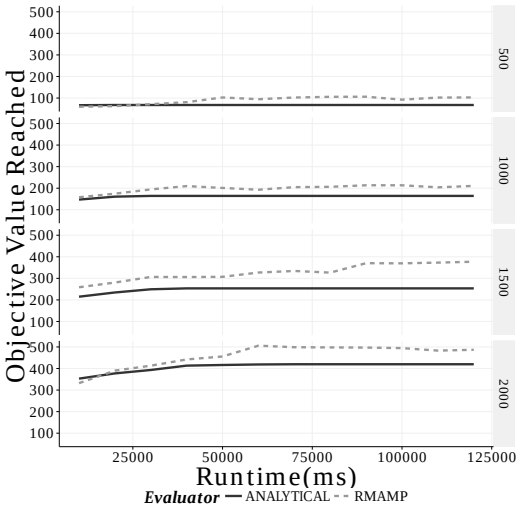


Figure 2.36. Objective value over time till 2 minutes for the dataset of 1800 nodes

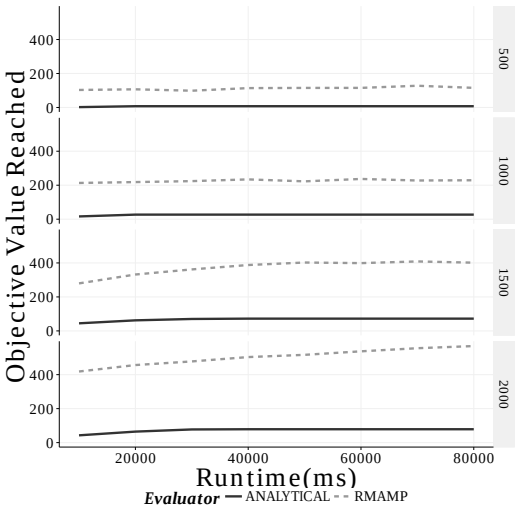


Figure 2.37. Objective value over time till 2 minutes for the dataset of 4000 nodes

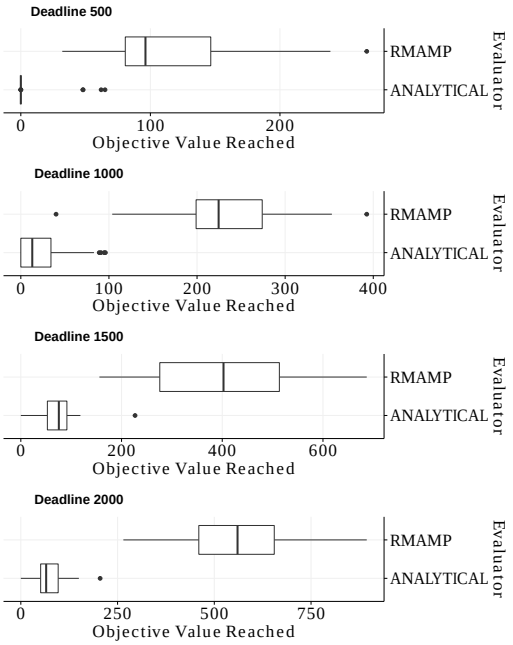


Figure 2.38. Comparison of solution quality for the dataset of 4000 nodes

we investigate further the usefulness of our proposed methods. The main questions we try to answer are about the evaluators' performance and how different evaluators with different configurations (area ratio and reward-penalty ratio) affect the final metaheuristics solution found as time progresses. To answer our questions with a limited number of experiments we conducted full factorial experiments with DA ratio taking values of 0.1, 0.8, Reward-Penalty (RP) ratio values 0.2, 0.9 (where applicable) evaluator being one of R-M-A-M-P, Analytical, MC, M-A-M, RAP, metaheuristics being one of RS or VNS. We execute the metaheuristic for 10 minutes and sample the value found every 10 seconds. We run our methods for different datasets with 1000, 1800, 2500, 4000 customers with deadlines 500, 1000, 1500 and 2000. In each sampling we record the instance, metaheuristic, evaluator, deadline, area ratio, RP ratio, runtime, number of evaluations, final solution found as evaluated by the respective evaluator as well as evaluated by the Analytical Evaluator. We also record information specific to each metaheuristic such as which heuristics ran and which one found better solutions. The results of the RS can be reproduced exactly by providing the same seed for generating the random solutions. From this information we are able to analyze several characteristics of our methods and answer multiple questions. All experiments have been carried out on a Quad-Core AMD Opteron 2350 processor running at 2.0 GHz with 32GB of RAM. Only one core was used

	Deadline	p value
1	500	0.0142*
2	1000	0.0072**
3	1500	0.0001**
4	2000	0.0286*

Table 2.4. Null hypothesis: VNS with R-M-A-M-P is not better than VNS with Analytical for the dataset of 1800 nodes. * means the null hypothesis is not accepted with $p < .05$ and ** $p < .01$

	Deadline	Analytical	R-M-A-M-P	Improvement(%)
1	500	111.96	113.86	1.70
2	1000	193.63	246.59	27.35
3	1500	289.68	442.29	52.68
4	2000	395.30	590.14	49.29

Table 2.5. Final average objective values achieved by the VNS with Analytical and R-M-A-M-P evaluators and the improvement of R-M-A-M-P for a dataset of 1000 nodes

in each run.

2.5.1 Evaluators Tuning

The performance of the evaluators in the metaheuristics depends on the tuning of their parameters. The effects of the parameters when the evaluators are tested in isolation have been studied before in Section 2.3.1. In this section we use a slightly different method to tune the parameters. In order to see the effects of Area Ratio and Reward-Penalty area ratio (RP ratio) and select the values for our application we embed the evaluators in the metaheuristic and we run it for 30 times for each evaluator, deadline and each Area ratio and RP ratio where it makes sense. The full results for this can be seen in the online addendum [Papapanagiotou, 2016]. We use these values in our experiments to determine the best tuning for each deadline. The final tuning that is used in the subsequent sections can be seen in Table 2.9.

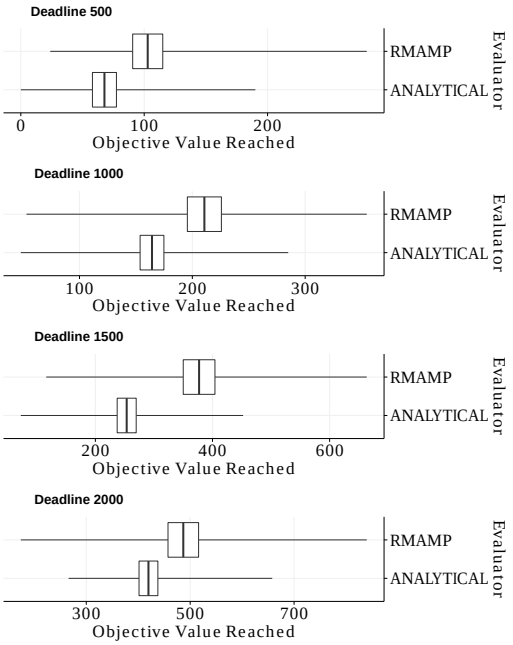


Figure 2.39. Boxplot for the dataset of 1800 nodes with data in the box being within ±1 of the standard error

2.5.2 Evaluators Speed

One of the important factors for finding good solutions is to be able to examine more solutions in a fixed runtime. Notice that in this section we deliberately omit approximation quality and we only concentrate on pure speed. A complementary analysis, focusing on approximation quality will be presented in Section 2.5.3.

In Table 2.8 we can see pairwise t-tests for the number of evaluations for runtimes of 10, 20, 30, 40 seconds for all evaluators for the RS and VNS metaheuristic for 1000 and 2500 nodes. The values in bold are $p < 0.05$ and it means that the method in the row is statistically significantly faster (achieves more evaluations at the same time).

We can observe that in all cases the proposed evaluators are faster than the conventional Analytical Evaluator and in most of them they are also **significantly** faster. Also, R-M-A-M-P is very competitive in speed, being significantly faster than the others in most cases, being outperformed (in terms of pure speed) only to R-A-P in some cases. However, speed is not the definitive factor, since accuracy is also very important. As we will see, R-M-A-M-P tends to outperform the other evaluators in most cases, despite not being always the fastest one, due to its low error.

	Deadline	Analytical	R-M-A-M-P	Improvement(%)
1	500	27.87	63.63	128.29
2	1000	55.56	92.92	67.23
3	1500	119.93	157.05	30.95
4	2000	149.53	154.79	3.52

Table 2.6. Final average objective values achieved by VNS with Analytical and R-M-A-M-P evaluators and the improvement of R-M-A-M-P for a dataset of 2500 nodes

Deadline	Evaluator	Value range		
		Median	Min	IQR
500	Analytical	67.7280	0.0000	70.5716
	R-M-A-M-P	79.5000	23.8352	49.5775
1000	Analytical	169.0695	48.0000	78.7230
	R-M-A-M-P	210.9990	52.9999	126.6765
1500	Analytical	258.8520	72.7217	90.7275
	R-M-A-M-P	374.4665	116.1500	192.6325
2000	Analytical	409.9925	266.0090	132.9370
	R-M-A-M-P	483.7065	174.0000	237.7930

Table 2.7. Value ranges for different deadlines for the dataset of 1800 nodes

In Figure 2.40 and Figure 2.41 we can see the evolution of the objective value reached by the VNS metaheuristic by the datasets of 1000 and 1800 nodes over the course of 10 minutes with deadline 2000. We can observe that the solution quality when R-M-A-M-P is in use, is consistently better than that of any other evaluator. The second better evaluator is M-A-M and the third better R-A-P. We also observe that R-M-A-M-P, M-A-M and R-A-P continue to improve while Analytical and MC soon reach a plateau (stagnation).

When VNS uses R-M-A-M-P, R-A-P and M-A-M, the quality of the best solutions is not monotonically increasing. This is due to approximation errors: in the charts we report the objective value of each solution evaluated with the Analytical Evaluator. Therefore, when an objective value appears to get worse then a previous one, this indicates that the respective evaluator made an approximation error of such a solution and (erroneously) accepted it as a better solution. It is interesting to notice how such a phenomenon seems to improve the performance of the VNS, introducing a behaviour apparently useful to exit local minima.

For further results one can consult the online addendum [Papapanagiotou,

2016]. The results are however aligned with those presented.

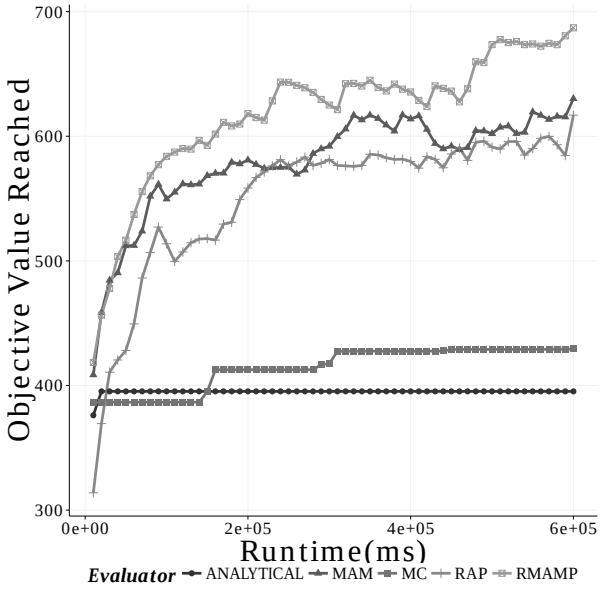


Figure 2.40. Objective Value vs Runtime, 1000 nodes, Deadline: 2000, VNS metaheuristic

In Table 2.10 we can see a pairwise one-sided t-test (the p-values) for the final objective values for deadline 2000, reached by VNS for the datasets of 1000 and 1800 nodes when using the respective evaluators. We consider something statistically significantly better if $p < 0.05$. The values in bold mean that if we incorporate in VNS the evaluator in the row we will get a significantly better objective value than if we incorporate the evaluator in the column. In the 1000 dataset all proposed evaluators reach significantly better values than Analytical and R-M-A-M-P is significantly better than all the other evaluators. In the same dataset M-A-M is significantly better than R-A-P and R-A-P is significantly better than MC. In the 1800 dataset M-A-M and R-M-A-M-P are significantly better than Analytical, all evaluators are significantly better than MC and R-M-A-M-P is significantly better than all the evaluators except M-A-M. Similar patterns are seen for all other deadlines and datasets in the online addendum [Papapanagiotou, 2016]. The proposed evaluators have greater impact in non-trivial metaheuristics such as VNS because their characteristics and most notably speed and accuracy drive the metaheuristic in different neighbourhoods.

To conclude, it appears that the proposed objective function evaluators have great impact on non-trivial metaheuristics such as the VNS we consider, since

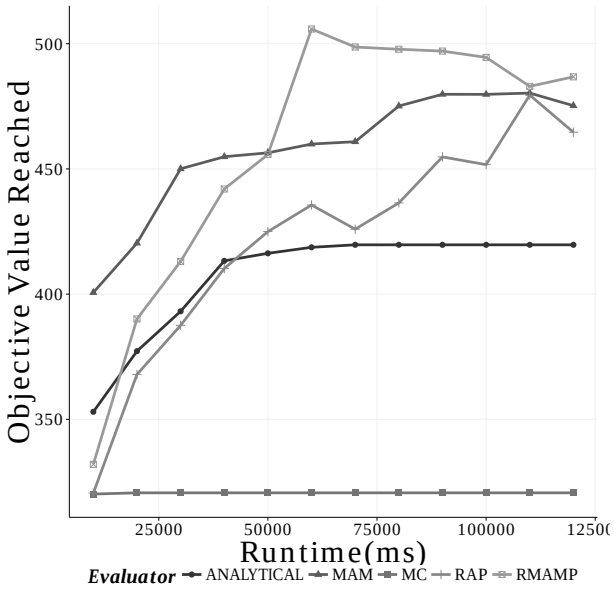


Figure 2.41. Objective Value vs Runtime, 1800 nodes, Deadline: 2000, VNS metaheuristic

their characteristics and most notably speed and accuracy are useful to drive the metaheuristic in different neighbourhoods, leading to an overall better exploration of the search space.

2.5.3 VNS as compared to RS

We now compare the 2 metaheuristics by comparing the final objective values reached when they are coupled with R-M-A-M-P. A tuned R-M-A-M-P is the most adaptable and in the vast majority of the cases the best performing evaluator as shown in [Papapanagiotou et al., 2015a] and also in our current results (see also the online addendum [Papapanagiotou, 2016]). In Table 2.11 we can see for each instance and deadline the best value obtained by RS and VNS using R-M-A-M-P with the same tuning (see Table 2.9). In bold we can see the maximum of the 2 metaheuristics for each instance and deadline. We observe that RS obtains better solutions in every single case. For completeness we also did hypothesis testing (Student's t-tests) to see if RS is statistically significantly better than VNS.

Since each objective value reached is the average of 30 runs, we can use one-sided paired t-tests to compare if different combinations of "Metaheuristic-Evaluator" are worse than others (they reach statistically signifant worse re-

sults). If the p value that the t-tests return is $p < .05$ we consider that the first "Metaheuristic-Evaluator" of the comparison is significantly worse than the second one. We examined all combinations of metaheuristics and evaluators and we found that, no matter the evaluator used, VNS reaches statistically significantly worse objective values. In fact, the p values reached are practically zero. The results are in the online addendum [Papapanagiotou, 2016].

2.6 Discussion and Conclusions

In this chapter we have shown ways to improve the state-of-the-art for obtaining solutions to the OPSTS. The first approach was to improve the bottleneck of the optimization of OPSTS which is the objective function. That was achieved by developing ways to approximate it with Monte Carlo sampling. We examined how the number of samples affect the error and performance and showed how performance scales as the total runtime increases. Because of the nature of the problem, the Monte Carlo Evaluator (MC) in certain cases examined yields a significant error.

To alleviate that problem we developed hybrid evaluators that combine evaluators with different accuracy for different parts of the solution in order to speedup the evaluation with minimal error. We introduced the concepts of Deadline Area and reward-penalty area representing the parts of the solution with highest and lowest error respectively. According to our definitions, we developed three new evaluators namely M-A-M, R-M-A-M-P, R-A-P.

We also presented results comparing and contrasting all the different evaluators in different contexts and discussing their usefulness. We saw that they can have dramatic effects in the reduction of the error and reduce the need of the number of samples to achieve the same levels of accuracy and thus making it possible to increase the speedup we can yield. To achieve that, we presented ways to tune them for best results in optimizers and more specifically inside metaheuristics. In this work, We presented two ways to tune the evaluators before running the optimization procedure. A further interesting development would be to auto-tune the parameters during the running time of the metaheuristics.

Initially, we tested them inside the metaheuristic (VNS) that was state-of-the-art and we saw that all of the evaluators had significant gains in speedup compared to the one that was state-of-the-art. The small approximation errors that these evaluators intrinsically introduce, because of the large speedup of the optimizer, become negligible.

We also generated big datasets with thousands of nodes in order to show the

utility of our approaches for much larger datasets than the ones used before. The generation methodology used was similar to the one applied in the existing datasets in the literature. We then proceeded in conducting experiments using the generated datasets to demonstrate the usefulness of our approach when we have more nodes.

We also developed a Random Search Metaheuristic (RS) and compared it to the state-of-the-art Variable Neighborhood Search Metaheuristic. We argued that RS is more suitable for comparing the evaluators in an optimizer context because the evaluators can be tested using the same solution evaluation set and then designed and made extensive experiments. Finally, we showed that in these large datasets RS performed significantly better than VNS.

RS - 1000 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	9.04E-01			
MC	**2.50E-04**	**2.18E-06**		
RAP	1.73E-01	**1.33E-02**	9.96E-01	
R-M-A-M-P	**1.87E-06**	**8.89E-09**	8.99E-02	**6.25E-05**

VNS - 1000 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	**3.25E-62**			
MC	**5.79E-03**	1.00E+00		
RAP	**1.26E-128**	**6.28E-22**	**2.34E-110**	
R-M-A-M-P	**0.00E+00**	**0.00E+00**	**0.00E+00**	**3.03E-264**

RS - 2500 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	1.00E+00			
MC	9.93E-01	1.49E-01		
RAP	**2.70E-02**	**1.71E-07**	**1.23E-05**	
R-M-A-M-P	9.96E-01	2.08E-01	5.90E-01	1.00E+00

VNS - 2500 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	**6.64E-05**			
MC	3.31E-01	1.00E+00		
RAP	**2.00E-176**	**1.06E-171**	**6.63E-176**	
R-M-A-M-P	**4.77E-62**	**1.06E-49**	**1.07E-60**	1.00E+00

Table 2.8. Pairwise one-tailed t-tests p-values for all deadlines. The highlighted ones mean that the method in the row achieves statistically significantly more evaluations than the one in the column

Instance	Deadline	Area Ratio	RP Ratio
1000	500	0.8	0.9
1000	1000	0.1	0.2
1000	1500	0.1	0.2
1000	2000	0.1	0.2
1800	500	0.8	0.9
1800	1000	0.1	0.2
1800	1500	0.1	0.2
1800	2000	0.1	0.2
2500	500	0.8	0.9
2500	1000	0.1	0.2
2500	1500	0.1	0.2
2500	2000	0.1	0.2
4000	500	0.8	0.9
4000	1000	0.1	0.2
4000	1500	0.1	0.2
4000	2000	0.1	0.2

Table 2.9. Final Tuning Table

Table 2.10. Pairwise one-tailed t-test for final objective value reached when using VNS with each evaluator for deadline 2000. Values $p < 0.05$ mean that when we use the evaluator in the row in VNS for 1000 and 1800 nodes, a statistically significantly better objective value is reached

VNS - 1000 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	**6.03E-72**			
MC	**6.03E-72**	1.00E+00		
RAP	**1.97E-58**	1.00E+00	**4.65E-49**	
R-M-A-M-P	**3.51E-89**	**3.57E-07**	**9.00E-81**	**5.61E-17**
VNS - 1800 nodes				
	Analytical	M-A-M	MC	RAP
M-A-M	**4.14E-04**			
MC	1.00E+00	1.00E+00		
RAP	1.62E-01	9.93E-01	**1.26E-09**	
R-M-A-M-P	**3.91E-04**	4.93E-01	**9.25E-14**	**6.62E-03**

Instance	Deadline	RS	VNS
1000	500	**360.35**	155.98
1000	1000	**580.00**	289.62
1000	1500	**724.00**	484.78
1000	2000	**766.39**	687.04
1800	500	**314.00**	106.16
1800	1000	**474.04**	213.60
1800	1500	**662.07**	377.12
1800	2000	**707.46**	505.85
2500	500	**226.35**	63.63
2500	1000	**227.00**	92.92
2500	1500	**355.00**	157.05
2500	2000	**378.00**	180.42
4000	500	**191.00**	9.27
4000	1000	**225.40**	38.11
4000	1500	**246.71**	63.04
4000	2000	**250.00**	85.29

Table 2.11. Results for all instances and metaheuristics using R-M-A-M-P. In bold we see the average values found by R-M-A-M-P for each deadline of each instance (after running the metaheuristics 30 times). Maximum runtime: 10 minutes

Chapter 3

Solution methods for the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands

In this chapter, we are going to discuss our solution approaches for the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands, introduced in Section 1.2.3. Our contribution consists of both defining this new problem and proposing different approaches. The results mentioned here are based on our published work [Toklu et al., 2013], [Toklu et al., 2014] and [Klumpp et al., 2014]. First we will discuss the main metaheuristic used for solving the problem, which is the Ant Colony Metaheuristic and then we will proceed with examining our proposed objective functions. Subsequently we will see the experiment design and results and in section after that an environmental application of this problem. The final section concludes our findings.

3.1 The Ant Colony Metaheuristic

The Ant Colony Optimization algorithm (ACO) first appeared in [Dorigo, 1992] and [Dorigo et al., 1991]. It is a metaheuristic for finding solutions to difficult combinatorial optimization problems which can be reduced to finding near optimal paths in graphs like the Vehicle Routing Problem (VRP) or the traveling salesman problem (TSP).

In ACO we simulate a number of artificial ants that "walk" on the graph $G = (L, A)$ and each one finds a solution. The ants begin by "walking" randomly on the graph. When they have found a solution artificial pheromones trails are used on the arcs they have walked. The quantity of artificial pheromones on

the arcs of a solution, depends on the quality of the generated solution. Over time, pheromone trails start to "evaporate". In a shorter path there is less evaporation because it is marched over more frequently and the amount of artificial pheromone on it becomes higher. Pheromone evaporation helps avoiding early convergence to local optima and if it did not exist, the paths chosen by the first ants would be excessively attractive.

Subsequent ants trying to find new solutions when they encounter a pheromone trail, they are tempted to follow the arcs with higher amounts of pheromone left by previous ants. However, there is a smaller probability that the ant may follow other arcs on the graph too. The procedure is repeated iteratively and because the pheromone amount increases in the arcs belonging to good solutions, we can observe that the ants in the end converge to a near-optimal solution.

The ant colony system used is an *elitist* algorithm. In our problem, elitism means that only ants improving the currently known best solution put new pheromones on the graph. Elitism also implies that ants mainly apply local searches around the best solutions.

The implementation of the ant colony system described in this paper is based on [Gambardella et al., 1999]. Firstly, a solution is generated using the nearest neighbour heuristic (NNH) (for more details see [Johnson and McGeoch, 1997]). The best solution of NNH is then saved in a variable called *best*. Afterwards, a new ant colony is generated with Ω ants. The cost of the new solution is evaluated by the objective function. If $best$ is smaller than our current solution, then the current solution is saved in the variable $best$. Then, the procedure of generating new ant colonies and constructing new solutions is repeated until a finishing criterion is met.

For each vehicle, an artificial ant constructs a solution by picking locations one by one. At first, always the depot (the location 0) is picked for a vehicle. After that, other locations are added one by one onto the route, among the visitable locations. A location is considered visitable, if that location is not visited yet in the current location, and, adding that location does not violate the capacity constraint of the vehicle which is currently being considered. Now, considering an ant at the location i (i.e. an ant which added the location i in its most recent step), for picking the next location j, let us define the following: N_w is the set of visitable locations for the ant w; Δ_{ij} is the euclidean distance between the locations i and j; $\eta_{ij} = 1/\Delta_{ij}$ is a heuristic indicator of proximity between the locations i and j; α is a parameter which configures the balance of importance between exploitation and exploration; and finally β is a parameter which configures the importance of proximity during the process of picking the next visitable

location. Further details about exploitation and exploration are as follows: with probability α the ant colony decides to do exploitation and follows the arc (i, j) which maximizes $\tau_{ij} \cdot [\eta_{ij}]^{\beta}$, and with probability $1 - \alpha$ decides to explore. In the case of exploration, the next location j is picked with probability:

$$p_{ij}^{w} = \begin{cases} \dfrac{\tau_{ij} \cdot [\eta_{ij}]^{\beta}}{\sum_{j' \in N_w} \tau_{ij'} \cdot [\eta_{ij'}]^{\beta}} & \text{if } j \in N_w \\ 0 & \text{otherwise} \end{cases}$$

For more information about the ant colony system algorithm, the reader is referred to [Gambardella et al., 1999].

3.2 Solution Approaches

In the following section we present the actual objective functions we proposed in order to approximate the objective function shown in Equation (1.10). For better reference we repeat here the notation introduced in Section 1.2.3.2. V is the set of vehicles available and c_{ij} is the cost of traveling from location i to location j, x^v is the route decided for vehicle $v \in V$, $|x^v|$ the length of the route x^v, and x_k^v is the k-th visited place of the vehicle v in x. S is the set of scenarios considered, and they are sampled using Monte Carlo sampling, and d_i^s is the demand in location i in scenario $s \in S$. $\underline{d}_i$ is the "base demand", the one that is known before visitation of location i and $d_i^{increase}$ is a random variable following a half normal distribution representing demand increases in each approach.

3.2.1 1-Stage Best-Case approach

This approach treats our problem as a 1-stage regular CVRP. All the demands are assumed to be fixed and to be equal to their original lowest possible values by this approach. The objective function is, like in the regular CVRP, the total travel cost.

$$\min\left[\sum_{v \in V} \sum_{k=1}^{|x^v|-1} (c_{ij} | i = x_k^v, j = x_{k+1}^v)\right] \tag{3.1}$$

Demands: $d_i = \underline{d}_i$.

3.2.2 1-Stage Average-Case approach

Like the 1-Stage Best-Case approach, this approach treats our problem as a 1-stage regular CVRP, except that all the demands are assumed to be equal to the expected values of their related probability distributions. The objective function is, like in the regular CVRP and in the previous approach, the total travel cost.

$$\min\left[\sum_{v\in V}\sum_{k=1}^{|x^v|-1}(c_{ij}|i=x_k^v, j=x_{k+1}^v)\right] \tag{3.2}$$

Demands: $d_i = \underline{d}_i + E[d_i^{increase}]$.

3.2.3 1-Stage Worst-Case approach

Again, this approach treats our problem as a 1-stage regular CVRP. All the demands are assumed to be equal to their original lowest possible values plus two times the standard deviations of their related probability distributions. Like in the previous approaches, the objective function is the total travel cost.

$$\min\left[\sum_{v\in V}\sum_{k=1}^{|x^v|-1}(c_{ij}|i=x_k^v, j=x_{k+1}^v)\right] \tag{3.3}$$

Demands: $d_i = \underline{d}_i + 2\sigma_{d_i^{increase}}$.

3.2.4 2-Stage Average-Case

In this approach, the artificial ants of the algorithm generates the solutions by considering the original demands. Later, in the objective function, a solution is tested against the average scenario z, in which each demand increase is assumed to be equal to the expected value of its related probability distribution. This corresponds to treating the problem as a 2-stage problem: first stage representing the original scenario, and the second stage representing the scenario revealed later with increased demands. For testing a solution x against the average scenario z, the accurate objective function would be:

$$\sum_{v\in V}\sum_{k=1}^{|x^v|-1} c_{x_{k-1}^v, x_k^v} + F_z(x) \tag{3.4}$$

that is, the cost of the solution in the original base scenario plus the extra costs for satisfying the customers who are unsatisfied in the average scenario z. The

problem with using the function $F_z(x)$, however, is that it solves the sub-VRP problem for satisfying the unsatisfied customers by using an exact method, which demands too much execution time and therefore becomes unsuitable for being embedded into the objective function of a metaheuristic. Therefore, we define another function, $F'_z(x)$, which is the same with $F_z(x)$, except that it uses NNH, instead of a slower exact method, for solving the sub-VRP problem. The Nearest Neighbor Heuristic is good enough for small sized problems (see [Johnson and McGeoch, 1997]) as the ones that $F'_z(X)$ has to solve. Therefore, the objective function of this approach becomes:

$$\sum_{v \in V} \sum_{k=1}^{|x^v|-1} c_{x^v_{k-1}, x^v_k} + F'_z(x) \tag{3.5}$$

3.3 Experimental Results

In this section, we present the results generated by our approaches: 2-Stage Average-Case approach, 1-Stage Average-Case approach, 1-Stage Best-Case approach, and 1-Stage Worst-Case approach. Each approach was implemented in C, and the results were executed on an Intel Core 2 Duo P9600 @ 2.66GHz computer with 4GBs of RAM, running Ubuntu 12.04.

The instances that we used for the experiments are tai100 {a,b,c,d} and tai150 {a,b,c,d}, which have 100 and 150 customers, respectively. These instances from the literature fit best the requirements of the company that motivated this research. Due to the scope of this study which is mainly economics related the provided 8 instances are sufficient and a good fit to provide evidence for decision making. These instances were originally designed for deterministic CVRP. To adapt these instances into our problem, we applied the following procedure: for each arc (i, j), the random cost variable is made to behave according to the half-gaussian distribution for which the lower bound is c'_{ij}, and the standard deviation is $c'_{ij} \cdot (1 + RND(0, 0.2))$, where c'_{ij} is the deterministic cost of the arc (i, j) in the original instance, and $RND(a, b)$ is a random real number between a and b.

Let us now see how all these approaches compare to each other. For this purpose, we compare the costs and vehicle requirements of the solutions generated by each approach on each instance. On each instance, each approach was run 9 times. The reported costs and vehicle requirements are averaged over these 9 runs. Note that, as explained in Section 3.1, each ant colony optimization approach has its own objective function for evaluating the quality of a solution during the optimization. However, after the algorithms produce their solutions,

we evaluate these solutions, according to the evaluation function (1.10) mentioned in the problem definition in Section 1.2.3.2, as the minimization of that function is the true objective of this study. While using the evaluation function (1.10), we set the number of scenarios as 1000 (i.e. $|S| = 1000$). The results are presented in Table 3.1. Under the column groups "Cost" and "Number of vehicles", the mean values and the standard deviations (labeled as "StDev") over the 9 runs are given. The general comments that we can derive from the table are as follows:

— The 1-Stage Best-Case approach generates the most expensive results. An explanation for this is that it does not anticipate the extra costs at all by focusing only on base demands, therefore, it can not foresee which solutions are really cheap. Another interesting thing to observe is that this approach generates solutions with very low vehicle numbers. This comes at a price though: as it focuses only on the best-case scenario, in scenarios where there are significant increases in the demands, the guarantee of visiting a customer only once dissappears easily, as there will be many unsatisfied customers to be revisited. This results in a decrease in the quality of service.

— The 1-Stage Worst-Case approach generates the results which require the highest number of vehicles. This is caused by the over-conservative nature of the approach: to make sure that no customer is left unsatisfied, it sends too much vehicles.

— The 1-Stage Average-Case approach generates the cheapest results. As it focuses on the expected values of the random variables, it can foresee the actual costs of the solutions more successfully than the others.

— The 2-Stage Average-Case approach generates the solutions similar to the ones generated by the 1-Stage Best-Case approach, in the sense that it produces solutions with very low vehicle requirements. However, thanks to its second stage, it can foresee more successfully, which solutions are actually cheaper. Therefore, its solutions are cheaper than the solutions of the 1-Stage Best-Case approach. To sum up, a decision maker who wants stronger guarantees of visiting a customer once and/or wants cheaper solutions could go for the 1-Stage Best-Case approach. On the other hand, if it is more important to minimize the number of vehicles used, 2-Stage Average-Case approach could be used.

In the objective function (3.5) of the 2-Stage Average-Case approach, on the average scenario z, $F'_z(x)$ is used instead of $F_z(x)$ for the sake of having a fast solution evaluation procedure. In more details, an exact integer linear programming formulation could take even minutes on the evaluation of a solution on a single scenario, rendering a metaheuristic depending on it very slow and useless. Because of this, the 2-Stage Average-Case approach depends on NNH, instead of

Instance	Approach	Cost		Number of vehicles	
		Mean	StDev	Mean	StDev
tai100a.dat	2-Stage Average-Case	2473.29	101.14	12.02	0.01
	1-Stage Average-Case	2268.23	29.07	13.45	0.04
	1-Stage Best-Case	2558.7	50.77	12.02	0.01
	1-Stage Worst-Case	2371.26	39.56	14.04	0.01
tai100b.dat	2-Stage Average-Case	2286.9	55.72	12.0	0.0
	1-Stage Average-Case	2182.79	38.61	12.46	0.06
	1-Stage Best-Case	2364.22	34.31	12.0	0.0
	1-Stage Worst-Case	2175.27	33.83	13.03	0.02
tai100c.dat	2-Stage Average-Case	1575.71	47.88	12.0	0.0
	1-Stage Average-Case	1545.82	36.15	12.46	0.05
	1-Stage Best-Case	1732.28	54.98	12.0	0.0
	1-Stage Worst-Case	1618.44	20.67	13.03	0.01
tai100d.dat	2-Stage Average-Case	1796.25	24.12	12.0	0.0
	1-Stage Average-Case	1771.16	37.91	12.5	0.03
	1-Stage Best-Case	1916.58	75.24	12.0	0.0
	1-Stage Worst-Case	1748.89	20.96	13.02	0.01
tai150a.dat	2-Stage Average-Case	3670.26	56.07	16.17	0.08
	1-Stage Average-Case	3612.9	47.28	17.61	0.05
	1-Stage Best-Case	3709.71	41.18	16.23	0.04
	1-Stage Worst-Case	3697.63	58.51	18.08	0.02
tai150b.dat	2-Stage Average-Case	3437.09	88.02	15.11	0.08
	1-Stage Average-Case	3229.12	89.38	16.62	0.02
	1-Stage Best-Case	3455.96	67.26	15.16	0.09
	1-Stage Worst-Case	3357.25	107.76	17.07	0.01
tai150c.dat	2-Stage Average-Case	3076.15	138.64	15.68	0.01
	1-Stage Average-Case	2926.06	99.51	16.72	0.03
	1-Stage Best-Case	3150.65	60.71	15.7	0.02
	1-Stage Worst-Case	2946.59	94.82	17.29	0.65
tai150d.dat	2-Stage Average-Case	3170.31	56.22	15.12	0.04
	1-Stage Average-Case	3063.32	95.69	16.62	0.07
	1-Stage Best-Case	3271.07	85.92	15.12	0.05
	1-Stage Worst-Case	3061.49	32.42	17.08	0.02

Table 3.1. Comparison of the results of the different ant colony approaches

an exact method, within its objective function. However, because $F'_z(x)$ uses NNH to solve a sub-VRP on the graph of unsatisfied customers at the end of the first stage, it is not able to solve the sub-VRP to optimality, so, it overestimates the extra cost of the second stage. Let us now compare the functions (3.4) (which uses exact method) and (3.5) (which is based on NNH and is used within the 2-Stage Average-Case approach) to see if the overestimated costs of the 2-Stage Average-Case approach are still realistic. For this purpose, we evaluate the final results provided by the 2-Stage approach first by using (3.4) and then by using (3.5). The results are presented in Table 3.2. In the table, the mean values and the standard deviations ("StDev") of the costs and the vehicle requirements are given, averaged over 9 runs of the 2-Stage Average-Case approach. The results of both functions (3.4) and (3.5) are given, and also, in the rows labeled "Increase (%)", the percentage of the overestimation made by the function (3.5) are reported. In the table, it can be seen that the overestimation percentages in the costs are tolerable (the highest being 1.71%). Also, there does not seem to be any change in the number of vehicle requirements.

However, it should also be noted that the standard deviation of the overestimation of function (3.5) over (3.4) ("Increase (%)") is high. This shows that, in general, the NNH method is not consistent in performance over different runs. In turns, the performance of 2-Stage Average-Case, which uses NNH extensively, is likely to be affected by this.

3.4 Environmental Application

In the paper [Klumpp et al., 2014], the authors examine an environmental application of this problem, related to the "Green Bullwhip Effect".

The bullwhip effect in a distribution channel refers to the fact that shifts in customer demand propagate in increasing swings in inventory as one moves further up the supply chain. It is also known as the "Forrester effect" and has been described in research since more than fifty years [Forrester, 1961], [Lee et al., 1997]. Since the first documentation of this effect, a lot of research has been conducted in transport scheduling and lot sizing in order to mitigate the problem and support the decision making of logistics managers in different supply chains [Agrawal et al., 2009], [Chatfield et al., 2004], [Crisan and M., 2012], [Wright and Yuan, 2008].

Since the field of logistics is responsible of about 5.5% of global climate gas emissions, several efforts have been made to reduce the environmental impact [Aronsson et al., 2008], [Murphy and Poist, 2000], [Charter and Tischner, 2001],

Instance	Function used for evaluation	Cost		Number of vehicles	
		Mean	StDev	Mean	StDev
tai100a.dat	(3.4)	2622.61	142.39	12.0	0.0
	(3.5)	2626.89	145.32	12.0	0.0
	Increase (%)	0.16	2.06	0.0	0.0
tai100b.dat	(3.4)	2383.2	127.3	12.0	0.0
	(3.5)	2390.62	129.84	12.0	0.0
	Increase (%)	0.31	2.0	0.0	0.0
tai100c.dat	(3.4)	1712.17	190.99	12.0	0.0
	(3.5)	1728.42	197.15	12.0	0.0
	Increase (%)	0.95	3.23	0.0	0.0
tai100d.dat	(3.4)	1939.75	126.93	12.0	0.0
	(3.5)	1947.05	129.01	12.0	0.0
	Increase (%)	0.38	1.64	0.0	0.0
tai150a.dat	(3.4)	3922.22	256.61	16.0	0.0
	(3.5)	3930.32	260.71	16.0	0.0
	Increase (%)	0.21	1.60	0.0	0.0
tai150b.dat	(3.4)	3750.23	406.86	15.0	0.0
	(3.5)	3774.76	417.04	15.0	0.0
	Increase (%)	0.65	2.50	0.0	0.0
tai150c.dat	(3.4)	3530.76	402.06	17.11	0.78
	(3.5)	3591.22	423.63	17.11	0.78
	Increase (%)	1.71	5.36	0.0	0.0
tai150d.dat	(3.4)	3485.24	344.57	15.22	0.67
	(3.5)	3517.29	355.35	15.22	0.67
	Increase (%)	0.92	3.13	0.0	0.0

Table 3.2. The results evaluated by the objective function of the 2-Stage Average Case approach, by using (3.4) and (3.5)

[Sundarakani et al., 2010]. The "bullwhip effect" in turn tends to create increased safety stock levels in order to reduce the anticipated flexibility cost increases in case of demand uncertainty and have an environmental impact. The hypothesis that the bullwhip effect has important consequences on the environment has been suggested in [Klumpp, 2011] and it has been named "The Green Bullwhip Effect". By using the results of this problem's chapter which is based on the assumptions of real companies in Germany, one can decide how to schedule the deliveries (e.g. expected shorter distances or less vehicles) in order to minimize the environmental impact.

3.5 Discussion and Conclusions

In this chapter we examined a 2-Stage vehicle routing problem with probabilistic demand increases. In the first stage, the vehicles start from the depot and try to satisfy the base demands of the customers. In the second stage, extra vehicles are sent to the customers whose demands were not previously met in their entirety because of probabilistic increases. In order to solve the problem, we used our various ant colony optimization approaches. From the experiments, it can be concluded that adopting a 1-stage average case approach leads to shortest overall distances while a 2-stage approach leads to solutions with less vehicles. The conclusions reached by the methods could serve as a decision supporting system for logistics managers in order to reduce the environmental impact of gas emissions and the "green bullwhip effect".

Chapter 4

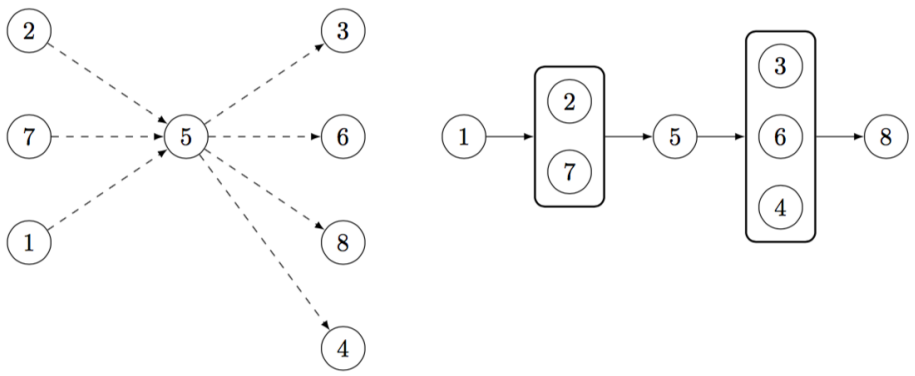
Exact Methods for the Sequential Ordering Problem

In this chapter we are going to discuss exact solution approaches for the Sequential Ordering Problem introduced in Section 1.2.4. The results mentioned here are based on our work published in [Papapanagiotou et al., 2015b] and [Jamal et al., 2017]. First we made a comparison of 2 existing methods in the literature, namely the Decomposition Based Approach (DEC) presented in [Montemanni et al., 2013], [Mojana et al., 2012] and the Branch-and-Bound Approach (B&B) presented in [Shobaki and Jamal, 2015]. The first method was used for applications in cargo and transportation problems and the second for minimising instruction power switching in compilers. The insights of the comparison led to an improved version of the Branch-and-Bound Approach. In Sections 4.1-4.4 we demonstrate and discuss the experimental comparison of DEC and $B\&B$ and in section 4.5 we discuss an enhanced B&B method to solve the Sequential Ordering Problem.

4.1 The Decomposition Based Approach (DEC)

The exact method that we call the Decomposition-Based Approach (DEC) was developed and described in [Mojana, 2011], [Montemanni et al., 2013]. DEC begins by splitting the original problem in two or more subproblems, solves them and then merges the solutions. The main idea is to reduce the complexity of solving it by exploiting substructures that are present in the precedence graph. An important observation is that the effort to find an exact solution to an instance is not dependent only on the number of nodes but also on how the precedence graph is structured. For example, a very dense or very sparse precedence graph

is usually easier to solve than one with medium complexity. Additionally, the substructures present in the precedence graph could change its complexity for finding solutions. The goal of DEC is to find k subproblems such that when they are solved and recombined, they will be a solution to the original instance. For example, the structure in Figure 4.1(a) with $k = 2$ can be decomposed to substructures as in Figure 4.1(b). We can observe that Vertex 5 is connected to all other vertices, therefore in this situation we know what will precede or follow vertex 5. We call such a vertex, a fixed vertex. In Figure 4.1(b) we can see the structure of a possible feasible solution, starting from node 1, followed by a permutation of nodes $\{2, 7\}$, followed by node 5, followed by permutation of nodes $\{3, 6, 4\}$, ending with node 8. Therefore the problem is decomposed in two parts: the first consists of the vertices $\{1, 2, 7, 5\}$ and the second consists of the rest of the vertices, namely $\{5, 3, 6, 4, 8\}$. This decomposition is exploited by a branch and bound framework. In order to branch, artificial constraints are imposed on the first new node and its opposite on the second one. The goal is to have new artificial fixed vertices in the different branches of the search-tree. The resulting substructures when they get small enough, they are solved using a mixed integer linear program described in [Montemanni et al., 2013].



(a) Precedence relation involving a fixed node 5. The dashed arrows represent a subset of the precedence relations between the nodes.

(b) Structure of each feasible solution. The two subproblems are needed to find the optimal permutation of the nodes in the two boxes.

Figure 4.1. Fixed node decomposition

4.2 The Branch-and-Bound Approach (B&B)

The second method tested is called The Branch and Bound Approach (B&B) was introduced and was first described in detail in [Shobaki and Jamal, 2015]. Using branch and bound we can enumerate systematically the solution space. The representation used is a tree which has an empty root, inner nodes that represent partial solutions and leaves that represent complete solutions. For each existing partial solution, lower bounds on the optimal solution that can be found later, are computed. If the lower bound computed is not less than the best solution already found, we stop expanding this branch (we prune the tree), since it cannot offer something better. The pruning techniques used are very important for the efficiency of the algorithm since we want to minimize the number of explicitly enumerated nodes. In brief, we can compute the lower bound on a path by observing that there is exactly one incoming and one outgoing edge in a Hamiltonian path and taking the minimum cost outgoing and incoming edges and summing them up. For example in Figure 4.2 we can see that the Minimum Outgoing Edge (MOE) is equal to 11 (7+1+3) and the Minimum Incoming Edge is 13 (5+6+2) and the result of the heuristic is $max(MIE, MOE) = 13$. History utilization consists of storing solutions to previously visited substructures and reusing them in case these substructures are encountered again. For example in Figure 4.3 subproblems 9 and 3 are similar because in both of them the nodes A,C,D have been visited but subproblem 9 has higher lower-bound. Therefore expanding it cannot find a better solution than 3 and it is pruned. Apart from these pruning techniques, another one is also used based on minimum-cost perfect matching (MCPM). This technique further tightens the edge-based lower bound. The edge-based lower bound may be too loose since it may use the same minimum-cost edges multiple times. By formulating the problem as MCPM we avoid this problem and a solution to the MCPM gives a tighter lower bound. Pruning using MCPM is defined and examined in Section 4.5.2.

4.3 Experimental comparison of DEC and B&B

Three benchmark libraries that have been used in published work are considered in this evaluation: TSPLIB [Ascheuer, 1996], SOPLIB06 [Montemanni et al., 2008] and COMPILERS [Shobaki and Jamal, 2015]. These benchmarks originate from different domains and have different characteristics, thus making the comparison of the two exact algorithms more interesting. All experiments have been carried out on a Quad-Core AMD Opteron 2350 processor running at 2.0 GHz

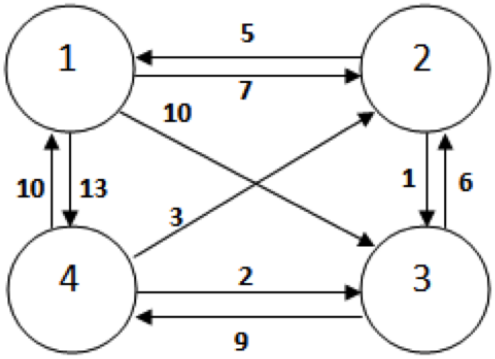


Figure 4.2. Edge-based heuristic: Summing the lowest $n-1$ edge weights. Minimum Outgoing Edge: $7+1+3=11$. Minimum Incoming Edge: $5+6+2=13$. $Result = max(11,13) = 13$

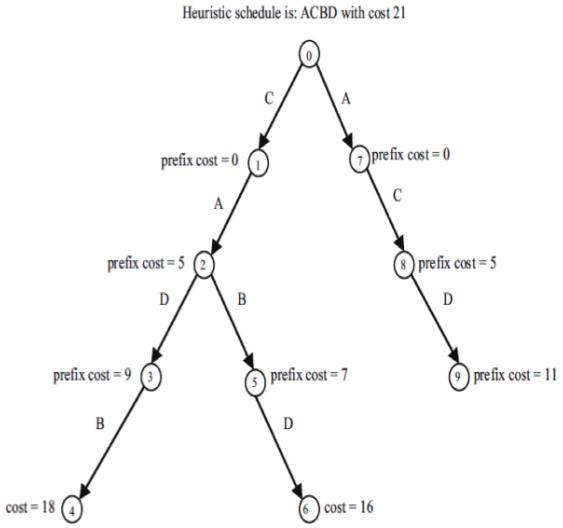


Figure 4.3. History Utilization. Sub-problems 9 and 3 are similar but since subproblem 9 has higher lower-bound it is pruned

with 32GB of RAM. Only one core was used in each run. A computation time limit of 48 hours per instance was used in all experiments. For the DEC method the MILP solver adopted in the experiments is IBM ILOG CPLEX 12.6.

The experimental results are shown in Tables 4.1, 4.2 and 4.3 (one table for each benchmark set). For each instance we report the density of (the transitive closure of) the precedence graph (calculated as $\frac{2 \cdot |R|}{|V| \cdot (|V|-1)}$), the best known lower bound (LB) and upper bound (UB) (according to published results), and the results produced by each of two algorithms under study (DEC and B&B). For each algorithm, we report the LB and UB obtained, the size of the optimality gap (calculated as $\frac{UB-LB}{LB}$) and the time in seconds required to prove optimality (a dash is reported if the algorithm does not solve an instance to optimality). An instance name in italic indicates that the instance has not been solved to optimality yet. An instance name in boldface highlights an instance that has been closed for the first time in this work. Similarly, lower or upper bounds in boldface indicate new best known bounds found in this work. In the last row of each table, the average optimality gap is reported for each algorithm.

The TSPLIB instances in Table 4.1 are the most studied instances, and the quality of the bounds available is already extremely high. The DEC algorithm was able to solve to optimality 14 of the 41 instances, while the B&B algorithm closed 10 instances. The average optimality gaps are 6.4% for DEC and 34% for B&B. On these instances, DEC appears to perform better. Examining the time taken by each algorithm to close an instance leads to an interesting observation. While DEC needs a relatively long time to close some instances, B&B tends to prove optimality in a shorter time (on average, shorter than DEC) or to spend the available time without substantial improvements. It can be said that DEC exhibits a constant but slow rate of improvement of the bounds.

For approximately half of the SOPLIB06 instances in Table 4.2, no optimality proof has been provided yet. The DEC algorithm was able to close 18 instances, while B&B closed 21. The evolution of bounds over time for the two algorithms seems to follow the same pattern that was observed on TSPLIB. While DEC reaches optimality near the end of the computation time limit on some instances, B&B is typically either fast in closing an instance or fails to prove optimality within the time limit. This characteristic is reflected in the optimality gaps, which are 22.7% for DEC and 44.3% for B&B. A number of new improved bounds have been found: 9 lower bounds and 1 upper bound for B&B, 7 lower bounds for DEC. These improvements led to optimality proofs for 9 new instances: 7 by B&B and 2 by both the methods. Overall, it may be stated that B&B performs better than DEC on this benchmark set.

The results for the COMPILERS set are reported in Table 4.3. Optimality is

known for all but one of the 27 instances in this set [Shobaki and Jamal, 2015]. The B&B algorithm that was originally developed to solve instances in the compilers domain is performing better than DEC on this set. The number of closed instances is 26 for B&B and only 16 for DEC, with an average gap of 1.8% for B&B and 3.4% for DEC. It is interesting to observe that for these instances the bound improvement over time appears to follow a pattern similar to those already observed for TSPLIB and SOPLIB06 instances. DEC, apart from one case, tends to either close an instance quickly or fail. Finally, it is interesting to observe that DEC computed improved lower and upper bounds for the open instance.

Instances	**Density**	**Best known**		**DEC**				**B&B**			
	of P	LB	UB	LB	UB	Gap	Sec	LB	UB	Gap	Sec
br17.10	0.314	55	55	55	55	0	344.37	55	55	0	1.29
br17.12	0.359	55	55	55	55	0	491.53	55	55	0	181.02
ESC07	0.611	2125	2125	2125	2125	0	0.03	2125	2125	0	0.02
ESC11	0.359	2075	2075	2075	2075	0	0.12	2075	2075	0	0.28
ESC12	0.396	1675	1675	1675	1675	0	0.67	1675	1675	0	0.25
ESC25	0.177	1681	1681	1681	1681	0	23.48	1681	1681	0	46.13
ESC47	0.108	1288	1288	1288	1288	0	8168.49	917	3843	0.761	-
ESC63	0.173	62	62	62	62	0	79.63	55	76	0.276	-
ESC78	0.139	18230	18230	18230	18230	0	3216.3	9360	22600	0.586	-
ft53.1	0.082	7531	7531	7118	7612	0.065	-	5806	10404	0.442	-
ft53.2	0.094	8026	8026	7290	8113	0.101	-	5806	12175	0.523	-
ft53.3	0.225	10262	10262	8760	10310	0.15	-	5818	13435	0.567	-
ft53.4	0.604	14425	14425	13665	14425	0.053	-	14425	14425	0	4988.76
ft70.1	0.063	39313	39313	39033	39514	0.012	-	37603	46060	0.184	-
ft70.2	0.075	40101	40419	39372	40976	0.039	-	37667	48359	0.221	-
ft70.3	0.142	42535	42535	41104	42687	0.037	-	38320	52067	0.264	-
ft70.4	0.589	53530	53530	50527	53629	0.058	-	42193	54645	0.228	-
rbg048a	0.444	351	351	351	351	0	45.74	327	438	0.253	-
rbg050c	0.459	467	467	467	467	0	123.12	436	568	0.232	-
rbg109a	0.909	1038	1038	1038	1038	0	21910.1	1038	1038	0	2669.7
rbg150a	0.927	1750	1750	1747	1750	0.002	2446.44	1629	1999	0.185	-
rbg174a	0.929	2033	2033	2030	2033	0.001	-	1892	2444	0.226	-
rbg253a	0.948	2950	2950	2928	2950	0.007	-	2754	3558	0.226	-
rbg323a	0.928	3140	3140	3131	3146	0.005	-	2933	4032	0.273	-
rbg341a	0.937	2568	2568	2470	2626	0.059	-	2153	3786	0.431	-
rbg358a	0.886	2545	2545	2479	2654	0.066	-	2232	4110	0.457	-
rbg378a	0.894	2809	2816	2693	2922	0.078	-	2260	4109	0.45	-
kro124p.1	0.046	38762	39420	36551	41177	0.112	-	33498	52575	0.363	-
kro124p.2	0.053	39841	41336	36966	43311	0.146	-	33787	57723	0.415	-
kro124p.3	0.092	43904	49499	37814	53016	0.287	-	33872	68962	0.509	-
kro124p.4	0.496	73021	76103	55876	77139	0.276	-	39983	92438	0.567	-
p43.1	0.101	28140	28140	28083	28140	0.002	-	690	28480	0.976	-
p43.2	0.126	28480	28480	28124	28480	0.013	-	690	28795	0.976	-
p43.3	0.191	28835	28835	27309	28835	0.053	-	690	29300	0.976	-
p43.4	0.614	83005	83005	82933	83005	0.001	-	83005	83005	0	938.2
prob.100	0.048	1045	1163	1000	1973	0.493	-	761	2071	0.633	-
prob.42	0.116	243	243	243	243	0	32404.1	159	329	0.517	-
ry48p.1	0.091	15805	15805	14836	15805	0.061	-	12216	20190	0.395	-
ry48p.2	0.103	16074	16666	14974	16741	0.106	-	12216	19424	0.371	-
ry48p.3	0.193	19490	19894	16290	19894	0.181	-	12528	22483	0.443	-
ry48p.4	0.588	31446	31446	27061	31446	0.139	-	31446	31446	0	2471.05
Average Gap						0.064				0.34	

Table 4.1. TSPLIB Instances

Instances	**Density**	**Best known**		**DEC**				**B&B**			
	of P	LB	UB	LB	UB	Gap	Sec	LB	UB	Gap	Sec
R.200.100.1	0.02	61	61	61	61	0	3786.22	61	400	0.848	-
R.200.100.15	0.85	1257	1792	1243	2225	0.441	-	810	4074	0.801	-
R.200.100.30	0.96	4185	4216	**4216**	4216	0	86869.6	**4216**	4216	0	741.45
R.200.100.60	0.99	71749	71749	71749	71749	0	0.15	71749	71749	0	32.13
R.200.1000.1	0.02	1404	1404	1403	1404	0.001	-	1392	5022	0.723	-
R.200.1000.15	0.87	14565	20481	14689	24527	0.401	-	10104	40333	0.749	-
R.200.1000.30	0.96	40170	41196	**41196**	41196	0	170875	**41196**	41196	0	517.29
R.200.1000.60	0.99	71556	71556	71556	71556	0	0.15	71556	71556	0	35.29
R.300.100.1	0.01	26	26	26	104	0.75	-	26	363	0.928	-
R.300.100.15	0.91	2166	3161	2160	3796	0.431	-	1298	6640	0.805	-
R.300.100.30	0.97	5839	6120	5846	6133	0.047	-	**6120**	6120	0	3364.72
R.300.100.60	0.99	9726	9726	9726	9726	0	0.18	9726	9726	0	141.47
R.300.1000.1	0.01	1294	1294	1292	2011	0.358	-	1291	6078	0.788	-
R.300.1000.15	0.91	21096	29183	21077	36766	0.427	-	14352	57296	0.75	-
R.300.1000.30	0.96	51495	54147	51789	54744	0.054	-	**54147**	54147	0	8390.3
R.300.1000.60	0.99	109471	109471	109471	109471	0	0.32	109471	109471	0	176.08
R.400.100.1	0.01	13	13	13	90	0.856	-	13	285	0.954	-
R.400.100.15	0.93	2747	3906	2737	5115	0.465	-	1811	8794	0.794	-
R.400.100.30	0.98	7755	8165	8055	8167	0.014	-	**8165**	8165	0	65486.19
R.400.100.60	1.00	15228	15228	15228	15228	0	1.97	15228	15228	0	491.28
R.400.1000.1	0.01	1343	1343	1343	2032	0.339	-	1336	4437	0.699	-
R.400.1000.15	0.93	28159	29685	28207	49825	0.434	-	19886	81262	0.755	-
R.400.1000.30	0.98	79868	85132	82264	85223	0.035	-	**85128**	**85128**	0	2699.44
R.400.1000.60	0.99	140816	140816	140816	140816	0	0.51	140816	140816	0	494.2
R.500.100.1	0.01	4	4	4	85	0.953	-	4	383	0.99	-
R.500.100.15	0.94	3543	5361	3523	6813	0.483	-	2370	11486	0.794	-
R.500.100.30	0.98	8600	9665	8693	10026	0.133	-	**9665**	9665	0	21075.92
R.500.100.60	1.00	18240	18240	18240	18240	0	0.5	18240	18240	0	1301.61
R.500.1000.1	0.01	1316	1316	1315	2090	0.371	-	1313	6205	0.788	-
R.500.1000.15	0.94	32950	50725	32522	64092	0.493	-	22597	111129	0.797	-
R.500.1000.30	0.98	91272	98987	93698	100270	0.066	-	**98987**	98987	0	26041.65
R.500.1000.60	1.00	178212	178212	178212	178212	0	0.57	178212	178212	0	1466.67
R.600.100.1	0.01	1	1	1	85	0.988	-	1	378	0.997	-
R.600.100.15	0.95	3656	5684	3611	7479	0.517	-	2355	13271	0.823	-
R.600.100.30	0.98	11841	12465	12127	12527	0.032	-	**12465**	12465	0	39004.34
R.600.100.60	1.00	23293	23293	23293	23293	0	0.48	23293	23293	0	1945.58
R.600.1000.1	0.01	1337	1337	1337	1337	0	131432	1336	4931	0.729	-
R.600.1000.15	0.94	36546	57237	36575	73389	0.502	-	27096	120975	0.776	-
R.600.1000.30	0.98	116037	126789	118311	129869	0.089	-	79564	178608	0.555	-
R.600.1000.60	1.00	214608	214608	214608	214608	0	0.67	214608	214608	0	3652.19
R.700.100.1	0.01	1	1	1	1	0	13781.6	1	446	0.998	-
R.700.100.15	0.96	4494	7311	4499	9213	0.512	-	3117	14643	0.787	-
R.700.100.30	0.99	13663	14510	14084	14601	0.035	-	8994	19138	0.53	-
R.700.100.60	1.00	24102	24102	24102	24102	0	1.49	24102	24102	0	6561.8
R.700.1000.1	0.01	1231	1231	1231	1231	0	56712	1228	4886	0.749	-
R.700.1000.15	0.96	40662	66837	40808	88621	0.54	-	29226	151331	0.807	-
R.700.1000.30	0.99	118718	134474	121173	138442	0.125	-	80969	187072	0.567	-
R.700.1000.60	1.00	245589	245589	245589	245589	0	1.54	245589	245589	0	8243.04
Average gap						0.227				0.443	

Table 4.2. SOPLIB Instances

Instances	**Density**	**Best known**		**DEC**				**B&B**			
	of P	LB	UB	LB	UB	Gap	Sec	LB	UB	Gap	Sec
gsm.153.124	0.97	1109	1109	1109	1109	0	972.61	1109	1109	0	1.82
gsm.444.350	0.99	2745	2745	2745	2745	0	1.29	2745	2745	0	1.58
gsm.462.77	0.84	577	577	577	577	0	3.77	577	577	0	17.47
jpeg.1483.25	0.48	93	93	93	93	0	9.47	93	93	0	6.26
jpeg.3184.107	0.89	769	793	769	793	0.03	-	791	791	0	82.19
jpeg.3195.85	0.74	28	68	28	68	0.588	-	68	68	0	47.66
jpeg.3198.93	0.75	304	312	304	312	0.026	-	312	312	0	53.2
jpeg.3203.135	0.90	830	852	830	852	0.026	-	850	850	0	306.84
jpeg.3740.15	0.26	40	40	40	40	0	4.12	40	40	0	3.12
jpeg.4154.36	0.63	167	167	167	167	0	288.28	167	167	0	18.55
jpeg.4753.54	0.77	241	245	241	245	0.016	-	245	245	0	25
susan.248.197	0.94	1338	1338	1304	1338	0.025	-	1338	1338	0	672.84
susan.260.158	0.92	1016	1016	945	1016	0.07	-	1016	1016	0	81.91
susan.343.182	0.94	1207	1207	1159	1207	0.04	-	1207	1207	0	195.43
typeset.10192.123	*0.74*	*214*	*630*	***565***	***602***	*0.061*	-	*328*	*634*	*0.483*	-
typeset.10835.26	0.35	127	127	127	127	0	114.55	127	127	0	55.74
typeset.12395.43	0.52	174	174	168	174	0.034	-	174	174	0	527.88
typeset.15087.23	0.56	98	98	98	98	0	0.33	98	98	0	10.01
typeset.15577.36	0.56	155	155	154	155	0.006	-	155	155	0	114.64
typeset.16000.68	0.66	84	84	84	84	0	6624.72	84	84	0	52.91
typeset.1723.25	0.24	64	64	64	64	0	40.32	64	64	0	40.38
typeset.19972.246	0.99	2018	2018	2018	2018	0	2.63	2018	2018	0	1.84
typeset.4391.240	0.98	1605	1605	1605	1605	0	4815.67	1605	1605	0	40.29
typeset.4597.45	0.49	184	184	184	184	0	22868	184	184	0	1017.7
typeset.4724.433	0.99	3466	3466	3466	3466	0	46.66	3466	3466	0	11.22
typeset.5797.33	0.75	131	131	131	131	0	0.48	131	131	0	69.5
typeset.5881.246	0.99	1732	1732	1732	1732	0	686.96	1732	1732	0	47.73
Average Gap						0.034				0.018	

Table 4.3. COMPILERS Instances

4.4 Discussions & Conclusions

In the Sections 4.1 -4.3 we studied experimentally two exact algorithms for the same problem that arose from different domains. Our experiments led to closing nine previously open instances and getting improved upper and lower bounds for seventeen other instances. A closer look at the results leads to more general conclusions about the relative performance of the two algorithms. Although it is difficult to predict how successful an algorithm will be on a given instance, it is possible to observe that DEC usually performs well on instances with extreme densities of precedence constraints (either very low or a very high) but does not perform very well on instances with medium densities. The reason is that the problem can be split into subproblems more easily when the densities are extreme. If the density is low, it will be split into few large subproblems and if it is high it will be split in multiple smaller subproblems. All in all, density is a powerful predictor for the success of DEC on a dataset. On the other hand,

B&B appears to be less effective on instances with a low density of precedence constraints but is more effective than DEC on instances with medium densities. It is also interesting to note that B&B performs very well on the COMPILERS instances that are characterised by symmetrical cost graphs (the cost from vertex *i* to vertex *j* is always equal to the cost from vertex *j* to vertex *i* for all COMPILERS instances). The success of B&B on such instances is attributed to the effectiveness of the history utilization pruning technique on the more symmetrical cases. When edge costs are symmetrical, it will be more likely for the current sub-problem to relate to a previously visited sub-problem, thus enabling more history-based pruning.

4.5 Enhancing the B&B algorithm

As a continuation of the Section 4.2, in the current Section, we show that using a Minimum Cost Perfect Matching (MCPM)-based lower bound and a new local search heuristic significantly improves the performance of the pure B&B algorithm.

4.5.1 Enumeration Scheme

The B&B algorithm proposed in the following sections applies at each tree node a lower bound technique and two pruning techniques: local-search domination and history utilization. The lower bound technique is based on MCPM as described in Section 4.2, but it was computed using a minimum-cost network flow algorithm. As explained in Section 4.5.2, MCPM is computed in the current work using the dynamic Hungarian algorithm [Mills-Tettey et al., 2007]. The experimental results show that using the dynamic Hungarian algorithm to compute a MCPM-based lower bound at each tree node significantly improves the performance of the B&B algorithm. The local search heuristic is a new pruning technique proposed in this paper. As explained in Section 4.5.4, the idea is determining, in linear time, if there is a better permutation of the vertices in the current node's partial path and pruning the sub-tree below the current node if such a permutation is found. The history utilization pruning technique used in this work is the same technique used in [Shobaki and Jamal, 2015]. The idea of this technique to store information about previously explored sub-problems and using the stored information to quickly process similar sub-problems that may be encountered later. Two sub-problems are similar if the set of vertices in the partial paths is the same in both sub-problems and the two sets end with the

same vertex. By exploiting the similarity, pruning or lower-bound computation may be done in constant time.

4.5.2 Lower Bounds

In a previous work [Shobaki and Jamal, 2015], lower bounds were computed using an edge-based technique as described in the Section 4.2. The problem with this approach is that the same edge could be used more than once, thus violating the in-degree/out-degree constraint of the SOP and potentially producing a loose lower bound. The in-degree/out-degree constraint is that each vertex, other than the start and end vertices, must have exactly one outgoing edge and one incoming edge. To mitigate this disadvantage of the edge-based lower bound, the in-degree/out-degree constraint may be imposed by formulating the problem as a Minimum-Cost Perfect Matching Problem to ensure that each edge is used exactly once. First, we define the MCPM Problem, and then we describe how it can be used to compute a tight lower bound for the SOP. Minimum-Cost Perfect Matching (MCPM), also known as the *Assignment Problem*, is an optimization problem in which the objective is finding a maximum-cardinality matching with minimum cost in a weighted bipartite graph [Cormen, 2009]. Given a bipartite graph $G = (V, E)$ where V is set of vertices and $E = \{(i, j)|i, j \in V\}$ is the set of edges. A graph is bipartite if the vertex set V can be partitioned into two sets A and B, called the bipartition, such that every edge has one endpoint in A and one endpoint in B. A matching $M \subset E$ is a set of edges such that every vertex $v \in V$ is incident to at most one edge in M. If a vertex has no edge in M incident to it, that vertex is said to be unmatched. A matching is *perfect* if every vertex in the graph is matched. The cost of matching is defined as

$$C(M) = \sum_{e \in M} c(e)$$

Given a cost $c_{i,j}$ for each edge $e \in E$, a matching is called a *minimum-cost perfect matching* if M has the minimum cost among all perfect matchings.

MCPM may be used to compute a lower bound for the SOP as follows. Given a cost graph, we construct a bipartite graph in which each vertex in the cost graph appears in both partitions of the bipartite graph. More precisely, for each vertex v in the cost graph, we create a vertex v_A in the A partition and a vertex v_B in the B partition. The cost-graph edges are then added between the vertices in the A partition and the vertices in the B partition, such that for each edge (u, v) in the cost graph, we create an edge (v_A, v_B) in the bipartite graph. In a prefect matching of the resulting bipartite graph, each edge in cost graph will

be used exactly once and each vertex will have exactly one outgoing edge and one incoming edge, thus satisfying the in-degree/out-degree constraint of the SOP. The start and end vertices will have to be handled differently by either not including them in the bipartite graph or including them and then subtracting the extra edge weight from the resulting minimum cost. In this paper, we take the latter approach, as illustrated in the example of Section 4.5.3.

Although a perfect matching of the resulting bipartite graph satisfies the in-degree/out-degree constraint, it does not necessarily give one Hamiltonian path; the matching may correspond to one partial path as well as one or more cycles. Therefore, the bipartite matching formulation is a relaxation of the SOP, and a MCPM gives a lower bound on the optimal solution to the given SOP. It is important to note here that since this MCPM lower bound has nothing to do with the precedence constraints, it is a valid lower bound for the TSP as well. Using the assignment problem to compute a lower bound for the TSP has been previously proposed by [Christofides, 1972]. [Miller et al., 1991] have used a matching-based lower bound in a B&B algorithm for the SOP.

In the following sections, we use the *Dynamic Hungarian algorithm* [Mills-Tettey et al., 2007] to solve the MCPM problem. The dynamic Hungarian algorithm is a dynamic version of the original Hungarian algorithm [Kuhn, 1955] that makes it possible to repair an already-computed matching instead of constructing a new matching from scratch when the graph is partially modified. This idea works very well in a B&B solver, where each decision (branch in the enumeration tree) only makes a small change to the matching. The change is fixing one edge in the matching. The dynamic Hungarian algorithm runs in $O(kn^2)$ time, where k is the number of changed rows/columns in the cost matrix. When k is a constant, which is the case in a B&B solver, this leads to an $O(n^2)$ algorithm for repairing a matching. This is a significant improvement compared to the original Hungarian algorithm that runs in $O(n^3)$ time. Our experimental results show that using the dynamic Hungarian algorithm does indeed produce a significantly faster B&B solver than using the original Hungarian algorithm.

4.5.3 Dynamic Hungarian Algorithm

The dynamic Hungarian algorithm is used to compute lower bounds in our proposed B&B algorithm as follows. Before enumeration starts, a bipartite graph is constructed as described above using the original weights of the given cost graph. This lower bound is called the *static lower bound* to distinguish it from the *dynamic lower bounds* that are computed during enumeration. Each forward step during enumeration augments the partial path by adding a new vertex to it.

If the current partial path ends with vertex u and the newly added vertex is v, this adds Edge (u, v) to the partial path. Therefore, the MCPM must be repaired by forcing Edge (u, v) to be in the set of edges that form the matching. This is accomplished by modifying the cost matrix such that all entries in Row u and Column v are set to infinity, except for Entry (u, v). The MCPM is repaired using the dynamic Hungarian algorithm instead of re-computing a new matching using the original Hungarian algorithm. The repaired MCPM gives a tighter lower bound at the current tree node by satisfying the constraint of adding Edge (u, v) to the matching. This tightened lower bound is called a dynamic lower bound, since it changes dynamically during enumeration. When the B&B algorithm backtracks to the previous node, it reverses the changes that have been made to the cost matrix (setting some cells to infinity) and restores the previous dynamic lower bound. As detailed in the paper of [Mills-Tettey et al., 2007], the dynamic Hungarian algorithm computes a new matching by un-matching the affected vertices and reducing the cardinality of the matching by one. If Row i and Column j in the cost matrix change, new feasible values are computed for the dual variables or according to the following two equations

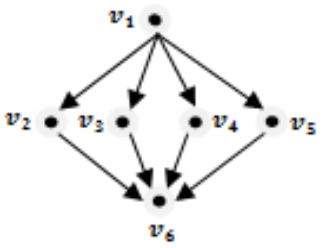
$$a_i = min_j(c_{i,j} - \beta_j)$$

$$\beta_j = min_i(c_{i,j} - a_i)$$

In the dynamic Hungarian algorithm, only a single iteration of the original Hungarian algorithm is executed for each column/row change. A single iteration of the Hungarian algorithm has a complexity of $O(V^2)$. Since repairing the matching to compute a dynamic lower bound at each tree node in our B&B algorithm involves modifying one row and one column, the overall complexity of computing a new dynamic lower bound in our algorithm is $O(V^2)$.

In the rest of the section, we illustrate how dynamic lower bounds are computed during the proposed B&B algorithm using MCPM. Figure 4.4 shows an instance of the SOP. As shown in the figure, u_1 and u_6 are the start and end vertices, and vertices v_2, v_3, v_4 and v_5 are independent vertices that may be visited in any order. As mentioned in Section 4.5.2, the cost matrix is the adjacency matrix representation of the cost graph. A weight of ∞ for Edge implies that cannot be visited after. A valid numerical value for ∞ is one plus the maximum edge weight in the cost matrix. Substituting this numerical value for ∞ gives the following cost matrix.

As defined in Section 4.5.2, a solution to the SOP is a Hamiltonian path that consists of $|V| - 1$ edges, while a MCPM consists of $|V|$ edges. In a Hamiltonian path, the start vertex does not have an incoming edge and the end vertex does



	v_1	v_2	v_3	v_4	v_5	v_6
u_1	∞	2	5	0	3	∞
u_2	∞	∞	2	20	20	0
u_3	∞	2	∞	20	20	0
u_4	∞	2	3	∞	4	0
u_5	∞	20	2	4	∞	0
u_6	∞	∞	∞	∞	∞	∞

Figure 4.4. Example used to illustrate dynamic lower bound computation using MCPM. The graph is the precedence graph and the matrix is the cost matrix in which $Entry(i, j)$ is the weight of Edge $e = (u_i, v_j)$

not have an outgoing edge. This is accounted for by setting the values of all entries in the first column and the last row in the cost matrix to infinity. Since each of the other vertices has at least one finite-weight outgoing edge and one finite-weight incoming edge, the end vertex will always be matched to the start vertex in any MCPM, and the weight of the edge connecting them will always be ∞. Thus, a lower bound on the cost of the Hamiltonian path is computed by subtracting the numerical value that is substituted for ∞ from the cost of the MCPM. In this example, the numerical value to be subtracted is 22. It should be noted here that the Hungarian algorithm finds a maximum-cost matching rather than a minimum-cost matching. However, it is fairly easy to transform the minimization problem into a maximization problem by multiplying edge weights by -1. To simplify the presentation, this detail is hidden, and the example is presented as a minimization problem using the original positive weights.

Before enumeration starts, a static lower bound on the optimal solution is computed by finding a MCPM using the Hungarian algorithm. Applying the Hungarian algorithm to this example gives an initial matching that consists of the following set of edges: $\{(1,4),(2,3),(3,2),(4,5),(5,6),(6,1)\}$.

Subtracting 22 from the cost of this matching gives a lower bound of 8 on the cost of the Hamiltonian path. During enumeration, a path is constructed incrementally by selecting a *ready* vertex and adding it to the path. A vertex is ready if all of its predecessors in the precedence graph have been visited. Initially, the start vertex v_1 is the only ready vertex. Adding v_1 to the path does not add any constraints to the problem and thus does not lead to modifying the cost matrix. The dynamic lower bound remains equal to the static lower bound.

After adding v_1, vertices v_2 through v_5 become ready and one of them must be selected to add an edge to the partial path. Suppose that v_5 is selected, thus

adding Edge (v_1, v_5) to the partial path. To account for this constraint in the MCPM problem, the cost matrix must be modified to force the selection of Edge (v_1, v_5) in the matching. Thus, the matrix is modified by setting all entries in the v_1 row and v_5 column to ∞, except for Entry (u, v) . The result can be seen in Table 4.4.

	u1	**u2**	**u3**	**u4**	**u5**	**u6**
u1	22	2	5	0	3	22
u2	22	22	2	20	20	0
u3	22	2	22	20	20	0
u4	22	2	3	22	4	0
u5	22	20	2	4	22	0
u6	22	22	22	22	22	22

Table 4.4. Example Cost Matrix for the Dynamic Hungarian Algorithm

Applying the dynamic Hungarian algorithm to this modified cost matrix, repairs the previous matching and produces the following new matching:

$$(1,5),(2,3),(3,2),(4,6),(5,4),(6,1)$$

Subtracting 22 from the cost of this matching, gives a new dynamic lower bound of 11, which is tighter than the previous bound of 8. The process is repeated at every new node in the enumeration tree.

4.5.4 Local Search

In the proposed B&B algorithm a local-search domination technique is used. In this technique, local search is conduced at each tree node for a better partial solution. If a better partial solution is found, the sub-tree below the current tree node is pruned. A partial solution to the SOP is a partial path (sequence of vertices). Given a partial path $P = (x, v_1, v_2, v_3, \ldots, v_n, y)$ with cost C at a certain tree node, if there exists a permutation of the vertices through that has a lower cost than C, the sub-tree below this node can be pruned. Since the number of permutations is a factorial function of n, the set of all possible permutations cannot be explored entirely within reasonable time. Thus, we limit the search for a better permutation to a linear number of permutations by changing the position of only one vertex in the sequence. More specifically, we move v_n one step back until it either reaches the first position or results in a precedence violating sequence. If this limited local search finds a partial solution with cost less than C,

the sub-tree below the current node is pruned. An example is shown in Figure 4.5. In this example, the partial path at the current node is $(1, 2, 3, 4, 5, 6, 7)$ with cost 25, and v_n is Vertex 6. By examining the permutations in which the position of Vertex 6 is varied, a partial path with cost 21 is found. This implies that the partial path at the current node cannot lead to an optimal solution and the sub-tree below the current node may be pruned. The above-described local-search domination technique may lead to finding a better partial path than the current path. In theory, the better partial path may be used instead of the current partial path in searching the rest of the tree. However, due to complexities related to maintaining dynamic-lower bound information at each node, this idea is not currently implemented. Resolving these complexities to take advantage of the improved partial path is left for future work.

$$Original\ Sequence = \{1, 2, 3, 4, 5, \mathbf{6} \mid 7\}, \qquad c = 25$$

$$Local\ Search \begin{cases} \{1, 2, 3, 4, \mathbf{6}, 5 \mid 7\}, & c = 27 \\ \{1, 2, 3, \mathbf{6}, 4, 5 \mid 7\}, & c = 31 \\ \{1, 2, \mathbf{6}, 3, 4, 5 \mid 7\}, & c = 40 \\ \{1, \mathbf{6}, 2, 3, 4, 5 \mid 7\}, & c = 21 \\ \{\mathbf{6}, 1, 2, 3, 4, 5 \mid 7\}, & c = 33 \end{cases}$$

Figure 4.5. Local Search Domination example

4.5.5 Experiments

In this Section, we present the results of the experimental evaluation of the proposed algorithm. The proposed algorithm was evaluated using three benchmark suites: TSPLIB [TSP, n.d.], SOPLIB06 [Montemanni et al., 2008] and COMPILERS [Shobaki and Jamal, 2015]. The experiments were performed on a Quad-Core AMD Opteron 2350 processor running at 2.0 GHz with 32 GB of RAM. The time limit per instance was set to 48 hours.

4.5.5.1 Effectiveness of the new Lower Bound Technique

In this Section, we evaluate the effectiveness of the dynamic lower bound technique proposed in this paper. The new lower bound technique that is based on the dynamic Hungarian algorithm is compared with the network-flow-based

MCPM lower bound technique previously adopted in our work [Papapanagiotou et al., 2015b]. As explained in Section 4.5.2, both lower bounds are based on MCPM, but the dynamic Hungarian algorithm computes these lower bounds in $O(n^2)$, while the network-flow algorithm computes them in $O(n^3)$ time. In this comparison, only the instances that are solved optimally using both lower bound techniques are considered. In both cases, local-search domination and history utilization were applied. Tables 4.5, 4.6, 4.7 show the solution time in seconds required to reach optimality for each of these instances. The results in these tables show that the speedup achieved by using the dynamic Hungarian algorithm is quite substantial. On average, using the dynamic Hungarian algorithm gives a speedup of 98% on TSPLIB (Table 4.5), 98% on SOPLIB (Table 4.6) and 75% on COMPILERS (Table 4.7).

TSPLIB Instance	**Network Flow Time (s)**	**Dynamic Hungarian Time (s)**	**% Improvement**
br17.10	1.29	0.12	90.7
br17.12	181.02	0.06	99.97
ESC07	0.02	0	100
ESC11	0.28	0	100
ESC12	0.25	0	100
ESC25	46.13	0.19	99.59
ft53.4	4988.76	69.27	98.61
rbg109a	2669.7	35.49	98.67
p43.4	938.2	22.98	97.55
ry48p.4	2471.05	58.28	97.64
Average			98.27

Table 4.5. Comparison of the proposed dynamic-Hungarian-based lower bound with the previously published network-flow-based lower bound for the TSPLIB instances that could be solved using both lower bounds.

4.5.5.2 Effectiveness of Local-Search Domination

An experiment was conducted to evaluate the effectiveness of the local-search domination technique. The proposed B&B was applied to the instances under study once with local-search domination enabled and once with local search-domination disabled. In both cases, the lower bounds were computed using the dynamic Hungarian algorithm and history utilization was applied. The effectiveness of the local-search domination technique was evaluated by measuring the solution time and the number of sub-problems (tree nodes) explored in each case. The results of this comparison are shown in Tables 4, 5 and 6, which show the number of sub-problems explored and the solution time needed to solve an

SOPLIB06 Instance	Network Flow Time (s)	Dynamic Hungarian Time (s)	% Improvement
R.200.100.30	741.45	9.52	98.72
R.200.100.60	32.13	0.06	99.81
R.200.1000.30	517.29	11.79	97.72
R.200.1000.60	35.29	0.2	99.43
R.300.100.30	3364.72	78.07	97.68
R.300.100.60	141.47	0.63	99.55
R.300.1000.30	8390.3	194.16	97.69
R.300.1000.60	176.08	0.8	99.55
R.400.100.30	65486.19	317.08	99.52
R.400.100.60	491.28	2.3	99.53
R.400.1000.30	2699.44	463.11	82.84
R.400.1000.60	494.2	2.4	99.51
R.500.100.30	21075.92	1006.82	95.22
R.500.100.60	1301.61	4.52	99.65
R.500.1000.30	26041.65	1444.57	94.45
R.500.1000.60	1466.67	4.56	99.69
R.600.100.30	39004.34	708.1	98.18
R.600.100.60	1945.58	4.12	99.79
R.600.1000.60	3652.19	6.95	99.81
R.700.100.60	6561.8	11.86	99.82
R.700.1000.60	8243.04	9.21	99.89
Average			98

Table 4.6. Comparison of the proposed dynamic-Hungarian-based lower bound with the previously published network-flow-based lower bound for the SOPLIB instances that could be solved using both lower bounds.

COMPILER Instance	Network Flow Time (s)	Dynamic Hungarian Time (s)	% Improvement
gsm.153.124	1.82	0.17	90.66
gsm.444.350	1.58	0.27	82.91
gsm.462.77	17.47	12.45	28.73
jpeg.1483.25	6.26	0.24	96.17
jpeg.3184.107	82.19	46.42	43.52
jpeg.3195.85	47.66	18.19	61.83
jpeg.3198.93	53.2	22.85	57.05
jpeg.3203.135	306.84	52.55	82.87
jpeg.3740.15	3.12	0.39	87.5
jpeg.4154.36	18.55	1.38	92.56
jpeg.4753.54	25	6.42	74.32
susan.248.197	672.84	612.98	8.9
susan.260.158	81.91	39.58	51.68
susan.343.182	195.43	53.2	72.78
typeset.10835.26	55.74	5.13	90.8
typeset.12395.43	527.88	40.82	92.27
typeset.15087.23	10.01	0	100
typeset.15577.36	114.64	1.89	98.35
typeset.16000.68	52.91	2.03	96.16
typeset.1723.25	40.38	1.06	97.37
typeset.19972.246	1.84	0.08	95.65
typeset.4391.240	40.29	23.51	41.65
typeset.4597.45	1017.7	24.38	97.6
typeset.4724.433	11.22	1.85	83.51
typeset.5797.33	69.5	0.03	99.96
typeset.5881.246	47.73	37.16	22.15
Average			74.88

Table 4.7. Comparison of the proposed dynamic-Hungarian-based lower bound with the previously published network-flow-based lower bound for the COMPILER instances that could be solved using both lower bounds

instance to optimality for the instances that could be solved with and without local-search domination. In each table, the first percentage improvement is the percentage reduction in the number of explored sub-problems and the second percentage improvement is the percentage reduction in solution time. The percentage improvements in these tables show that the impact of the local-search domination technique is quite significant. The percentage reduction in the number of explored sub-problems ranges from 0% to 22% with an average of 8% on TSPLIB, from 0% to 58% with an average of 22% on SOPLIB and from 0% to 60% with an average of 17% on COMPILERS. The percentage reduction in solution time ranges from -8% (it was slower on ESC25) to 12% with an average of 4% on SOPLIB, from 1% to 33% with an average of 13% on SOPLIB and from 6% to 72% (excluding typeset.15087.23 which is solved very fast) with an average of 37% on COMPILERS. The local-search domination technique described in this paper reduces the number of explored sub-problems (tree node) at the cost of increasing the computation time per sub-problem (tree node). The experimental results in this Section show that the reduction in the number of explored sub-problems overweighs the increase in the computation time per tree node, thus giving an overall reduction in solution time. There is only one instance (ESC25) on which enabling the local-search domination technique resulted in a negative impact on the solution time.

TSPLib Instance	**Sub-problems**		**%**	**Time (s)**		**%**
	LS OFF	**LS ON**	**Improvement**	**LS OFF**	**LS ON**	**Improvement**
br17.10	75,903	75,903	0	0.18	0.18	0
br17.12	41,831	41,831	0	0.12	0.12	0
ESC07	39	39	0	0	0	0
ESC11	417	417	0	0.02	0.02	0
ESC12	3,727	3,634	2.5	0.03	0.03	0
ESC25	13,227	13,227	0	0.26	0.28	-7.69
ESC47	544,080	544,080	0	31.66	30.28	4.36
ESC63	2,287	2,287	0	0.12	0.11	8.33
ft53.4	41,322,976	33,785,419	18.24	158.76	145.12	8.59
ft70.4	1,645,159,076	1,448,510,681	11.95	5100.47	4887.66	4.17
rbg109a	6,151,882	4,769,686	22.47	33.02	30.4	7.93
rbg150a	6,610,432	5,453,541	17.5	59.1	54.47	7.83
p43.4	4,788,244	4,036,323	15.7	21.42	20.05	6.4
ry48p.4	28,096,631	22,204,422	20.97	57.96	51.24	11.59
Average			8%			4%

Table 4.8. Number of sub-problems explored and solution time needed with and without local search on TSPLIB

SOPLIB06 Instance	Sub-problems		%	Solution Time (s)		%
	LS OFF	LS ON	Improvement	LS OFF	LS ON	Improvement
R.200.100.1	4,995,942	4,995,942	0	1749.69	1666.15	4.77
R.200.100.30	5,344,788	4,594,598	14.04	29.33	27.31	6.89
R.200.100.60	6,718	5,666	15.66	0.33	0.32	3.03
R.200.1000.30	3,125,293	2,612,345	16.41	15.93	14.5	8.98
R.200.1000.60	38,940	22,990	40.96	0.55	0.46	16.36
R.300.100.30	9,868,973	8,738,422	11.46	80.21	69.04	13.93
R.300.100.60	5,984	5,418	9.46	0.86	0.81	5.81
R.300.1000.30	18,342,394	15,462,135	15.7	151.92	129.16	14.98
R.300.1000.60	40,863	21,918	46.36	1.19	0.9	24.37
R.400.100.30	26,325,040	22,089,735	16.09	253.23	226.29	10.64
R.400.100.60	115,258	89,644	22.22	2.93	2.76	5.8
R.400.1000.30	51,568,457	44,236,668	14.22	403.53	364.97	9.56
R.400.1000.60	237,008	99,602	57.98	4.28	2.85	33.41
R.500.100.30	92,852,231	74,071,755	20.23	1048.51	861.86	17.8
R.500.100.60	189,855	101,058	46.77	6.41	5.17	19.34
R.500.1000.30	92,042,506	88,745,961	3.58	956.66	944.15	1.31
R.500.1000.60	226,812	144,201	36.42	6.25	5.29	15.36
R.600.100.30	51,131,233	43,877,806	14.19	927.75	804.83	13.25
R.600.100.60	242,581	119,753	50.63	8.78	6.98	20.5
R.600.1000.30	143,295,846	126,106,542	12	1981.83	1651.4	16.67
R.600.1000.60	337,805	264,640	21.66	11.8	10.35	12.29
R.700.100.30	213,613,539	197,869,163	7.37	3231.7	2964.68	8.26
R.700.100.60	593,700	386,940	34.83	20.39	16.66	18.29
R.700.1000.30	191,947,642	181,423,750	5.48	3299.94	2974.59	9.86
R.700.1000.60	424,275	317,418	25.19	16.99	14.96	11.95
Average			22%			13%

Table 4.9. Number of sub-problems explored and solution time needed with and without local search on SOPLIB

COMPILER Instance	Sub-problems		%	Solution Time (s)		%
	LS OFF	LS ON	Improvement	LS OFF	LS ON	Improvement
gsm.153.124	13,594	10,543	22.44	0.31	0.17	45.16
gsm.444.350	1,803	1,803	0	0.52	0.27	48.08
gsm.462.77	1,120,024	994,779	11.18	16.01	12.45	22.24
jpeg.1483.25	48,955	44,842	8.4	0.33	0.24	27.27
jpeg.3184.107	5,252,360	4,297,369	18.18	56.89	46.42	18.4
jpeg.3195.85	1,654,926	1,184,444	28.43	24.17	18.19	24.74
jpeg.3198.93	2,111,955	2,108,240	0.18	49.36	22.85	53.71
jpeg.3203.135	10,349,003	7,478,110	27.74	72.05	52.55	27.06
jpeg.3740.15	254,645	224,565	11.81	0.5	0.39	22
jpeg.4154.36	198,436	171,206	13.72	1.84	1.38	25
jpeg.4753.54	1,653,784	1,290,876	21.94	8.54	6.42	24.82
susan.248.197	23,107,737	19,463,327	15.77	695.49	612.98	11.86
susan.260.158	3,971,946	2,819,246	29.02	53.04	39.58	25.38
susan.343.182	3,713,106	3,713,106	0	56.44	53.2	5.74
typeset.10835.26	2,226,650	2,226,650	0	5.51	5.13	6.9
typeset.12395.43	6,163,009	5,476,933	11.13	56.6	40.82	27.88
typeset.15087.23	673	673	0	0.01	0	100
typeset.15577.36	733,038	667,007	9.01	4.1	1.89	53.9
typeset.16000.68	494,344	253,504	48.72	7.19	2.03	71.77
typeset.1723.25	747,446	640,267	14.34	2.09	1.06	49.28
typeset.19972.246	1,977	788	60.14	0.27	0.08	70.37
typeset.4391.240	1,120,853	825,698	26.33	50.31	23.51	53.27
typeset.4597.45	9,106,973	8,494,767	6.72	49.03	24.38	50.28
typeset.4724.433	71,680	44,955	37.28	3.2	1.85	42.19
typeset.5797.33	2,989	2,719	9.03	0.05	0.03	40
typeset.5881.246	690,923	553,126	19.94	48.45	37.16	23.3
Average			17.36%			37.33%

Table 4.10. Number of sub-problems explored and solution time needed with and without local search on COMPILER

4.5.5.3 Overall Results

In this Section, we present the full performance results of the proposed B&B on all three benchmark suites. These experiments were run using our most powerful combination of techniques, including the dynamic Hungarian lower bounds and the local-search domination technique that are described in the current paper as well as the history utilization technique that was described in our previous work [Shobaki and Jamal, 2015]. The results are shown in Tables 4.11, 4.12 and 4.13. The second column in each table shows the density of the transitive closure of each instance's precedence graph, which is defined as $\frac{2|R|}{|V|(|V|-1)}$. The third column shows the cost of the initial solution produced by a heuristic method before starting the B&B search. The fourth column shows the optimal cost if known. If the instance is still open, the best known lower and upper bounds are shown in the fifth column. The last column shows the results of the proposed algorithm. The first and second sub-columns in that column show the best bounds computed by the proposed algorithm, and the third sub-column shows the optimality gap, which is defined as . Finally, the fourth sub-column in the last column shows the solution time in seconds if the problem is solved optimally within the time limit. The last row in each table shows the average optimality gap. Instances in bold are solved optimally for the first time in this work. Table 4.12 shows that the proposed algorithm proved the optimality of 8 SOPLIB instances that were previously open. Instances in italic are solved in this work but were not solved in our previous work. The results in these tables show that the proposed algorithm closed eight SOPLIB instances that were open. The results also show that the current work closed six TSPLIB instances and one SOPLIB instance that were not closed in our previous work. The current work also significantly reduced the sizes of the optimality gaps on all benchmark suites. Compared to [Papapanagiotou et al., 2015b], the enhanced algorithm proposed in the current paper reduces the average size of the optimality gap from 0.340 to 0.217 on TSPLIB, from 0.443 to 0.122 on SOPLIB and from 0.018 to 0.004 on COMPILERS.

4.6 Discussion and Conclusions

In this chapter we presented an enhanced B&B algorithm for the sequential ordering problem. The enhancement is due to two effective techniques that are proposed in this paper. The first technique is using the dynamic Hungarian algorithm to compute a tight dynamic lower bound based on a MCPM formulation. The second technique is a local-search domination technique. The experimental

TSPLIB	**Density**	**Starting**	**Optimal**	**Best Known**		**Proposed B&B**			
Instance	**of P**	**Cost**	**Cost**	**LB**	**UB**	**LB**	**UB**	**Gap**	**Time (s)**
br17.10	0.314	79	55			55	55	0	0.19
br17.12	0.359	79	55			55	55	0	0.1
ESC07	0.611	2300	2125			2125	2125	0	0
ESC11	0.359	2715	2075			2075	2075	0	0
ESC12	0.396	2034	1675			1675	1675	0	0
ESC25	0.177	2995	1681			1681	1681	0	0.13
ESC47	*0.108*	*3606*	*1288*			*1288*	*1288*	*0*	*35.01*
ESC63	*0.173*	*68*	*62*			*62*	*62*	*0*	*0.1*
ESC78	0.139	19715	18230			18205	19275	0.056	
ft53.1	0.082	10404	7531			5931	8411	0.295	
ft53.2	0.094	11652	8026			5931	10366	0.428	
ft53.3	0.225	15127	10262			6662	11998	0.445	
ft53.4	0.604	18549	14425			14425	14425	0	175.05
ft70.1	0.063	46060	39313			37978	43256	0.122	
ft70.2	0.075	47688		40101	40419	38042	43034	0.116	
ft70.3	0.142	50764	42535			38968	46271	0.158	
ft70.4	0.589	59795	53530			53530	53530	0	5716.5
rbg048a	0.444	464	351			337	361	0.066	
rbg050c	*0.459*	*532*	*467*			*467*	*467*	*0*	*4693.27*
rbg109a	0.909	1217	1038			1038	1038	0	134.3
rbg150a	*0.927*	*1973*	*1750*			*1750*	*1750*	*0*	*28.97*
rbg174a	*0.929*	*2281*	*2033*			*2033*	*2033*	*0*	*3995.7*
rbg253a	0.948	3327	2950			2834	2976	0.048	
rbg323a	0.928	3687	3140			3046	3687	0.174	
rbg341a	0.937	3342	2568			2274	2952	0.23	
rbg358a	0.886	3845	2545			2389	3318	0.28	
rbg378a	0.894	3824		2809	2816	2520	3401	0.259	
kro124p.1	0.046	52575		38762	39420	33978	48914	0.305	
kro124p.2	0.053	51094		39841	41336	34267	51094	0.329	
kro124p.3	0.092	70844		43904	49499	34535	62671	0.449	
kro124p.4	0.496	93107		73021	76103	46505	84585	0.45	
p43.1	0.101	29010	28140			900	28260	0.968	
p43.2	0.126	29570	28480			900	28565	0.968	
p43.3	0.191	31340	28835			970	28885	0.966	
p43.4	0.614	84425	83005			83005	83005	0	9.78
prob.100	0.048	2414		1045	1163	773	1704	0.546	
prob.42	0.116	458	243			174	259	0.328	
ry48p.1	0.091	22464	15805			12531	16631	0.247	
ry48p.2	0.103	19697		16074	16666	12883	18078	0.287	
ry48p.3	0.193	25455		19490	19894	13372	20772	0.356	
ry48p.4	0.588	41110	31446			31446	31446	0	22.65
Average Gap								0.217	

Table 4.11. Full results for TSPLIB instances. Solutions times are reported only for those instances that could be solved optimally within the 48-hour time limit.Instances in italic were not closed by our latest published work and are closed in this work.Instances in boldface were open before this work and are closed for the first time in this work

SOPLIB06	Density	Starting	Optimal	Best Known		Proposed B&B			
Instance	of P	Cost	Cost	LB	UB	LB	UB	Gap	Time(s)
R.200.100.1	0.02	269	61			61	61	0	771.36
R.200.100.15	**0.847**	**4070**		**1257**	**1792**	**1792**	**1792**	**0**	**8720.6**
R.200.100.30	0.957	5509	4216			4216	4216	0	27.39
R.200.100.60	0.991	75968	71749			71749	71749	0	0.31
R.200.1000.1	0.02	3603	1404			1392	1409	0.012	
R.200.1000.15	**0.876**	**36427**		**14689**	**20481**	**20481**	**20481**	**0**	**9585.37**
R.200.1000.30	0.958	53838	41196			41196	41196	0	10.04
R.200.1000.60	0.989	82130	71556			71556	71556	0	0.25
R.300.100.1	0.013	363	26			26	26	0	2712.07
R.300.100.15	**0.905**	**5425**		**2166**	**3161**	**3152**	**3152**	**0**	**29583.57**
R.300.100.30	0.97	8389	6120			6120	6120	0	93.2
R.300.100.60	0.994	10875	9726			9726	9726	0	1.13
R.300.1000.1	0.013	4404	1294			1294	1294	0	14050.89
R.300.1000.15	**0.905**	**56770**		**21096**	**29183**	**29006**	**29006**	**0**	**53986.09**
R.300.1000.30	0.965	78690	54147			54147	54147	0	187.11
R.300.1000.60	0.994	121391	109471			109471	109471	0	1.07
R.400.100.1	0.01	285	13			13	14	0.071	
R.400.100.15	**0.927**	**8402**		**2747**	**3906**	**3879**	**3879**	**0.566**	**157991.72**
R.400.100.30	0.978	11280	8165			8165	8165	0	325.65
R.400.100.60	0.996	16688	15228			15228	15228	0	3.19
R.400.1000.1	0.01	4437	1343			1336	1355	0.014	
R.400.1000.15	0.93	81262		28159	29685	22369	46814	0.522	
R.400.1000.30	0.977	118887	85128			85128	85128	0	504.28
R.400.1000.60	0.995	155862	140816			140816	140816	0	3.09
R.500.100.1	0.008	356	4			4	5	0.2	
R.500.100.15	0.945	10130		3543	5361	2821	6856	0.589	
R.500.100.30	0.98	13458	9665			9665	9665	0	1179.44
R.500.100.60	0.996	20445	18240			18240	18240	0	6.91
R.500.1000.1	0.008	5318	1316			1313	1342	0.022	
R.500.1000.15	0.94	95692		32950	50725	27009	62490	0.568	
R.500.1000.30	0.981	139868	98987			98987	98987	0	1319.2
R.500.1000.60	0.996	194354	178212			178212	178212	0	5.96
R.600.100.1	0.007	326	1			1	5	0.8	
R.600.100.15	0.95	11206		3656	5684	2929	7622	0.616	
R.600.100.30	0.985	17427	12465			12465	12465	0	1379.06
R.600.100.60	0.997	25101	23293			23293	23293	0	8.95
R.600.1000.1	0.007	4931	1337			1336	1345	0.007	
R.600.1000.15	0.945	110393		36546	57237	31026	79145	0.608	
R.600.1000.30	*0.984*	*171245*		*118311*	*126798*	*126798*	*126798*	*0*	*2068.37*
R.600.1000.60	0.997	243027	214608			214608	214608	0	11.67
R.700.100.1	0.006	264	1			1	4	0.75	
R.700.100.15	0.957	13478		4494	7311	3753	9944	0.623	
R.700.100.30	*0.987*	*20626*		*14084*	*14510*	*14510*	*14510*	*0*	*1928.38*
R.700.100.60	0.997	26196	24102			24102	24102	0	7.9
R.700.1000.1	0.006	4886	1231			1228	1264	0.028	
R.700.1000.15	0.956	133273		40808	66837	33703	98286	0.657	
R.700.1000.30	*0.986*	*193531*		*121173*	*134474*	*134474*	*134474*	*0*	*1070.1*
R.700.1000.60	0.997	270194	245589			245589	245589	0	6.23
Average Gap								0.122	

Table 4.12. Full results for SOPLIB instances. Solutions times are reported only for those instances that could be solved optimally within the 48-hour time limit. Instances in italic were not closed by our latest published work and are closed in this work.Instances in boldface were open before this work and are closed for the first time in this work

COMPILERS	Density	Starting	Optimal	Best Known		Proposed B&B			
Instance	of P	Cost	Cost	LB	UB	LB	UB	Gap	Time(s)
gsm.153.124	0.97	1201	1109			1109	1109	0	0.17
gsm.444.350	0.99	2814	2745			2745	2745	0	0.27
gsm.462.77	0.84	585	577			577	577	0	12.45
jpeg.1483.25	0.484	101	93			93	93	0	0.24
jpeg.3184.107	0.887	857	791			791	791	0	46.42
jpeg.3195.85	0.74	69	68			68	68	0	18.19
jpeg.3198.93	0.752	340	312			312	312	0	22.85
jpeg.3203.135	0.897	922	850			850	850	0	52.55
jpeg.3740.15	0.257	50	40			40	40	0	0.39
jpeg.4154.36	0.633	171	167			167	167	0	1.38
jpeg.4753.54	0.769	265	245			245	245	0	6.42
susan.248.197	0.939	1396	1338			1338	1338	0	612.98
susan.260.158	0.916	1099	1016			1016	1016	0	39.58
susan.343.182	0.936	1266	1207			1207	1207	0	53.2
typeset.10192.123	0.744	658		565	602	538	612	0.121	
typeset.10835.26	0.349	139	127			127	127	0	5.13
typeset.12395.43	0.518	194	174			174	174	0	40.82
typeset.15087.23	0.557	101	98			98	98	0	0
typeset.15577.36	0.555	175	155			155	155	0	1.89
typeset.16000.68	0.658	97	84			84	84	0	2.03
typeset.1723.25	0.245	73	64			64	64	0	1.06
typeset.19972.246	0.993	2060	2018			2018	2018	0	0.08
typeset.4391.240	0.981	1633	1605			1605	1605	0	23.51
typeset.4597.45	0.493	196	184			184	184	0	24.38
typeset.4724.433	0.995	3507	3466			3466	3466	0	1.85
typeset.5797.33	0.748	135	131			131	131	0	0.03
typeset.5881.246	0.986	1745	1732			1732	1732	0	37.16
Average Gap								0.004	

Table 4.13. Full results for COMPILER instances. Solutions times are reported only for those instances that could be solved optimally within the 48-hour time limit.Instances in italic were not closed by our latest published work and are closed in this work.Instances in boldface were open before this work and are closed for the first time in this work

evaluation shows that both techniques significantly improve the performance of the B&B algorithm. The dynamic-Hungarian-based lower bound speeds up the B&B algorithm by 98% on TSPLIB and SOPLIB, and the local-search domination techniques gives a speedup of 4% on TSPLIB and 13% on SOPLIB. With the application of both techniques, the enhanced algorithm closes five SOPLIB instances that were previously open and significantly reduces the sizes of the optimality gaps on many other instances.

The proposed algorithm still does not perform well on instances with extreme densities. Therefore, yet more powerful pruning techniques are needed. In future work, we will continue to improve the effectiveness of the current pruning techniques and explore new pruning techniques such as integrating lower bounds from a mixed linear programming formulation.

Chapter 5

Conclusions

In this thesis, we studied and developed novel solution approaches for 3 logistic problems, namely the Orienteering Problem with Stochastic Travel and Service Times (OPSTS), the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands (2-stage CVRPSD) and the Sequential Ordering Problem (SOP). Concerning the problems, the main focus and contribution was on the OPSTS. Concerning the methods the main contribution was on sampling-based metaheuristics for Stochastic Combinatorial Optimization Problems (SCOPs) like OPSTS and 2-stage CVRPSD. The metaheuristics developed, had a strong sampling component in their objective functions, beat the state-of-the-art, showed the utility of sampling in solving SCOPs and revealed the disadvantages of sampling when problems have a structure similar to OPSTS. The hybrid methods developed later counteracted the disadvantages of sampling improving further our results. Secondary but not less important were our contributions in our joint work with external collaborators in the 2-stage CVRPSD motivated by the "Green Bullwhip effect" created by companies in Germany. We had similar contributions in the comparison and development of new exact methods for SOP that delivered results beating the state-of-the-art and closing previously unclosed instances in the literature.

In the beginning of studying the OPSTS we examined ways to increase the efficiency of the solution methods proposed in the paper that introduced the problem [Campbell et al., 2011]. The bottleneck of the metaheuristic in that paper as in many SCOPs, was the objective function which was computed using an analytical approximation. The first attempt to accelerate the computation was to use Monte Carlo sampling which has the added advantage that can be modified easily to work with any distribution. It was shown that after some usages of the Monte Carlo Evaluator (MC), it becomes significantly faster than the Analytical

objective function. The effect of number of samples to error and speed of the Monte Carlo Evaluator (MC) was also studied.

The study of MC uncovered cases where it is error-prone. This probed for further research on reducing the error while maintaining the advantages of MC. We named the node where the deadline occurs in the deterministic version of the problem as the *deadline node*. We then observed that small errors in arrival time in nodes neighboring the "deadline node" — that we call the Deadline Area — can result in bigger accumulated error in the final objective value. This happens because errors in arrival time estimation accumulate and an error in deciding whether the node is before or after the deadline occurs in wrongly assigning reward or penalty to that node and probably to other after that node.

These observations led to the development of hybrid methods to compute the objective function. The first approach was to use an Analytical Evaluator to evaluate the nodes belonging to the deadline area and use the Monte Carlo Evaluator (MC) for the rest of the nodes. We called this evaluator MC-Analytical-MC Evaluator or in short M-A-M. We observed experimentally, that using an Analytical Evaluator only for a critical part of the solution -which usually turns out to be small- reduces the error of the evaluator dramatically. This enables us to reduce the number of samples used in the Monte Carlo evaluators, improving the overall speed of the evaluator (apart from very small datasets e.g. 21 customers) and most importantly improving the accuracy of the evaluator dramatically. This in turn yields higher quality results when we embed this evaluator in metaheuristics than the Analytical or MC ones.

The "deadline area" depending on its size affects both the relative error (to the Analytical Evaluator) and the speed gain. As we grow the deadline area, the relative error decreases exponentially (the best model fitted is exponential with very small residual error) while the speed gain decreases only linearly. This enables us to sacrifice a small linear speed gain for an exponential error decrease.

Based on the observation that in the sequence of nodes the ones that are many nodes away before or after the deadline area most likely get a reward or penalty respectively, we developed another evaluator to further increase the performance of M-A-M. The new evaluator defines 2 new ares the Reward and Penalty area which consist of nodes further away from the deadline area. These nodes are assigned a reward or penalty without further computation. So the new hybrid evaluator consists of the Reward-MC-Analytical-MC-Penalty Evaluator and in brief is called R-M-A-M-P. The new evaluator has more areas to tune and if tuned correctly it outperforms most other evaluators, in terms of the speed gains versus the amount of error it yields. The benefits are more noticeable when this evaluator gets embedded in a metaheuristic.

Another evaluator we developed and tested on was the R-A-P or R-A-P which is like the R-M-A-M-P evaluator but without the sampling part of MC evaluators. R-A-P is a completely deterministic evaluator and was initially created to act as a baseline for the usefulness of Monte Carlo sampling in the evaluators. It was found experimentally that it is of practical importance and can outperform other evaluators in the cases where the deadline occurs rarely and a small Analytical part can keep the error very low.

Constructing better ways to approximate the objective function of OPSTS helps us in something more essential which is reaching higher quality solutions in the same time or finding a solution of predefined quality faster. The larger the dataset, the greater the benefit from using the proposed evaluators. For the purposes of our experiments we generated datasets of 1000, 1800, 2500 and 4000 nodes. We performed experiments with the VNS by embedding the Analytical and our most promising evaluator R-M-A-M-P and we measured their performance in terms of the objective value reached for the same amount of runtime on the same system. The hybrid evaluator R-M-A-M-P consistently aids the metaheuristic find a better solution for the same amount of time, in certain cases even 3 times as good.

Apart from the Variable Neighborhood Search Metaheuristic, which was the state of the art at the time, we developed a Random Search Metaheuristic for OPSTS. In almost all our experiments the Random Search Metaheuristic produced statistically significantly higher quality solutions, indicating that either VNS is not suitable for the OPSTS or better neighbourhoods/heuristics need to be used by VNS.

The second problem we studied was the 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands. In the first stage of the problem, the vehicles start from the depot and try to satisfy the base demands of the customers. In the second stage, extra vehicles are sent to the customers whose demands were not previously fully met because of probabilistic increases. In order to solve the problem, we used various objective functions embedding them in an ant colony optimization system. From the experiments, it can be concluded that adopting a 1-stage average case approach leads to shortest overall distances while a 2-stage approach leads to solutions with less vehicles. The conclusions reached by the methods could serve as a decision supporting system for logistics managers in order to reduce the environmental impact of gas emissions and the “green bullwhip effect”.

The third problem studied was the Sequential Ordering Problem where our contribution was in comparing experimetally 2 existing state-of-the-art exact algorithms that were applied in different domains. From our experimental survey,

the advantages and disadvantages of the 2 algorithms were revealed leading to insights for a better exact algorithm. Although it is difficult to predict how successful an algorithm will be on a given instance, it is possible to observe that DEC usually performs well on instances with extreme densities of precedence constraints (either very low or a very high) but does not perform very well on instances with medium densities. On the other hand, B&B appears to be less effective on instances with a low density of precedence constraints but is more effective than DEC on instances with medium densities. It is also interesting to note that B&B performs very well on the COMPILERS instances that are characterized by symmetrical cost graphs (the cost from vertex i to vertex j is always equal to the cost from vertex j to vertex i for all COMPILERS instances). The success of B&B on such instances is attributed to the effectiveness of the history utilization pruning technique on the more symmetrical cases and it was developed with these datasets in mind. We finally took the chance to refine the B&B solver in order to achieve new state-of-the-art results.

At this point let us draw a final conclusion. To sum up, we strongly believe that this work was able to contribute significantly in developing insights and state-of-the-art solution methods to important logistics problems. Our results have indicated new directions for future research. Firstly, our research emphasized the usefulness of hybrid sampling-based metaheuristics for problems with structure similar to the OPSTS such as variants of Stochastic Knapsack. There are many promising directions for future research in generalizing the methods to work with problems in other areas such as scheduling as well as parallelizing them. Another promising direction is creating metaheuristics that would automatically decide the hybrid sampling technique to use and auto-tune it. Secondly, our research in the Sequential Ordering Problem showed directions on how to combine state-of-the-art exact algorithms into a better exact algorithm and obtain better results and find the optimal solution to 9 instances that were open. We believe there are many extremely promising future directions to take in our research in the OPSTS and the SOP that will most probably bear novel important results.

Acronyms

2-stage CVRPSD 2-stage Capacitated Vehicle Routing Problem with Stochastic Demands. 4, 18

ACO Ant Colony Optimization Metaheuristic. 2, 3

Analytical Analytical Evaluator. 21–23, 25, 26, 29–34, 39, 41–47, 50, 54, 57, 59–62, 64–67, 69, 72, 73, 111–113

DA Deadline Area. 32–34, 39, 41–47, 64, 70, 112

IQR Interquartile Range. 62

M-A-M MC-Analytical-MC Evaluator. 32–34, 37, 39, 41–43, 47, 54, 64, 67, 69, 70, 72, 73, 112

MC Monte Carlo Evaluator. 22, 23, 25, 26, 28–34, 37, 39, 41, 43, 44, 47, 50, 54, 64, 67, 69, 70, 72, 73, 111–113

NNH Nearest Neighbor Heuristic. 14, 15, 79

OPSTS Orienteering Problem with Stochastic Travel and Service Times. 2, 4–6, 8–10, 17, 18, 21–23, 26, 55–57, 70, 111, 113, 114

R-A-P Reward-Analytical-Penalty Evaluator. 43, 44, 46, 47, 50, 54, 64, 66, 67, 69, 70, 113

R-M-A-M-P Reward-MC-Analytical-MC-Penalty Evaluator. 43, 44, 47, 54, 58–62, 64, 66, 67, 69, 70, 72–74, 112, 113

RP Reward-Penalty. 64, 65, 73

RS Random Search Metaheuristic. 2, 3, 18, 56, 57, 64–66, 69–72, 74, 113

SCOPs Stochastic Combinatorial Optimization Problems. 1, 21

SEM Standard Error of the Mean. 62

SOP Sequential Ordering Problem. 5, 18

VND Variable Neighborhood Descent. 57, 59

VNS Variable Neighborhood Search Metaheuristic. 2, 3, 18, 41, 46, 57–59, 61, 62, 64, 66–74, 113

Bibliography

Agrawal, S., Sengupta, R. and Shanker, K. [2009]. Impact of information sharing and lead time on bullwhip effect and on-hand inventory, *European Journal of Operations Research* **192 (2)**: 576–593.

Anghinolfi, D., Montemanni, R., Paolucci, M. and Gambardella, L. [2009]. A particle swarm optimization approach for the sequential ordering problem, *Proceedings of the VIII Metaheuristic International Conference (MIC 2009)*.

Anghinolfi, D., Montemanni, R., Paolucci, M. and Gambardella, L. [n.d.]. A hybrid particle swarm optimization approach for the sequential ordering problem.

Arkin, E. M., Mitchell, J. S. B. and Narasimhan, G. [1998]. Resource-constrained geometric network optimization, *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, ACM, New York, NY, USA, pp. 307–316.
URL: *http://doi.acm.org/10.1145/276884.276919*

Aronsson, H., Huge-Brodin, M. and Kohn, C. [2008]. Logistics structures–drivers of environmental impact, *Northern lights in logistics and supply chain management, Gylling* pp. 183–200.

Ascheuer, N. [1996]. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*, PhD thesis.

Ascheuer, N., Escudero, L., Grötschel, M. and Stoer, M. [1993]. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing), *SIAM Journal on Optimization* **3**(1): 25–42.

Balas, E. [1995]. The prize collecting traveling salesman problem: Ii. polyhedral results, *Networks* **25**(4): 199–216.
URL: *http://dx.doi.org/10.1002/net.3230250406*

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

Bertsimas, D. [1992]. A vehicle routing problem with stochastic demand, *Operations Research* **40**(3): 574–585.

Bertsimas, D. and Howell, L. [1993]. Further results on the probabilistic traveling salesman problem, *European Journal of Operational Research* **65**(1): 68–95.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., Mastrolilli, M., Paquete, L., Rossi-Doria, O. and Schiavinotto, T. [2006]. Hybrid metaheuristics for the vehicle routing problem with stochastic demands, *Journal of Mathematical Modelling and Algorithms* **5**(1): 91–110.

Bianchi, L., Dorigo, M., Gambardella, L. and Gutjahr, W. [2009]. A survey on metaheuristics for stochastic combinatorial optimization, *Natural Computing* **8**(2): 239–287.

Blum, C. and Roli, A. [2003]. Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Comput. Surv.* **35**(3): 268–308.
URL: *http://doi.acm.org/10.1145/937503.937505*

Boost [2013]. Lanczos approximation for the gamma distribution. [Online; accessed 17-November-2013].
URL: *http://www.boost.org/doc/libs/1_53_0/libs/math/doc/sf_and_dist/html/math_toolkit/backgrounders/lanczos.html*

Brownlee, J. [2011]. *Clever algorithms: nature-inspired programming recipes*, Jason Brownlee.

Bruzzone, A. and Longo, F. [2014]. An application methodology for logistics and transportation scenarios analysis and comparison within the retail supply chain, *European Journal of Industrial Engineering* **8**(1): 112–142.

Butt, S. E. and Cavalier, T. M. [1994]. A heuristic for the multiple tour maximum collection problem, *Computers & Operations Research* **21**(1): 101–111.

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

Chao, I.-M., Golden, B. and Wasil, E. [1996]. A fast and effective heuristic for the orienteering problem, *European Journal of Operational Research* **88**: 475–489.

Charter, M. and Tischner, U. [2001]. *Sustainable solutions: developing products and services for the future*, Greenleaf publishing.

Chatfield, D., Kim, J., Harrison, T. and Hayya, J. [2004]. The bullwhip effect - impact of stochastic lead time, information quality, and information sharing: A simulation study, *Production and Operations Management* **13 (4)**: 340–353.

Chen, S. and Smith, S. [1996]. Commonality and genetic algorithms, *Technical Report CMU-RI-TR-96-27*, The Robotic Institute, Carnagie Mellon University.

Christofides, N. [1972]. Technical noteâĂŤbounds for the travelling-salesman problem, *Operations Research* **20**(5): 1044–1056.

Cormen, T. H. [2009]. *Introduction to algorithms*, MIT press.

Crisan, E. and M., K. [2012]. A model for business network governance: Case study in the pharmaceutical industry, *World Academy of Science* **Engineering and Technology**: 71.

Denardo, E. and Fox, B. [1979]. Shortest-route methods: 1. reaching, pruning, and buckets, *Operations Research* **27**: 161–186.

Denardo, E. V. [2012]. *Dynamic programming: models and applications*, Courier Corporation.

Dorigo, M. [1992]. *Optimization, Learning and Natural Algorithms (in Italian)*, PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Dorigo, M., Maniezzo, V. and Colorni, A. [1991]. Positive feedback as a search strategy, *Technical report*.

Dunn, W. L. and Shultis, J. K. [2011]. *Exploring Monte Carlo Methods*, Elsevier.

Erera, A. L., Morales, J. C. and Savelsbergh, M. [2010]. The vehicle routing problem with stochastic demand and duration constraints, *Transportation Science* **44**(4): 474–492.

Escudero, L. [1988]. An inexact algorithm for the sequential ordering problem, *European Journal of Operational Research* **37**(2): 236–249.

Escudero, L., Guignard, M. and Malik, K. [1994]. A Lagrangean relax-and-cut approach for the sequential ordering problem with precedence relationships, *Annals of Operations Research* **50**: 219–237.

Feillet, D., Dejax, P. and Gendreau, M. [2005]. Traveling salesman problems with profits, *Transportation Science* **39**(2): 188–205.

Feillet, D., Dejax, P., Gendreau, M. and Gueguen, C. [2004]. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems, *Networks* **44**(3): 216–229.

Forrester, J. [1961]. Industrial dynamics, *Productivity Press, Cambridge, US* .

Gambardella, L. and Dorigo, M. [2000]. An ant colony system hybridized with a new local search for the sequential ordering problem, *INFORMS Journal on Computing* **12**(3): 237–255.

Gambardella, L. and Dorigo, M. T. M. [1999]. Ant colonies for the quadratic assignment problem, *Journal of the operational research society* pp. 167–176.

Gambardella, L., Montemanni, R. and Weyland, D. [2011]. Coupling ant colony systems with strong local searches. Submitted for publication.

Gambardella, L., 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., Laporte, G. and Seguin, R. [1995]. An exact algorithm for the vehicle routing problem with stochastic demands and customers, *Transportation Science* **29**(2): 143.

Gendreau, M., Laporte, G. and Semet, F. [1998]. The selective traveling salesman problem, *Networks* **32**: 263–273.

Gendreau, M. and Potvin, J. [2005]. Metaheuristics in combinatorial optimization, *Annals of Operations Research* **140**(1): 189–213.
URL: *http://dx.doi.org/10.1007/s10479-005-3971-7*

Hernàdvölgyi, I. [2003]. Solving the sequential ordering problem with automatically generated lower bounds, *Proceedings of Operations Research 2003*, pp. 355–362.

Hernàdvölgyi, I. [2004]. *Automatically Generated Lower Bounds for Search*, PhD thesis, University of Ottawa.

İlhan, T., Iravani, S. M. R. and Daskin, M. S. [2008]. The orienteering problem with stochastic profits, *Iie Transactions* **40**: 406–421.

Jaksic, K. and Rusjan, B. [2008]. The effect of replenishment policies on the bullwhip effect - a transfer function approach, *European Journal of Operations Research* **184 (3)**: 946–961.

Jamal, J., Shobaki, G., Papapanagiotou, V., Gambardella, L. M. and Montemanni, R. [2017]. Solving the sequential ordering problem using branch and bound, *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–9.

Johnson, D. and McGeoch, L. [1997]. The traveling salesman problem: A case study in local optimization, *Local Search in Combinatorial Optimization* pp. 215–310.

Johnson, N. L., Kotz, S. and Balakrishnan, N. [2002]. *Continuous Multivariate Distributions, Volume 1, Models and Applications*, Vol. 59, New York: John Wiley & Sons.

Jula, H., Dessouky, M. and Ioannou, P. A. [2006]. Truck route planning in non-stationary stochastic networks with time windows at customer locations, *Intelligent Transportation Systems, IEEE Transactions on* **7**(1): 51–62.

Kao, E. P. [1978]. A preference order dynamic program for a stochastic traveling salesman problem, *Operations Research* **26**(6): 1033–1045.

Kataoka, S. and Morito, S. [1988]. An algorithm for single constraint maximum collection problem., *J. OPER. RES. SOC. JAPAN.* **31**(4): 515–530.

Kenyon, A. S. and Morton, D. P. [2003]. Stochastic vehicle routing with random travel times, *Transportation Science* **37**(1): 69–82.

Klumpp, M. [2011]. Green bullwhip effect simulation concept, *Navais* **P.**: Machado.

Klumpp, M., Clausen, U. and ten M. Hompel [2013]. Logistics research and the logistics world of 2050, *Clausen* **U.**: ten Hompel.

Klumpp, M., Toklu, N., Papapanagiotou, V., Montemanni, R. and Gambardella, L. [2014]. Green bullwhip effect cost simulation in distribution networks, *to appear*.

Kuhn, H. W. [1955]. The hungarian method for the assignment problem, *Naval research logistics quarterly* **2**(1-2): 83–97.

Laporte, G. and Martello, S. [1990]. The selective travelling salesman problem, *Discrete applied mathematics* **26**(2-3): 193–207.

Lee, H. L., Padmanabhan, V. and Whang, S. [1997]. Information distortion in a supply chain: the bullwhip effect, *Management science* **43**(4): 546–558.

Lovasz, L. [1998]. Combinatorial optimization : algorithms and complexity.

Luke, S. [2013]. *Essentials of Metaheuristics*, second edn, Lulu.

Metters, R. [1997]. Quantifying the bullwhip effect in supply chains, *Journal of Operations Management* **15**: 89–100.

Miller, C., Tucker, A. and Zemlin, R. [1960]. Integer programming formulation of travelling salesman problems, *Journal ACM* **3**: 326–329.

Miller, D. L., Pekny, J. and Thompson, G. L. [1991]. An exact two-matching based branch and bound algorithm for the symmetric traveling salesman problem, *Technical report*, DTIC Document.

Mills-Tettey, G. A., Stentz, A. and Dias, M. B. [2007]. The dynamic hungarian algorithm for the assignment problem with changing costs.

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

Mojana, M. [2011]. *Matheuristic techniques for the sequential ordering problem*, Master's thesis, Faculty of Informatics, Università della Svizzera italiana.

Mojana, M., Montemanni, R., Caro, G. D. and Gambardella, L. [to appear]. 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*.

Mojana, M., Montemanni, R., Di Caro, G. and Gambardella, L. [2012]. A branch and bound approach for the sequential ordering problem, *Lecture Notes in Management Science* **4**: 266–273.

Montemanni, R. and Gambardella, L. [2009]. An ant colony system for team orienteering problems with time windows, *Foundations of computing and Decision Sciences* **34**: 287–306.

Montemanni, R., Mojana, M., Caro, G. A. D. and Gambardella, L. M. [2013]. A decomposition-based exact approach for the Sequential Ordering Problem, **5**: 2–13.

Montemanni, R., Smith, D. and Gambardella, L. [2007]. Ant colony systems for large sequential ordering problems, *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, IEEE, pp. 60–67.

Montemanni, R., Smith, D. and Gambardella, L. [2008]. A heuristic manipulation technique for the sequential ordering problem, *Computers and Operations Research* **35**(12): 3931–3944.

Montemanni, R., Smith, D., Rizzoli, A. and Gambardella, L. [2008b]. Sequential ordering problems for crane scheduling in port terminals, *Proceedings of the 11th Intermodal Workshop on Harbor, Maritime and Multimodal Logistic Modeling and Simulation (HMS 2008)*, Campora San Giovanni, Italy, pp. 180–189.

Montemanni, R., Smith, D., Rizzoli, A. and Gambardella, L. [2009]. Sequential ordering problems for crane scheduling in port terminals, *International Journal of Simulation and Process Modelling* **5**(4): 348–361.

Moon, C., Kim, J., Choi, G. and Seo, Y. [2002]. An efficient genetic algorithm for the travelling salesman problem with precedence constraints, *European Journal of Operational Research* **140**(3): 606–617.

Murphy, P. R. and Poist, R. F. [2000]. Green logistics strategies: an analysis of usage patterns, *Transportation Journal* pp. 5–16.

Özelkan, E. and Lim, C. [2008]. Conditions of reverse bullwhip effect in pricing for price-sensitive demand functions, *Annals of Operations Research* **164 (1)**: 221–22.

Papadimitriou, C. H. and Steiglitz, K. [1982]. Combinatorial optimization: algorithms and complexity.

Papapanagiotou, V. [2016]. `https://vassilis.ai/thesis/opsts_data.zip` [Online; accessed 07-Mar-2018].

Papapanagiotou, V., Jamal, J., Montemanni, R., Shobaki, G. and Gambardella, L. M. [2015b]. A comparison of two exact algorithms for the sequential ordering problem, *2015 IEEE Conference on Systems, Process and Control (ICSPC)*, pp. 73–78.

Papapanagiotou, V., Montemanni, R. and Gambardella, L. [2014]. Objective function evaluation methods for the orienteering problem with stochastic travel and service times, *Journal of Applied Operational Research* **6**(1): 16–29.

Papapanagiotou, V., Montemanni, R. and Gambardella, L. [2015a]. Hybrid sampling-based evaluators for the orienteering problem with stochastic travel and service times, *Journal of Traffic and Logistics Engineering Vol* **3**(2).

Papapanagiotou, V., Montemanni, R. and Gambardella, L. M. [2015b]. The orienteering problem with stochastic travel and service times new approaches to sampling-based objective function evaluation, *Annual International Conference on Computational Mathematics, Computational Geometry Statistics*.

Papapanagiotou, V., Montemanni, R. and Gambardella, L. M. [2016a]. Comparison of objective function evaluators for a stochastic orienteering problem, *2016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pp. 465–471.

Papapanagiotou, V., Montemanni, R. and Gambardella, L. M. [2016b]. A sampling-based metaheuristic for the orienteering problem with stochastic travel times, *Proc. 5th International Conference, Theory and Practice of Natural Computing, TPNC 2016, Sendai, Japan*, Lecture Notes in Computer Science, Springer, Berlin, Germany, pp. 97–109.

Papapanagiotou, V., Montemanni, R. and Gambardella, L. M. [2016c]. Sampling-based objective function evaluation techniques for the orienteering problem with stochastic travel and service times, *in* M. Lübbecke, A. Koster, P. Letmathe, R. Madlener, B. Peis and G. Walther (eds), *Operations Research Proceedings 2014*, Springer International Publishing, Cham, pp. 445–450.

Papapanagiotou, V., Weyland, D., Montemanni, R. and Gambardella, L. [2013]. A sampling-based approximation of the objective function of the orienteering problem with stochastic travel and service times, *5th International Conference on Applied Operational Research, Proceedings, Lecture Notes in Management Science*, pp. 143–152.

Pulleyblank, W. and Timlin, M. [1991]. Precedence constrained routing and helicopter scheduling: Heuristic design, *Technical Report RC17154 (#76032)*, IBM T.J. Watson Research Center, Yorktown Heights, New York, USA.

Puterman, M. L. [2014]. *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons.

Russell, R. and Urban, T. [2008]. Vehicle routing with soft time windows and erlang travel times, *Journal of the Operational Research Society* **59**(9): 1220–1228.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H. and Dueck, G. [2000]. Record breaking optimization results using the ruin and recreate principle, *Journal of Computational Physics* **159**(2): 139–171.

Sevkli, Z. and Sevilgen, F. [2006]. Variable neighborhood search for the orienteering problem, *Computer and Information Sciences–ISCIS 2006*, Springer, pp. 134–143.

Shobaki, G. and Jamal, J. [2015]. An exact algorithm for the sequential ordering problem and its application to switching energy minimization in compilers, *Computational Optimization and Applications* **61**(2): 343–372.

Solis, F. and Wets, R.-B. [1981]. Minimization by random search techniques, *Mathematics of Operations Research* **6**(1): 19–30.
URL: *http://dx.doi.org/10.1287/moor.6.1.19*

Spall, J. [2005]. *Introduction to stochastic search and optimization: estimation, simulation, and control*, Vol. 65, John Wiley & Sons.

Sundarakani, B., de Souza, R., Goh, M., Van Over, D., Manikandan, S. and Koh, S. L. [2010]. A sustainable green supply chain for globally integrated networks, *Enterprise Networks and Logistics for Agile Manufacturing*, Springer, pp. 191–206.

Talbi, E. [2009]. *Metaheuristics: from design to implementation*, John Wiley & Sons.

Tan, K., Cheong, C. and Goh, C. [2007]. Solving multiobjective vehicle routing problem with stochastic demand via evolutionary computation, *European Journal of Operational Research* **177**(2): 813 – 839.
URL: *http://www.sciencedirect.com/science/article/pii/S0377221706000208*

Tang, H. and Miller-Hooks, E. [2005]. Algorithms for a stochastic selective travelling salesperson problem, *Journal of The Operational Research Society* **56**: 439–452.

Tanis, E. A. [2008]. *A brief course in mathematical statistics*, Pearson Education India.

Teng, S. Y., Ong, H. L. and Huang, H. C. [2004]. An integer l-shaped algorithm for time-constrained traveling salesman problem with stochastic travel and service times, *Asia-Pacific Journal of Operational Research* **21**(02): 241–257.
URL: *http://www.worldscientific.com/doi/abs/10.1142/S0217595904000229*

Teodorović, D. and Pavković, G. [1992]. A simulated annealing technique approach to the vehicle routing problem in the case of stochastic demand, *Transportation Planning and Technology* **16**(4): 261–273.

Thomadsen, T. and Stidsen, T. K. [2003]. The quadratic selective travelling salesman problem, *Technical report*.

Toklu, N., Papapanagiotou, V., Klumpp, M., Gambardella, L. and Montemanni, R. [2014]. Ant colony optimization for a 2-stage capacitated vehicle routing problem with probabilistic demand increases, *International Journal of Business Innovation and Research* **Special Issue on:** "**Decision-Making Under Uncertainty Models and Approaches**".

Toklu, N., Papapanagiotou, V., Klumpp, M. and Montemanni, R. [2013]. An ant colony approach for a 2-stage vehicle routing problem with probabilistic demand increases, *Proceedings of the Finnish Operations Research Society 40th Anniversary Workshop (FORS40)*, LUT Scientific and Expertise Publications, pp. 5–8.

Toth, P. and Vigo, D. [2001]. *The vehicle routing problem*, Vol. 10.

Tripathi, M., Kuriger, G. et al. [2009]. An ant based simulation optimization for vehicle routing problem with stochastic demands, *Simulation Conference (WSC), Proceedings of the 2009 Winter*, IEEE, pp. 2476–2487.

Tsiligirides, T. [1984]. Heuristic Methods Applied to Orienteering, *Journal of The Operational Research Society* **35**: 797–809.

TSP [n.d.]. TSPLIB: a library of sample instances for the TSP (and related problems) from various sources and of various types, University of Heidelberg, Germany, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

Vansteenwegen, P., Souffriau, W. and Oudheusden, D. [2011]. The orienteering problem: A survey, *European Journal of Operational Research* **209**(1): 1 – 10.
URL: *http://www.sciencedirect.com/science/article/pii/S0377221710002973*

Weyland, D., Bianchi, L. and Gambardella, L. M. [2009]. New approximation-based local search algorithms for the probabilistic traveling salesman problem, *in* R. Moreno-Déz, F. Pichler and A. Quesada-Arencibia (eds), *Computer Aided Systems Theory - EUROCAST 2009*, Vol. 5717 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 681–688.
URL: *http://dx.doi.org/10.1007/978-3-642-04772-5*$_8$8

Weyland, D., Montemanni, R. and Gambardella, L. [2012]. Hardness results for the probabilistic traveling salesman problem with deadlines, *Combinatorial Optimization*, Vol. 7422 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg.

Weyland, D., Montemanni, R. and Gambardella, L. M. [2009]. New heuristics for the probabilistic traveling salesman problem, *Proceedings of the VIII Metaheuristic International Conference (MIC 2009)*.

Weyland, D., Montemanni, R. and Gambardella, L. M. [2013]. Heuristics for the probabilistic traveling salesman problem with deadlines based on quasi-parallel monte carlo sampling, *Computers Operations Research* **40**(7): 1661–1670.

Woeginger, G. J. [2003]. Exact algorithms for np-hard problems: a survey, *Combinatorial optimization - Eureka, you shrink!* pp. 185–207.

Wright, D. and Yuan, X. [2008]. Mitigating the bullwhip effect by ordering policies and forecasting methods, *International Journal of Production Economics* **113**(2): 587–597.

Index