Advanced Metaheuristics for the Probabilistic Orienteering Problem

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
Xiaochen Chou

under the supervision of
Prof. Luca Maria Gambardella
co-supervised by
Prof. Roberto Montemanni

October 2020

Dissertation Committee

Prof. Evanthia Papadopoulou	Università della Svizzera italiana, Switzerland
Prof. Olaf Schenk	Università della Svizzera italiana, Switzerland
Prof. Philipp Baumann	Universität Bern, Switzerland
Prof. Carlo Filippi	Università degli Studi di Brescia, Italy

Dissertation accepted on October 2020

Prof. Luca Maria Gambardella
Research Advisor
Università della Svizzera italiana, Switzerland

Prof. Roberto Montemanni
Research Co-Advisor
University of Modena and Reggio Emilia, Italy

Prof. Binder Walter
PhD Program Director

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.

Xiaochen Chou
Lugano, October 2020

To my beloved

Abstract

Stochastic Optimization Problems take uncertainty into account. For this reason they are in general more realistic than deterministic ones, meanwhile, more difficult to solve. The challenge is both on modelling and computation aspects: exact methods usually work only for small instances, besides, there are several problems with no closed-form expression or hard-to-compute objective functions. A state-of-the-art approach for several stochastic/probabilistic vehicle routing problems is to approximate their cost using Monte Carlo sampling.

The Orienteering Problem is a routing problem aiming at selecting a subset of a given set of customers to be visited within a given time budget, so that a total revenue is maximized. Multiple stochastic variants of the problem have been studied. The Probabilistic Orienteering Problem is one of these variants, where customers will require a visit according to a certain given probability. The objective is to select a subset of customers to visit within a given time budget, so that an expected total reward is maximized while the expected travel time is minimised. The problem is NP-hard. In this work we propose different metaheuristics based on hybrid Monte Carlo sampling approximation to solve the problem. Detailed computational studies are presented, with the aim of studying the performance of the metaheuristics in terms of precision and speed, while positioning the new method within the existing literature.

In this work, we also study the use of Machine Learning tools to help solve optimization problems. By shifting the problem of selecting the number of samples used by the Monte Carlo approximation to that of choosing a trade off between speed and precision, the best number of samples can be predicted by using Machine Learning models in a fast and efficient way.

The Tourist Trip Design Problem (TTDP) is a variant of a route-planning problem for tourists interested in visiting multiple points of interest. A practical application of the POP to the probabilistic version of the TTDP is also discussed, and this provides inspiration for more possible applications.

Acknowledgements

I am deeply grateful to my Research Advisor Prof. Luca Maria Gambardella, Research Co-Advisor Prof. Roberto Montemanni and Academic Advisor Prof. Jürgen Schmidhuber, who generously rendered help and encouragement during the process of this thesis. Whatever I have accomplished in pursuing this undertaking is due to their guidance and thoughtful advice.

I would also like to thank my dissertation committee (in alphabetical order): Prof. Carlo Filippi from Università degli Studi di Brescia, Prof. Evanthia Papadopoulou from Università Della Svizzera Italiana (USI), Prof. Olaf Schenk from Università Della Svizzera Italiana (USI) and Prof. Philipp Baumann from Universität Bern. All of them have already provided valuable feedback, ideas and advice related to my research.

At this point, I would like to thank everyone who helped me through my doctoral studies culminating and contributed to my research in one way or another (again in alphabetical order): Aleksandar Stanic, Arpitha Bharathi, Bingrong Chen, Francesco Faccio, François Févotte, Hui Yan, Krsto Prorokovic, Mengke Ren, Micheal Wand, Murodzhon Akhmedov, Paulo Rauber, Ricardo Omar Chavez-Garcia, Robert Csordas, Thi Viet Ly Nguyen, Vasileios Papapanagiotou and Yingjie Li.

Last but not least I would like to thank my family. Thank you all for your care, support and encouragement all the time. Thank you for the wise counsel and sympathetic ear, always be there for me.

Contents

Figures

Tables

List of Algorithms

Chapter 1

Introduction

In this chapter, the motivation of the research and the contributions of our study are introduced, along with the structure of the thesis. A literature review on the research areas relevant to the thesis is also presented.

1.1 Research Motivation

Stochastic Combinatorial Optimization Problems have drawn the attention of many researchers in recent years. With respect to classic optimization approaches, uncertainty factors are taken into account, thus the resulting models are more realistic. However, stochastic problems are also more difficult to solve. The challenge is both on modelling and computation aspects: exact methods usually work only for small instances, besides, there are several problems with no closed-form expression or hard-to-compute objective functions. Designing efficient (meta)heuristics to produce high quality solutions in a reasonable amount of time is an actual research goal.

Metaheuristics based on (hybrid) Monte Carlo sampling have become state-of-the-art approaches for several stochastic/probabilistic vehicle routing problems, such as the Probabilistic Traveling Salesman Problem (PTSP), the Probabilistic Traveling Salesman Problem with Deadlines (PTSPD) and the Orienteering Problem with Stochastic Travel and Service Times (OPSTS).

Given a problem with stochastic input data, non-stochastic constraints and an objective function which is the expectation of a certain random variable, the Monte Carlo sampling method is able to deal with different scenarios regarding the complexity of the objective function. For problems either with or without a close-form expression available for the objective function, the Monte Carlo sampling method can approximate the objective function in a fast and efficient way. Even for problems with objective functions that have already been computed efficiently, the Monte Carlo sampling method is still a strong competitor in speed and efficiency. This makes the Monte Carlo Approximation an effective evaluator to be used inside heuristic methods. In this work, we approximate the objective function of the Probabilistic Orienteering Problem by using the Monte Carlo sampling method and solve it with metaheuristic algorithms.

The Probabilistic Orienteering Problem (POP) was first introduced in Angelelli et al. [1]. It is a variant of the Orienteering Problem (OP) [63] where customers are requiring a visit with certain probabilities. In the OP, there is a set of customers at different locations and with

associated prizes. A prize is collected only if the respective customer is visited. The aim is to maximize the total prize collected within a given time budget. In the POP, the potential customers will require a visit according to given probabilities, correspondingly, a prize can be collected only when a customer actually requiring a visit is served before the given deadline. Thus, the target of the POP is to construct a planning that consists in selecting a subset of the given customers, in such a way that the respective *expected* prize collected is maximized and the *expected* total travel time cost is minimized.

Given a tour, the analytical approximation of the expected total travel time cost requires cumulative products considering all combinations in which each customer requires a visit or not. Given a feasible tour, the objective function value can be evaluated in quadratic time. However, optimizing the feasible tour is a complex task. The MILP model presented in [1] based on feasible solutions has an exponential number of constraints. The Monte Carlo sampling method is also able to approximate the objective function in polynomial time. By using it as an interrelated part of a simple local search heuristic, the POP can be solved. However, the searching process may get stuck in local extrema. We proposed a Random Restart Local Search algorithm and a Tabu Search algorithm to solve the POP. Both methods are designed to escape from local extrema and they both embed a Monte Carlo evaluator. An innovative integration of the local search heuristics and the Monte Carlo sampling method allows the creation of simple and effective algorithms. With this technique, we provide solutions for the POP with low error in a very short time.

One crucial parameter in the Monte Carlo sampling evaluator is the number of samples to be used. More samples means precision, while less samples means speed. An instance-dependent trade-off has to be found. Machine Learning techniques have been recently used in Optimization contexts. Apart from the classic experimental tests, we are also interested in understanding whether a Machine Learning model can effectively predict the number of samples required for an instance or not. Two methods are presented and compared from an experimental point of view. It is shown that a less intuitive and slightly more complex method is able to provide more precise estimations.

Finally, we are also interested in the possible application of the POP. The Touristic Trip Design Problem (TTDP) is a variant of a classic route-planning problem for tourists interested in visiting multiple points of interest. A simple formulation of the TTDP is proven to be identical with the OP. Adhering to the practicality, we also take uncertainties into consideration for the TTDP. In fact, a variety of uncertainties could affect the availability of the points of interest. For example, popular places may probabilistically require a long waiting time and open-air places may not be visitable due to unseasonable weather. In this work, we define the Probabilistic Tourist Trip Design Problem (PTTDP) and show that simple POP solvers can efficiently provide solutions for this application.

1.2 Structure of the Thesis

The structure of the document is as follows.

A review of the literature focusing on the POP and on common methods used for several related stochastic combinatorial problems is presented in Section 1.3. In Chapter 2, a detailed formal definition of the problem studied is provided. A Monte Carlo evaluator is presented in

Chapter 3. The Monte Carlo evaluator is then embedded in a Random Restart Local Search algorithm presented in Chapter 4 to solve the POP. In Chapter 5, we describe a Tabu Search algorithm based on the Monte Carlo evaluator as well. Experimental tests on the performance of the metaheuristic algorithms proposed are carried out in both Chapters, with an analysis of limitation and advantages of the approach with respect to other methods from the literature. In Chapter 6 we discuss the use of Machine Learning technologies for parameter tuning purposes. In Chapter 7 a stochastic Tourist Trip Design Problem with personalized customisation is presented and solved by using the methods proposed in the previous chapters. Finally, conclusions are drawn in Chapter 8.

1.3 Literature Review

In this section, we give an overview about publications regarding stochastic combinatorial problems related to our research, technical aspects of the Monte Carlo sampling methods, the metaheuristics using the Monte Carlo sampling approximation of the objective function, and the use of Machine Learning techniques in Optimization contexts.

1.3.1 Stochastic Combinatorial Optimization Problems

The Probabilistic Orienteering Problem (POP) was first introduced by Angelelli et al. [1]. It is a variant of the Orienteering Problem (OP) where customers are requiring a visit with certain probabilities. Detailed information of the OP along with survey can by found in Vansteenwegen et al. [63] and Gunawan et al. [27].

As shown in Feillet et al. [5] and Bérubé et al. [21], the OP can be formulated as a Travelling Salesman Problem with Profits. The problem has been proven to be NP-hard by Golden et al. [26]. To solve the OP, a brand-and-bound method is used by Laporte and Matello [38] and Ramesh et al. [50]. Improved approaches based on branch-and-cut have been proposed by Fischetti et al. [22] and Gendreau et al. [24]. There have been continuous improvements to the solving methods over the years, and a classification of the exact algorithms and details about the applied techniques can be found in Feillet et al. [21]. The OP arises in several real domains. A recent application is the Mobile Tourist Guide introduced in Souffri et al. [56], which makes feasible plan for tourists to visit the most valuable attractions among all the places that they wish to visit in the limited time of the trip. Such planning problems are called Tourist Trip Design Problems (TTDP) (Vansteenwegen and Van Oudheusden [60]). Similar Tourist Trip problems of selecting the most interesting combination of attractions is mentioned in Chou et al. [18] and Wang et al. [66]. In general, the application of the TTDP requires high quality solutions in a short calculation time, which helps to make real-time decisions.

There are several deterministic extensions to the OP. The Team Orienteering Problem (TOP) [12, 59, 3] considers multiple routes limited by the deadline and the goal is to maximise the total prize collected. The Orienteering Problem with Time Windows (OPTW) and the Team Orienteering Problem with Time Windows (TOPTW) add time window constraints for the customers, and a visit to a customer can only starts during this time window. The OPTW and TOPTW have been solved with Ant Colony optimization by Montemanni and Gambardella [46, 47], and Iterated Local Search metaheuristic by Vansteenwegen et al. [62]. Another interesting variation of the OP is the Clustered Orienteering Problem (COP) proposed by Angelelli et al. [2], where

each profit is associated with a cluster of customers and is gained only when all customers in the cluster are served.

The probabilistic variations of the OP are mainly considering uncertainty on time and/or profit. From the objective perspective, the POP we study in this work is similar to the Probabilistic Travelling Salesman Problem with deadline (PTSPD) [11]. In the PTSPD, a minimum expected cost a priori tour is to be find through a set of customers with probabilities of requiring service. Each customer has a deadline, and service at each customer should begin at or before its deadline. The objective is to minimize the expected travel time and the expected penalty caused by violation of the deadline. In Campbell et al. [11], different models are considered such as paying a penalty for any violation of a deadline, or paying penalties for skipping the customers whose deadline would be violated. In both the POP and the PTSPD, the stochasticity is on the probability of a customer requiring service. The main difference is that in a POP feasible solution every customer on the tour must be visited before the global deadline, while in PTSPD customers can be visited after the deadline, paying a penalty. Other variants of the OP with stochasticity on service or travel time have been studied in Tang and Miller-Hooks [58] and Campbell et al. [10]. A vehicle routing problem with stochastic travel times and soft time windows is considered in Russel and Urban [52]. An OP with uncertain prizes has been considered in Ilhan et al. [33]. From an application perspective, a tourism-related practical application of the POP has been studied in [18].

The basic models can be extended with extra feature. For example, a Probabilistic Team Orienteering Problem (PTOP) based on the Team Orienteering Problem (TOP) is discussed in Chao et al. [12] and Tang and Miller-Hooks [59]. The PTOP can be open to more applications, such as planning a different tour for each day during the length of the stay, or determine same or separate tours for a tour group, regarding the different satisfaction scores given by each member in the group and the availability of the points of interests on the list. A much further probabilistic extension of the problem can also be based on the variations of the OP, such as OPTW and TOPTW.

The POP inherits the NP-hardness from the OP [1]. To solve the POP, a MILP model has been proposed and the problem is solved by branch-and-cut methods in Angelelli et al. [1]. Since solving with a basic branch-and-cut method is computationally demanding, three matheurstic methods to reduce the solution time (detailed in Section 3.1.2) have also been also studied in [1]. The use of Monte Carlo sampling techniques in the context of the POP has been studied in Chou et al. [14]. A fast objective function cost approximator for the problem using Monte Carlo sampling method has been proposed. Similar uses of the Monte Carlo sampling approach for stochastic/probabilistic vehicle routing problems can be found in Weyland et al. [67] and Papapanagiotou et al. [48]. Given the characteristics of an instance, a Machine Learning-based study on estimating the best number of samples to use in such a Monte Carlo sampling evaluator for the POP can be found in Montemanni et al. [45].

Local search methods provide fast and efficient heuristic solutions for problems with high computational complexity [30]. For example, the 2-opt heuristic ([19, 23, 40, 34]) is proven to be crucial to obtain high quality results for the TSP and OP. However, a standard technique like 2-opt and 3-opt [6] which only searches in the neighborhood of the solution may get trapped in local extrema [54]. To take care of this undesired phenomenon, random restart strategies applied on top of the local search adopted are normally used, in order to increase the probability of success in searching procedures. The method works by restarting the optimization search

once no further improvement is possible by the embedded local search component. In our work, a solver with heuristic speedup criterion for the POP (details in Chapter 4) using Random Restart mechanism is proposed, which proved to be fast and efficient for small instances problems. A study on improving the performance of the solver by modifying the generation of re-initialising solutions is also presented in [17].

Tabu Search (TS) is a metaheuristic search technique that has been widely used in combinatorial optimization problems since it was introduced by Glover [25]. It is another effective method to deal with local extrema. The basic idea is to accept a sequence of non-improving solutions to escape from local extrema. This is achieved by the use of an explicit memory called Tabu List during the search process. It avoids visiting solutions that have been visited recently. A variety of implementations of Tabu Search in the context of the Travelling Salesman Problem and related problems is reviewed and compared in Basu [4]. Other methods have been proposed, such as Simulated Annealing in Kirkpatrick et al. [35], Genetic Algorithms in Holland [29] and Ant Colony Optimization in Dorigo and Gambardella [20]. They are other techniques frequently used to escape from local extrema. The Tabu Search method implemented by Kulteral-Konak et al. [37] has been compared with a Multi-Level Variable Neighbourhood Search (MLVNS) algorithm on solving large OP instances in Liang et al. [39]. In the MLVNS, instances with identical settings (i.e. dimension, coordinates, and associated scores of the nodes) differing only in the constraint levels will be solved concurrently. In this way, information from the searching process can be shared among the instances to improve the efficiency. Experimental results show that the MLVNS performances better on the quality of the solutions, while the TS requires shorter computing time.

1.3.2 Machine Learning techniques in Optimization

Machine Learning techniques have been recently used in Optimization domain. Examples of such methods can be found in the context of branch and bound solvers for Mixed Integer Programming: the branching strategy has a key role in the performance of methods. In Hutter et al. [31] some methods are devised to learn which is the best strategy. The idea is to identify some features of the present sub-problems or the present search-tree node, based on which the branching strategy to employ is selected, in order to mimic in a fast way the so-called strong branching that guarantees the best performance, but is time-consuming [42]. The training phase of the neural network can either be done before or during the run of the branch-and-bound algorithm. A successful combination of Monte Carlo Tree Search and Deep Learning can be found in Silver et al. [55] and Parascandolo et al. [49].

Another trend is to use Machine Learning techniques to tune the parameters of the solving methods such as in Calvet et al. [9], Hutter et al. [32], and Lodi et Zarpellon [41]. Such methods again rely on the identification of some features and as it was happening with the previous application the identification of such features is very crucial for the performance of the methods. A similar application, based on a similar concept of features identification and evaluation, is to adopt Machine Learning techniques to forecast the running time of an algorithm ([32]) or classifying quadratic programming problems in terms of the best algorithm to use to solve them [41]. The use of Artificial Neural Networks in Optimization has been considered in Villarrubia et al. [65]. Another Machine Learning-based study on Reinforcement Learning for the Travelling Salesman Problem is discussed in Mele et al. [44].

Chapter 2

The Probabilistic Orienteering Problem

2.1 Problem Definition

The Probabilistic Orienteering Problem answers the following question: given a time budget and a list of customers at different locations with different probabilities of requiring a visit and deterministic prizes to be collected, what is the best subset of customers to visit (and in which order) so that the expected total collected prize is maximized and the travel time is concurrently minimized? This problem, faced by companies while organising their day-ahead plan or general users having to take real time decisions [18], is a variation of the Orienteering Problem which takes uncertainties of the customers' requirements into consideration, with uncertainty modelled through probabilities.

As described in [63], the Orienteering Problem can be defined on a graph $G = (V,A)$ where $V = \{0, 1...n, n+1\}$ is the node set representing customers with the depot being node 0 and the destination being node $n+1$ and A is the arc set representing the paths between pairs of customers. A prize p_i is associated with each node, a travel time t_{ij} is associated with each arc $a_{ij} \in A$ and a deadline D is finally provided. A feasible tour σ is defined as a sequence of customers that can be visited before the deadline incurs. With this notation, the OP can be described as the problem of finding the tour that collect the largest possible amount of prizes from customers within the given deadline [63]. This of course implies a selection of the shortest tour among the selected customers. The objective function of the OP can be reduced therefore to the maximization of the following quantity $o(\sigma)$, given a feasible (partial) tour σ not exceeding the deadline D:

$$o(\sigma) = \sum_{i \in \sigma} p_i \tag{2.1}$$

Differently from the OP, the objective function of the POP is no longer a simple sum. Due to the probabilistic nature of the problem, customers that offers a large prize may require a visit with a very low probability. With this consideration, the POP is defined as follows.

A POP instance contains the information (V, t, π, D, p) where:

- $V = \{0, 1...n, n+1\}$ is the set of n customers (nodes) with the depot being node 0 and the destination being node $n+1$.
- D is the global deadline.

- t_{ij} is the deterministic travelling time from customer i to customer j defined, for all $i, j \in V$.
- π_i is the probability of a customer $i \in V$ to require a visit and is modeled in our study by a Bernoulli variable $b_i = \{0, 1\}$. b_i takes value 1 with an independent probability π_i for each customer. Depot and destination are required to be visited by definition, therefore $\pi_o = \pi_{n+1} = 1$.
- p_i is the prize collected when a customer $i \in V$ is served before the global deadline D. There is no prize for depot and destination, therefore $p_0 = p_{n+1} = 0$.

Formally, we define a *feasible solution* $\sigma = (i_0 = 0, i_1, i_2, ..., i_q, i_{q+1} = n+1)$ as a tour starting from the depot, allowing to serve a sequence of q customers even when all customers on the route require visits and going to the destination before the global deadline D. A *complete tour* $\tau = (0 = i_0, i_1, i_2, ..., i_n, i_{n+1} = n+1)$ is defined as a sequence of n customers, plus the depot and the destination node. For a given complete tour τ, the associated prize is $P(\tau)$ and the associated travel time is $T(\tau)$.

The POP is a combinatorial optimization problem with the objective of maximizing the total expected prizes collected while visiting customers before the given deadline, minus a measure of the total expected travel time. The last factor helps to make a choice when several tours are associated with a same total prize, and in this case we pick the one associated with smaller travel cost. Note that the travel time component is not present in the classic formulation of the OP described above.

The objective function of the POP is to retrieve a feasible solution σ among all possible tours that maximize the following quantity:

$$u(\sigma) = E[P(\sigma)] - CE[T(\sigma)] \tag{2.2}$$

where the difference between the expected total prize and the expected total travel time is multiplied by a given coefficient C that is normally set by the decision maker [1].

From a computational perspective, solving the POP is computationally demanding. Given a feasible tour, the objective function value can be evaluated in quadratic time by using the following formulas ([1]):

$$E[P(\sigma)] = \sum_{k=1}^{q} \pi_k p_k \tag{2.3}$$

$$E[T(\sigma)] = \sum_{h=0}^{q} [\pi_h \sum_{k=h+1}^{q+1} (t_{hk} \cdot \pi_k \cdot \prod_{j=h+1}^{k-1} (1 - \pi_j))] \tag{2.4}$$

where it is assumed that $\prod_{j=h+1}^{k-1}(1-\pi_j) = 1$ whenever $h+1 > k-1$.

However, optimizing the feasible tour is a complex task. The MILP model presented in [1] based on feasible solutions has an exponential number of constraints.

The main purpose of our study is to design simple heuristics to solve the POP in a simple and effective fashion. To achieve this, we think from another perspective to work with complete tours directly generated from the POP instances. The best complete tour can be easily obtained

by local search heuristics, and the best feasible solution can be extracted from the best complete tour. In this way, the bottlenecks are transferred to the evaluating part in our approach. We design a smart Monte Carlo evaluator. It not only evaluates the objective function value of a given complete tour, but also extracts the best feasible solution out of the given tour with the maximum profit, which is consistent with the objective function, during the evaluation. In this way, when two complete tours are associated with a same feasible solution, the evaluator is be able to tell the difference within the local search procedure, and picks the one that is more likely to lead to a better searching space and eventually improves the accuracy of the solving method. The smart Monte Carlo evaluator is described in the following Chapter 3, and it is used as an interrelated part of heuristics we propose (see Chapter 4 and Chapter 5)) to solve the POP eventually.

Chapter 3

Monte Carlo sampling for the Probabilistic Orienteering Problem

3.1 A Monte Carlo sampling approach

In this section we propose a way of evaluating the objective function value of the Probabilistic Orienteering Problem described in Chapter 2 by using a Monte Carlo sampling method. The work presented has appeared in [14].

3.1.1 Methodology

The objective function value is approximated by the Monte Carlo sampling method (MC) in the following way: for a given complete tour τ with n customers, according to the probability of each customer to require a visit, a set of s deterministic scenarios is generated. Each scenario is a fully connected graph with a fixed set of customers requiring a visit and implies therefore a tour τ_J obtained by τ by cancelling the customers not present in scenario J. For each scenario $J \in \{1 \dots s\}$, a deterministic objective function value can be evaluated in linear time as $u_d(\tau_J) = P(\tau_J) - C \cdot T(\tau_J)$, thus the objective function value $u(\tau)$ of the tour τ can be approximated by averaging these values in time $O(ns)$.

Figure 3.1 shows an example of generating s scenarios for a given complete tour $\tau = (0, 1, 2, 3, 4, 5, 6)$, with 0 being the depot and 6 being the destination. Each block represents a scenario and each dot in the block represents a node, which can be either a customer or the depot/destination. In each scenario, whether a customer requires a visit (yellow) or not (grey) is deterministic. When a customer i does not require a visit, he will be skipped while travelling on the tour τ, and the truck moves from the previous customer directly to the next customer that requires a visit. For example in scenario 1, customer 1 and 4 do not require a visit, therefore, a tour $\tau_1 = (0, 2, 3, 5, 6)$ is obtained, and straightforwardly, its objective function value $u_d(\tau_1)$ can be calculated. Similarly, a total of s scenarios are generated regarding the probability of each customer in tour τ requires a visit. With $u_d(\tau_J)$ being the deterministic cost of the tour τ_J adapted to the deterministic scenario J, the approximation value for $u(\tau)$ is given by

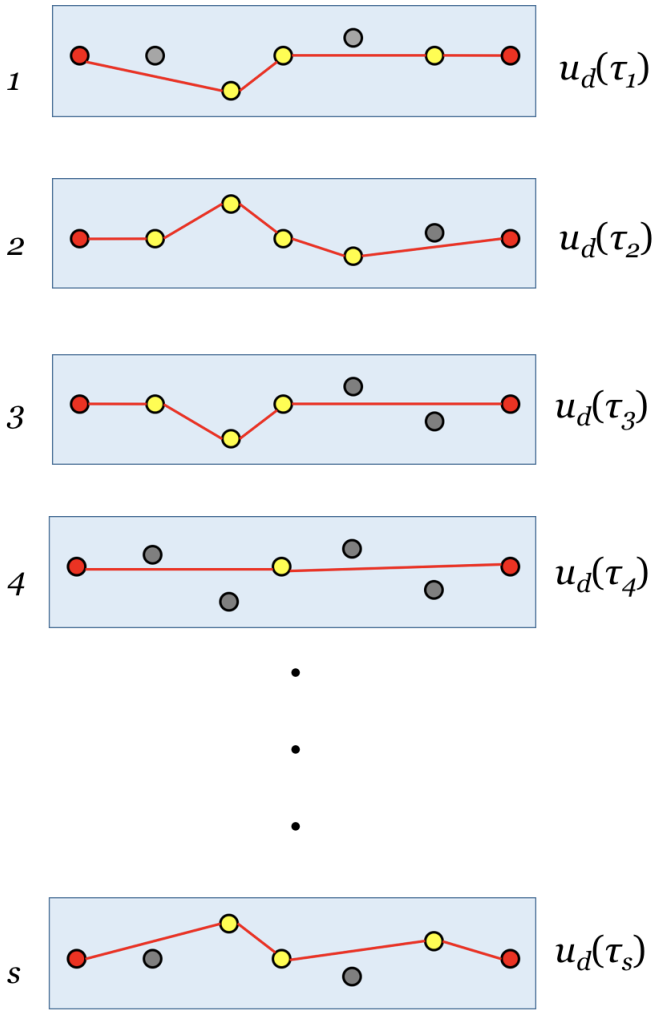

Figure 3.1. Example of s scenarios generated from a tour τ

the average objective function value of the deterministic costs of all the scenarios considered:

$$u(\tau) \approx \frac{\sum_{J=1}^{s} u_d(\tau_J)}{s} \tag{3.1}$$

By design, our solver treats complete tours touching all customers, while a feasible solution only visits a subset of customers that can be served before the deadline D. Thus the Monte Carlo evaluator is also delegated to extract a feasible solution $\sigma(\tau)$ as a prefix of a complete tour τ.

By the definition of the problem, the travel time of going from the last visited customer back to the depot has always to be considered. Due to this feature, when a complete tour $\tau = (i_0 = 0, i_1, i_2, ..., i_n, i_{n+1} = n + 1)$ is being evaluated over a given deterministic scenario s,

the tour stops at the last customer i_q ($q \leq n$) before the deadline D is incurred, and then goes directly to the destination node. The best feasible POP solution selected out of a complete tour can also be easily obtained in this way by supposing all customers on the route require visits. However in certain situation, it is possible that some customers visited within the deadline are not offering enough prize to make an increase on profit. Therefore, to make a smarter selection that is consistent with the objective function, we take the best feasible solution $\sigma(\tau)$ selected out of a complete tour τ as the one with the peak value of (3.1) encountered during the evaluation of τ, denoted as $u(\sigma(\tau))$. This coincide with the fact that during the evaluation of a complete tour τ, after the deadline is incurred only the travel time component increases, but no more prize is collected.

Experimental examples on customer selection regarding this peak value during evaluation can be found in Section 3.1.3. A detailed design choice for the heuristic speed-up criterion with parameters tuning is presented in Section 3.2, where the Monte Carlo evaluator is able to understand how to cut the best feasible solutions from given complete tours, and stops evaluating at proper time to improve the speed of the evaluation while not losing precision.

In conclusion, the Monte Carlo Approximation component receives in input a (complete) tour τ and outputs:

- a POP feasible solution $\sigma(\tau)$ extracted from the complete tour τ;
- an approximation for the objective function value $u(\tau)$ of the complete tour τ;
- an approximation for the objective function value $u(\sigma(\tau))$ of the POP solution $\sigma(\tau)$.

Due to the heuristic nature of the approach, the approximation of the objective function value is more accurate when more scenarios are used, but this requires more computational time. In the following subsections, experiments are carried out to understand which could be promising values for the number of samples to be used for evaluation.

3.1.2 Experimental data sets

The POP benchmark instances used for the main experiments in this work are the 264 instances introduced in [1][1]. They are based on 22 TSP benchmark instances from the TSPLIB95 library[2] [51]. With reference to the definition in Section 2.1, the characteristics of the POP instances are set as follows:

- The customers information and distances are taken directly from the corresponding original TSP instances. The first customer of the original instance is considered as the depot. The destination vertex coincides with the depot.
- The global Deadline D takes value of $\omega \cdot T_{max}$, with T_{max} being the known optimal value of the original TSP instance and $\omega = \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$ representing short, medium and long deadlines.
- The probabilities of the customers to require visits are either $\pi_i = 0.5$ for all customers, or π_i is a random number in the interval $[0.25, 0.75]$ for each customer i.
- The prizes collected while visiting the customers are either $p_i = 1$ for all customers, or p_i is a pseudo-randomly generated integer in $\{1, 2, ..., 100\}$ for each customer i.

[1] The POP instances are available at http://or-brescia.unibs.it/instances
[2] The TSPLIB library can be found at http://iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

The coefficient C of the objective function (2.2) used to balance between the travel times and the prizes components is taken as $C = 0.001$, in order to maintain full compatibility and comparability with the results of [1], where the same setting is considered.

The testing environment is a computer equipped with a Quad-Core Intel Core i7 processor 2.0 GHz (only one core is used for the experiment) and 8GB of RAM. The Monte Carlo evaluator was implemented in C++. These settings are used for all the experiments reported in this Chapter.

3.1.3 Customers selection

As mentioned in Section 3.1.1, the Monte Carlo evaluator is able to extract the best feasible solution out of a given complete tour during evaluation. Instead of using simple strategies such as calculating the average of maximum number of customers served before the deadline, we calculate the value of objective function for each node with all samples, supposing that it is the last customer visited. In such a way, by arriving at each customer on the tour, there is an objective function value of the corresponding feasible solution. The peak value will show the best feasible solution that can be extracted from the given complete tour. The approach is heuristic but it provides good results in practice.

First, we test the idea on two instances with the same location information and prize values, but different probability types of requiring a visit. In this two instances, the prize $p_i = 1, \forall i \in V$, and the probability π_i of requiring a visit is either fixed with value 0.5 or random. The results are plotted in Figure 3.2 and Figure 3.3, where the x-axis shows all nodes $\{0, 1...n\}$ in the order of a given solution, and the y-axis shows the approximated value for the objective function at each node, as described in Section 3.1.1.

From the results in Figure 3.2 and Figure 3.3 we can observe that, given two instances with the same location information and prize values, the slopes of the evolution of the objective function value are similar for both instances with random and fixed probability of requiring a visit. In both cases there exists a significant maximum at node 14 showing the best node to stop serving customers.

Then we try on two instances still with the same location values, but with random prize p_i for each customer. The probability π_i of requiring a visit is still either fixed with value 0.5 or random. The results are plotted in Figure 3.4 and Figure 3.5. This time total prize collected at local maximum has much higher value than in Figure 3.2 and 3.3, but the travel time after the local maximum is increasing as in the previous experiments. Therefore the curve is almost flat after the local maximum. A maximum can however be observed, showing again the best stopping point.

By inspecting the curves, the best feasible solution of a given complete tour τ can be extracted during the evaluation process. Therefore the idea provides an heuristic idea to perform the evaluation only on the subset of customers of the best feasible solution. This will speed up the evaluation and lead to more accurate results within appropriately designed heuristic solvers, since an external tool to select the customers to visit is not required. Further discussion on how to detect the peak value without evaluating until the end of the tour can be found in Section 3.2.

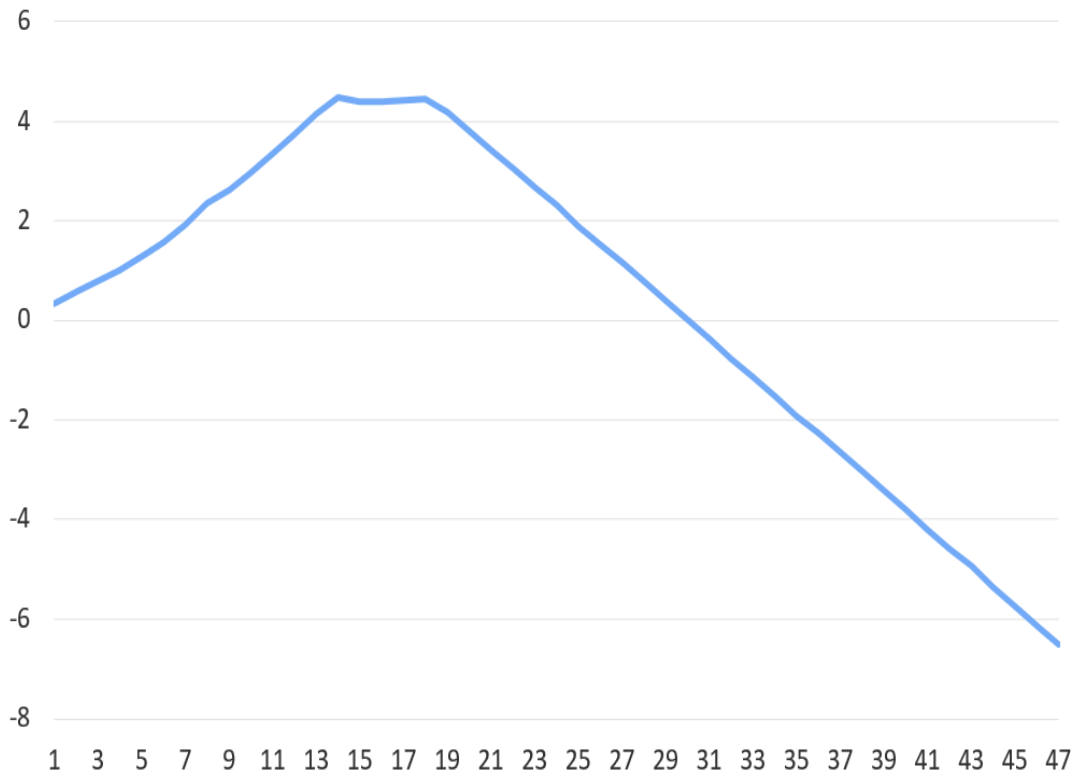

Figure 3.2. $u(\tau)$ of an example instance with $p_i = 1, \pi_i = 0.5$

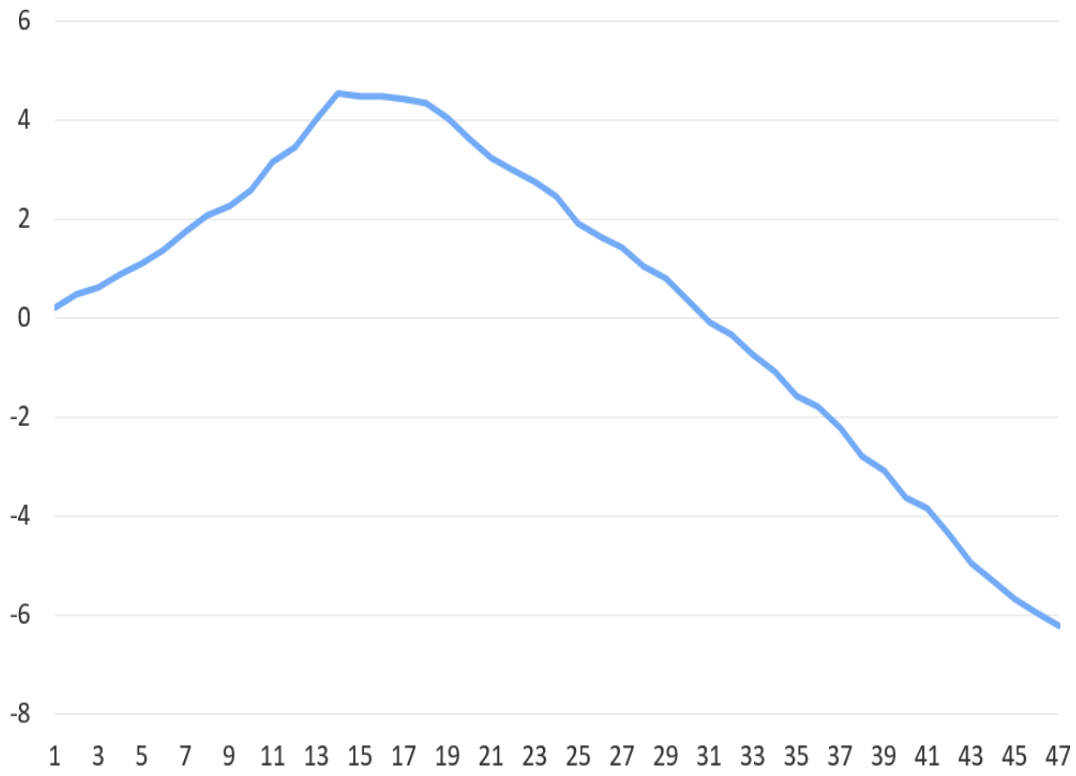

Figure 3.3. $u(\tau)$ of an example instance with $p_i = 1, \pi_i$ random

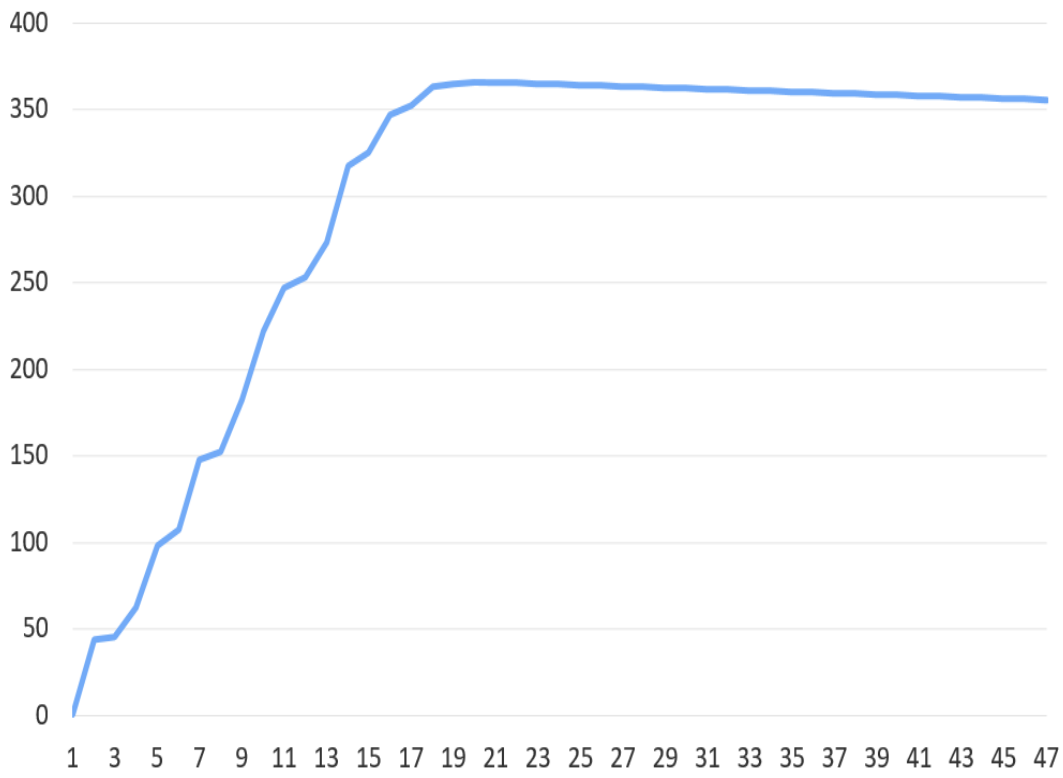

Figure 3.4. $u(\tau)$ of an example instance with p_i random, $\pi_i = 0.5$

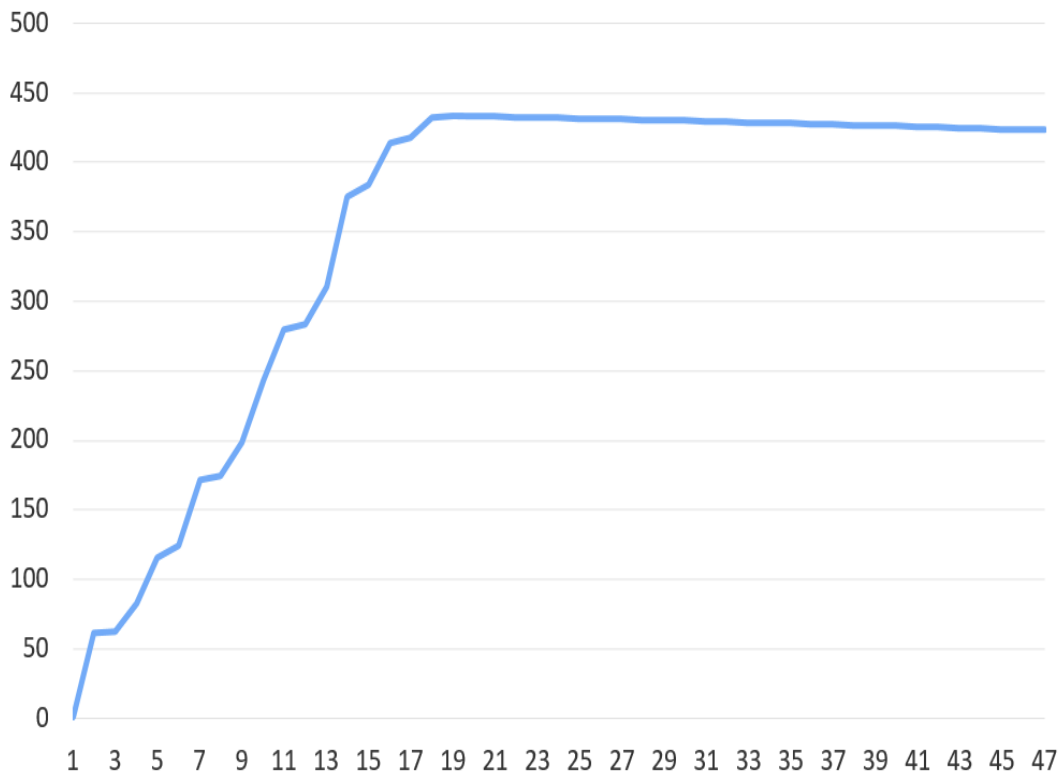

Figure 3.5. $u(\tau)$ of an example instance with p_i random, π_i random

3.1.4 Tuning of the number of samples

In this part, we study the influence of number of samples s when approximating the objective functions using Monte Carlo sampling.

In the experiments, the Monte Carlo evaluator is run 50 times on the solutions of 8 different POP instances, with different number of samples $s \in \{10, 20, \ldots, 90, 100, 200, \ldots 1000\}$. Given a solution, we compute $\overline{x}_s$ as the average value of the objective function approximated with s samples. Since an exact solution is not available for all the instances considered, the best (feasible) solution and the corresponding objective function value reported in [1] for each instance is considered as a reference value X_{ref}.

Two indicators are set for the analysis of the results on each instance. δ_1 is the relative difference between $\overline{x}_s$ and x_{ref}, which indicates the accuracy of the results. δ_2 is relative standard deviation for each number of sample s, which indicates the stability and consistency. The indicators are calculated as follows:

$$\delta_1 = \left|\overline{x}_s - x_{ref}\right| \cdot \frac{100}{\overline{x}_s} \tag{3.2}$$

$$\delta_2 = \sqrt{\frac{\sum_{i=1}^{50}(x_i - \overline{x}_s)^2}{N}} \cdot \frac{100}{\overline{x}_s} \tag{3.3}$$

Notice: Both indicators are calculated with the relative values instead of absolute values, in such a way that the results obtained on different instances can be compared together for more general conclusions.

Figures 3.6 and 3.7 show the values of indicator δ_1 and δ_2 on the y-axis, for the different number of samples s on the x-axis. Each curve represents a test instance of the 8 considered. In Figure 3.6 we can observe that generally δ_1 reduce to 1% when $s \geq 50$, but δ_2 in Figure 3.7 is still relatively high around $s = 50$. Considering δ_1 and δ_2 together, if a small number of samples is wanted for the experiments, 50 could be the choice, while considering stability of the results, 400 is a better choice. It is worth noting that, in a more instance-dependent analysis, for some instances, values of s lower than 50 work well, for others higher values are needed. Therefore, the choice of the number of samples can be instance dependent.

We are also interested in the computing speed as a function of the number of samples. We remind the reader that the theoretical computational time for the evaluation of a solution is $O(ns)$, where n is the dimension of an instance and s the number of samples. The computation time is longer for instances with larger dimension and when more samples are used for evaluation. However in practice, the evaluation is extremely fast. Therefore, we use the number of evaluations per second instead of computation time for a more meaningful observation. Figure 3.8 shows the number of evaluations per second on the y-axis, for the different number of samples s on the x-axis. Each curve represents a test instance among the 8 considered. With t being the actual computation time, the number of evaluation per second represents the value of $\frac{1}{t}$.

Theoretically, the computation time t increases linearly with increasing s, while the number of evaluations per second decreases gradually as a consequence. In Figure 3.8 we observe the same: for $s \leq 100$, the number of evaluations per second drops dramatically when s is

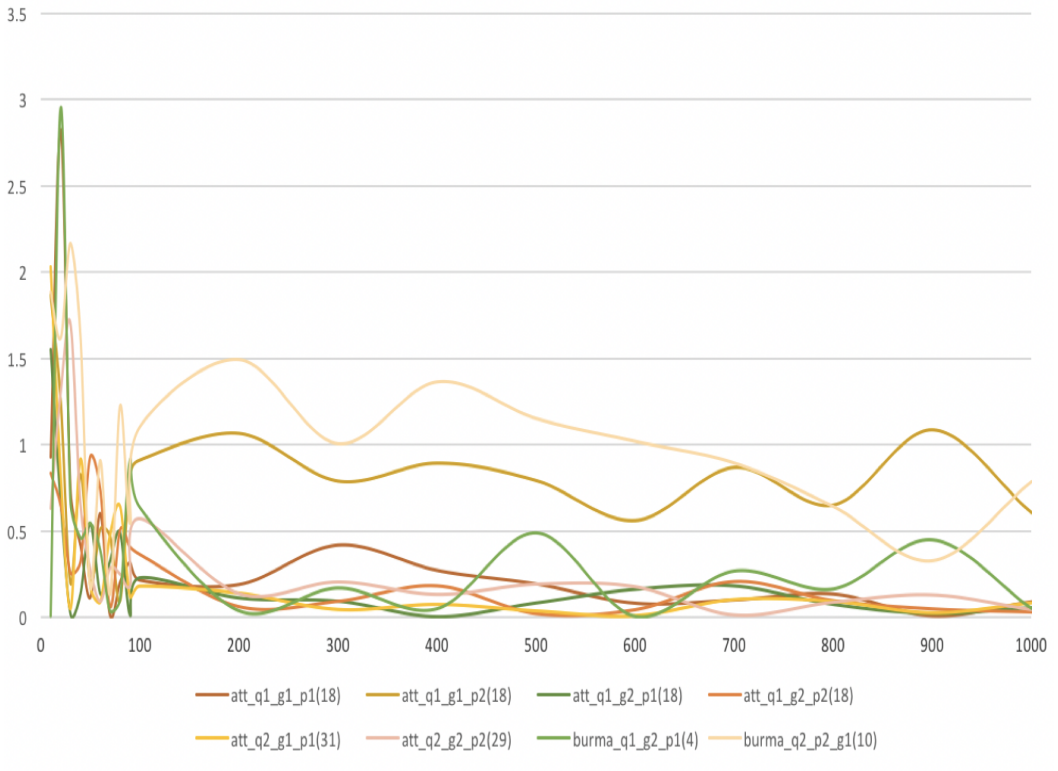

Figure 3.6. Relative difference between $\overline{x}_s$ and x_{ref} for 8 test instances

increased, while for $s > 100$ there is a trend of a more gradual decrease. The size of the instances n influences the computational speed as well as mentioned.

In general, the relative difference between $\overline{x}_s$ and x_{ref} is less than 1% when $s \geq 50$ according to Figure 3.6, and the computational speed is over 10^5 evaluations per second when $s \leq 250$ according to Figure 3.8. Therefore, number of samples $50 \leq s \leq 250$ is a considerable choice to be used for the Monte Carlo sampling method which gives a good balance between the quality of the approximation and the computation speed.

More in general, we can conclude that Monte Carlo sampling is an extremely fast and precise method, and appears to be suitable to be used inside heuristic solvers.

3.2 A heuristic speed-up criterion for the Monte Carlo sampling method

In Section 3.1 we have proven the Monte Carlo sampling approximation is fast and efficient to be used for evaluating the objective function of the POP. In this section we build a Monte Carlo evaluator with a heuristic speed-up criterion based on the characteristics of the POP, which offers the possibility of further improvements on evaluating speed. The work presented has appeared

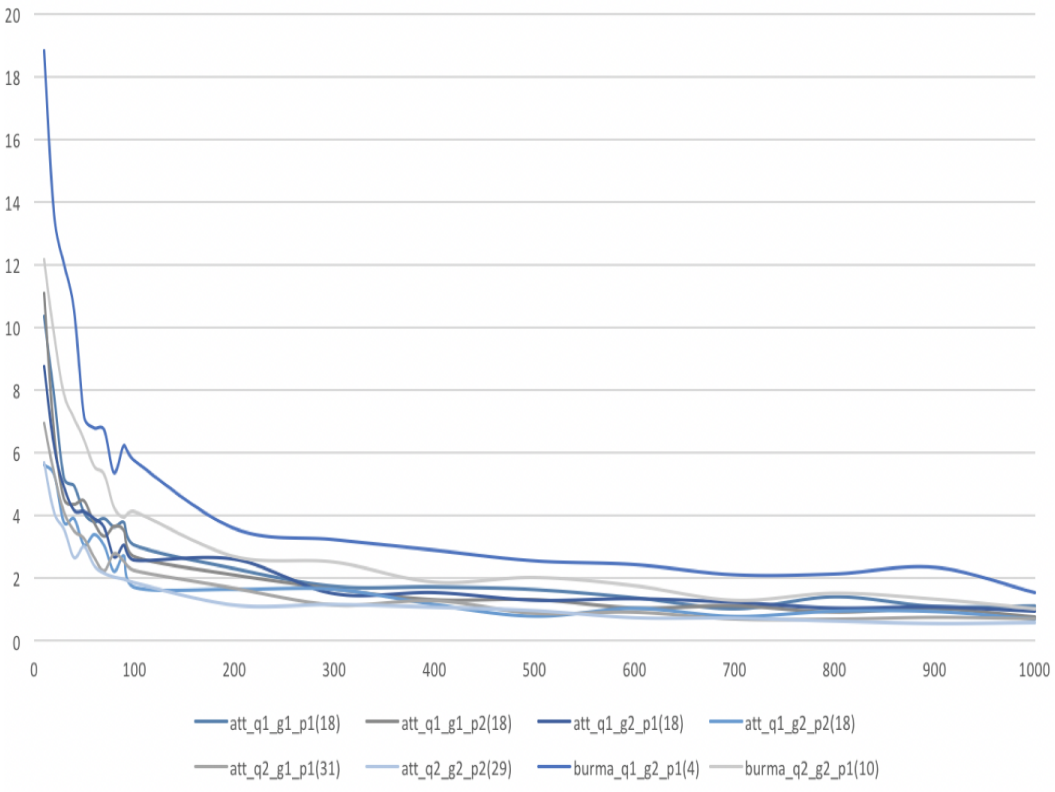

Figure 3.7. Relative standard deviation for 8 test instances

in [15].

3.2.1 Methodology of the heuristic speed-up criterion

By design, when the Monte Carlo evaluator is embedded in heuristic solvers for the POP (see Chapter 4 and Chapter 5), complete tours including all customers from the input instances are fed directly to the evaluator. As mentioned in Section 3.1.3, during the evaluation of a complete tour τ, a peak objective function value will be encountered, and only the customers visited until this peak are selected for a visit. Therefore, a heuristic approach could be used to identify this last customer visited at the peak value and stop the evaluation of the remaining customers. Since finding this maximum value is consistent with the objective of the POP, by stopping the evaluation after the deadline is incurred, unnecessary calculations will be avoided and the computation time can be reduced.

Ideally, the objective function value increases when we visit one customer after another, and the maximum value is obtained at the last node visited before the deadline D (Figure 3.9). This fits the usual situation encountered in real life problems: the customers of the sequence are visited with a positive profit until the deadline is incurred. At this point, a maximum value can be observed, and after that it is not worth to continue the tour because the balance only decreases due to the extra travel time incurred without picking up any prize.

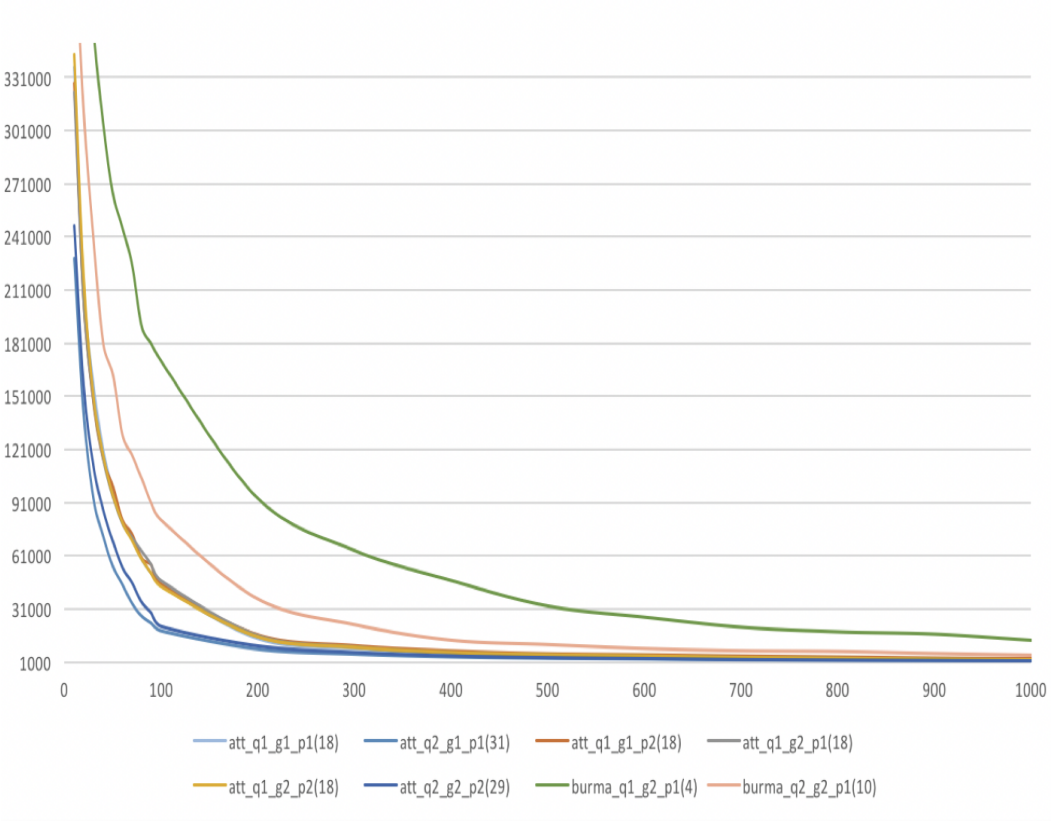

Figure 3.8. Computational speed for 8 test instances

However, the profit of the POP is the difference between the expected prize and expected travel time, and this subtraction causes an issue: we cannot guarantee a positive profit for all customers visited before the deadline, especially for instances in which customers are far away from each other and each of them has a relatively low prize associated. In such a case, a few customers might be visited with negative profit before the deadline incurred, thus the heuristic speed-up criterion cannot simply stop the evaluation once the first drop in the expected profit is detected by the Monte Carlo evaluator (Figure 3.10). To deal with this problem, a tolerance value y that allows the profit to drop several times is introduced to prevent the evaluation from stopping too early due to a temporary negative profit, while not wasting time to evaluate the whole tour anyway. With an appropriate value of y, the accuracy of the best feasible solution extracted during evaluation can also be improved, by excluding the circumstances when the travel time still allows to visit a few more customers before the deadline, but the prizes collected are not enough to make an increase of profit.

When y=1, the evaluation stops when the objective function value first starts to decrease. This would save the most time for the instance in Figure 3.9 but will severely affect the approximation accuracy for the instance in Figure 3.10. If y takes the value of the dimension n of a given instance, the evaluator always computes until the last node without stopping (and returns the maximum value and the corresponding position for the best feasible solution). This is equivalent to the case where the heuristic speed-up criterion is not implemented, thus no

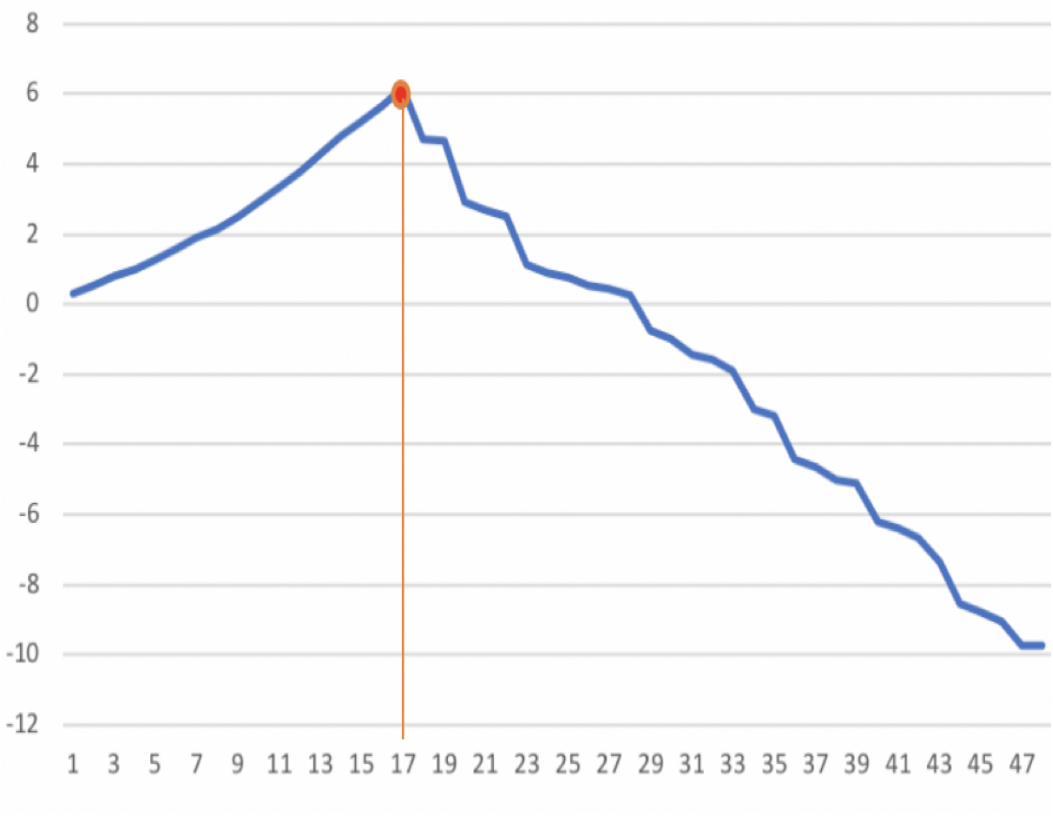

Figure 3.9. An instance with profit increase until the deadline D is incurred

speed is gained, and no accuracy is lost.

In the following subsection, we study on the impact of the heuristic speed-up criterion in terms of precision and speed.

3.2.2 Tuning of the tolerance value y

With the same data sets and testing environment described in Section 3.1.2, experiments are run for each instance of the 264 POP instances with the number of samples set to $s = 100$. As alternatives for the parameter y, four fixed values $\{1, 5, 10, 20\}$ and four flexible values $\{1/10n, 1/4n, 1/2n, 3/4n\}$ are tested, with n being the number of customers of the instance under investigation. The solutions considered for these tests are the best known solutions available for each instance.

We denote as X_v the evaluation value of a complete tour obtained by the Monte Carlo evaluator with a certain value v for parameter y. The value X_n obtained with $v = n$ is considered as a baseline. The gap for a given value v for parameter y is calculated as follows:

$$gap(v) = |X_v - X_n| \cdot \frac{100}{X_n} \tag{3.4}$$

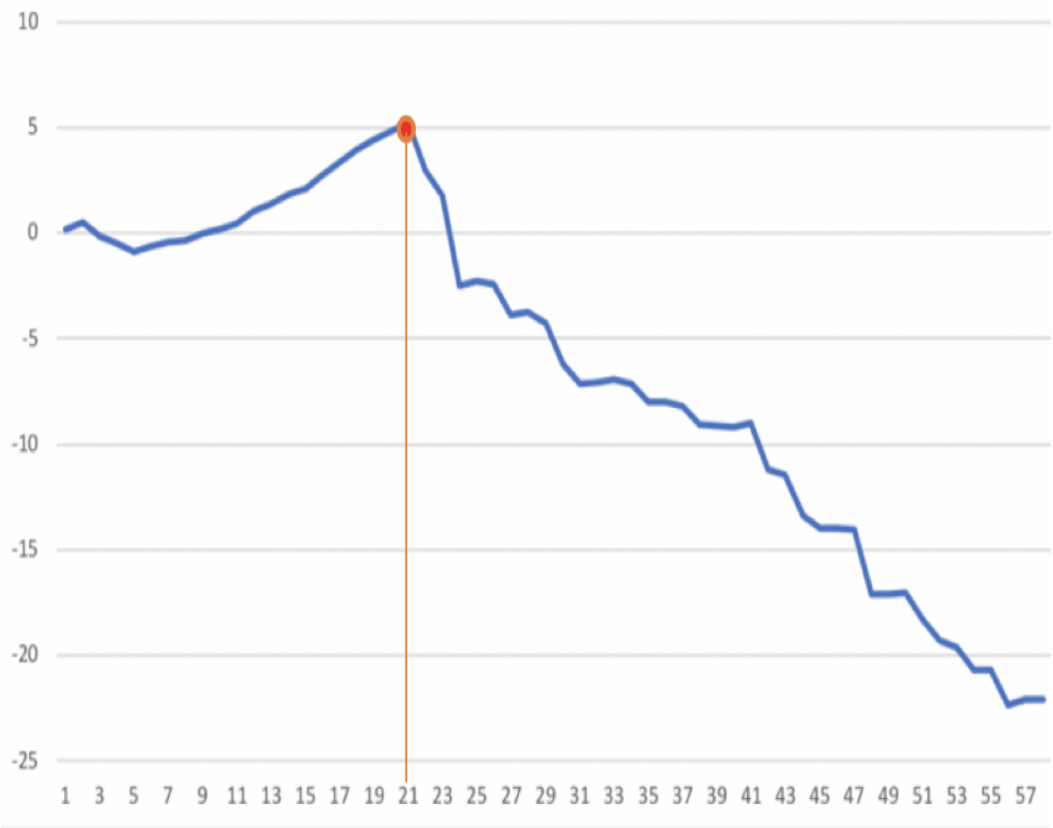

Figure 3.10. An instance with irregular profit before the deadline D is incurred

Table 3.1. The average gap increase (%) for different values v for parameter y

Fixed v	1	5	10	20
$gap(v)$	8.627	0.321	0.005	0
Flexible v	$0.1n$	$0.25n$	$0.5n$	$0.75n$
$gap(v)$	0.408	0	0	0

We calculate the average gap for the 264 instances in Table 3.1.

From Table 3.1 we can observe that when $v = 1$, the evaluation has a relatively large gap in average, which shows the negative profit situation described in Figure 3.10 exists among the 264 test instances considered. The gap goes down as v increases, and there is no gap between X_v and X_n when $v \geq 20$ or $v \geq 0.25n$.

We then consider the average computation speed for different v value. We again consider the computation speed with $v = n$ as a baseline S_n and the average computation speed with a certain v value is denoted as S_v. The average speed gained with different values of v for parameter y is calculated as follows:

$$\text{speed gain}(v) = |S_v - S_n| \cdot \frac{100}{S_n} \tag{3.5}$$

Table 3.2. The average speed gain (%) for different values v for parameter y

Fixed v	1	5	10	20
speed gain(v)	8.9	7.5	5.3	2.1
Flexible v	$0.1n$	$0.25n$	$0.5n$	$0.75n$
speed gained(v)	6.6	5.3	3.3	0.03

We calculate the average speed gained with different v values for each instance, then compute the average speed gained for all the 264 instances in Table 3.2.

From Table 3.2 we can observe that the speed is improved by 8.9% when we take $v = 1$. The speed decreases when v takes larger values. When the value reaches $v = 0.75n$, the speed-up effect is negligible.

From the Monte Carlo evaluator perspective, we conclude that $v = 0.25n$ or $v = 10$ are good choice for general case. Note that when the Monte Carlo evaluator with speed-up criterion is embedded into a solver (see the RRLS solver in Chapter 4), the value of tolerance y influences as well the search path, which may lead to different solutions. Therefore, the choice for the value of tolerance y could be peculiar in different solvers.

3.3 Conclusions

In this chapter we introduced a method of approximating the objective function of the Probabilistic Orienteering Problem based on Monte Carlo sampling. The same Monte Carlo sampling procedure is used to decide how many customers of a given complete tour should be visited in order to maximize the profit, and a relevant best feasible solution is extracted. Such a novel characteristic is going to be very useful once the evaluator is embedded into metaheuristic algorithms introduced in the coming chapters. Computational studies on the performance of the Monte Carlo evaluator with a speed-up criterion we introduced, both in terms of precision and speed, were also presented.

Chapter 4

A Random Restart Local Search Heuristic Algorithm

In Chapter 3 we have discussed the use of Monte Carlo sampling method for the approximation of the objective function value of the Probabilistic Orienteering Problem. Our purpose is to use the Monte Carlo evaluator described in Section 3.2 as interrelated part of heuristic methods to solve the problem. In this chapter, a POP heuristic solver based on the embedding of the Monte Carlo evaluator in a Random Restart Local Search heuristic is proposed. The work has appeared in [15]. A study on different methods of (re-)initialising solutions to improve the performance of the algorithm is also discussed. This related work has appeared in [17].

4.1 A 2-opt Local Search heuristic

In the searching process of an optimization algorithm, we move from one complete tour to another based on the concept of neighbourhood of a given tour. The neighbourhood is defined by a local search method. By combining the local search method and the Monte Carlo evaluator we propose, a POP solver is designed.

The first local search method we use in this work is a 2-opt heuristic proposed by Croes [19] with basic move suggested by Flood [23], which deletes two non-adjacent edges of a tour and reconnects the two paths resulting from this break in the other possible way without creating sub-tours. To be more specific, for a given tour $\tau = (i_0 = 0, i_1, i_2, ..., i_n, i_{n+1})$, we pick all possible first customer j from $\{i_1, ..., i_{n-1}\}$ and all possible second customer k from $\{i_{j+1}, ..., i_n\}$, then reverse the path between these two customers, obtaining $\tau' = (i_0, i_1, ..., i_j, i_k, i_{k-1}, ..., i_{j+1}, i_{k+1}, ..., i_n, i_{n+1})$. A corresponding example of the 2-opt move is shown in Figure 4.1. More details on the 2-opt heuristic can be found in [34]. The sequence of the new tours generated from τ is referred to as the neighborhood of τ, and is denoted as $\mathcal{N}(\tau)$.

Some theoretical results about the number of complete tours and feasible solutions present in the 2-opt neighbours are considered, which help to understand the searching process more intuitively, especially when metaheuristics are built on top of the local search procedure (for example, the Tabu Search heuristic presented in Chapter 5).

Given a complete tour τ with n customers, the total number of complete tours contained

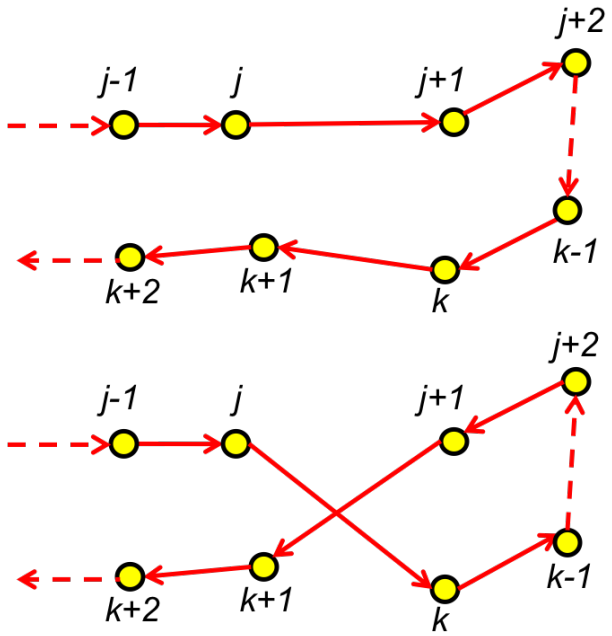

Figure 4.1. Example of a 2-opt move

in the 2-opt neighbourhood $\mathcal{N}(\tau)$ is determined by the number of combinations of picking 2 customers out of n:

$$|\mathcal{N}(\tau)| = \frac{n(n-1)}{2} \tag{4.1}$$

Given a feasible solution $\sigma(\tau)$, the total number of feasible solutions contained in the 2-opt neighbourhood $\mathcal{N}(\sigma(\tau))$ is determined by the number of combinations of picking 2 customers out of q, with $q = |\sigma(\tau)|$ being the number of customers in the feasible solution $\sigma(\tau)$ extracted from the complete tour τ:

$$|\mathcal{N}(\sigma(\tau))| = \frac{q(q-1)}{2} \tag{4.2}$$

4.2 Methodology of the RRLS algorithm

The method starts with a complete tour τ_0 generated at random. The tour is then optimized by crossing over with a 2-opt heuristic operator that iteratively deletes two edges of a first-improve tour and reconnects the two paths resulting from this break in the other possible way (as described in Section 4.1). The evaluation of each tour is done by the Monte Carlo sampling approximation (see Section 3.2), that evaluates the objective function value $u(\tau)$ of a given tour τ, as well as deciding the subset of customers to serve, by extracting the best feasible solution $\sigma(\tau)$ with the objective function value $u(\sigma(\tau))$. The current best tour τ_{local} is updated every time a better tour is found in the neighborhood $\mathcal{N}(\tau)$ generated by the 2-opt method, and a new sequence of swaps is generated from the current best tour. When no improving tour can be found in the neighborhood of the current best tour, a new tour is generated at random, and the

process restart from the beginning until the time limit is reached. Every time a new random tour is generated, the update of the current tour restart from it. This prevents the algorithm from being stuck on a bad search space area. The final output is the best feasible solution extracted from the best complete tour among all the runs of the method.

The pseudo code of the RRLS algorithm we propose is outlined in Algorithm 1.

Algorithm 1: Random Restart Local Search

```
τ_0 = RandomTour();
τ_best = τ_local = τ_0;
while runtime ≤ max_runtime do
    for τ ∈ N(τ_local) do
        if u(σ(τ)) > u(σ(τ_local)) then
            τ_local = τ;
        end
    end
    if u(σ(τ_local)) > u(σ(τ_best)) then
        τ_best = τ_local;
    end
    if no improving 2-opt move exists then
        τ_local = RandomTour();
    end
end
return σ(τ_best);
```

In Section 3.2 we have proposed a heuristic speed-up criterion in the Monte Carlo evaluator and studied the influence of the tolerance value y to balance between precision and speed. When the Monte Carlo evaluator is embedded in the RRLS algorithm, with the heuristic speed-up criterion, active the tolerance value y influences as well the searching path. This may lead to different tours and affect the performance of the algorithm. Experimental studies in this direction are carried out in the following Section 4.3.

4.3 Tuning of the MC evaluator embedded in the RRLS algorithm

4.3.1 General case

With the same data sets and testing environment as described in Section 3.1.2, the experiments are run with a maximum solving time of 600s for each instance, with number of samples $s = 100$ (according to the experiments in Chapter 3). From previous experimental results in Section 3.1.4, the computation time required by the Monte Carlo evaluator to approximate for a given tour with this number of samples is about 10^{-5} to 10^{-4} seconds, which is almost negligible. Therefore, the running time of the experiments is dominantly consumed by finding the best feasible solution among random restart 2-opt-moves.

In order to study the impact of the tolerance value y (see Section 3.2) on the RRLS solver, we denote as C_v the cost of the best feasible solution retrieved with a certain value v for tolerance y in $\{1, 5, 10, 20, 1/2n, 3/4n, n\}$. The results reported in [1] are considered as reference values

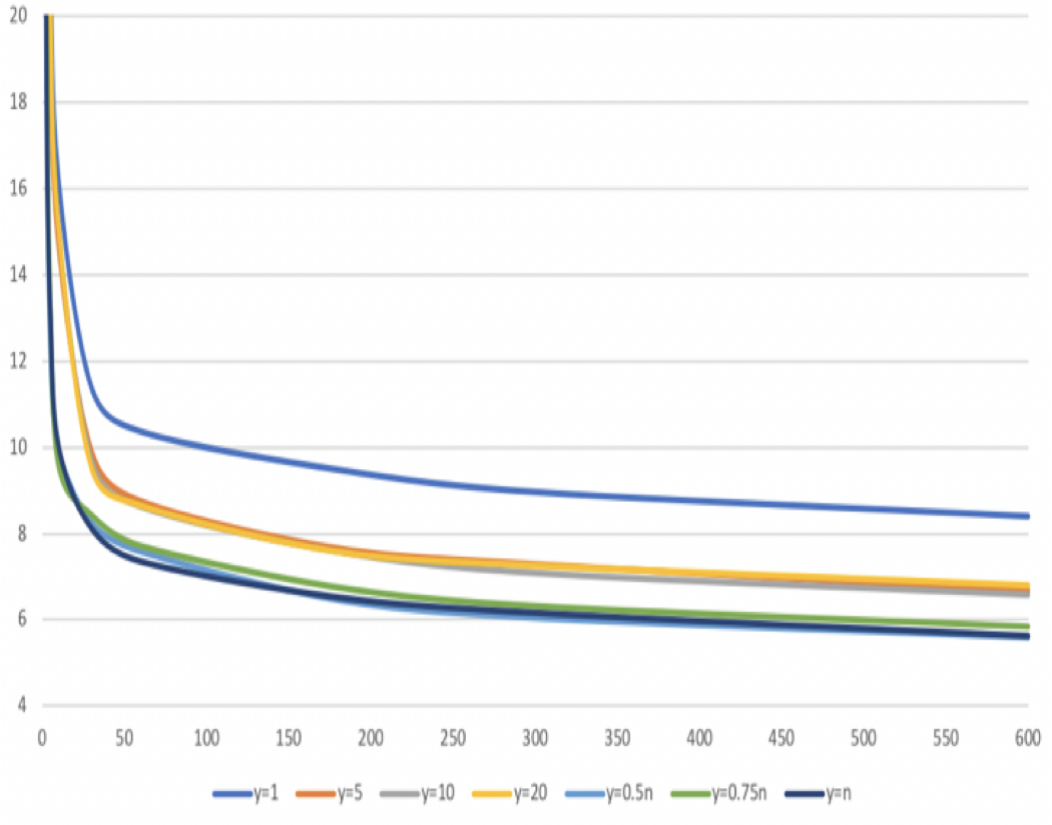

Figure 4.2. Average gap (%) over time for 264 POP instances

C_{ref}. The indicator of precision is defined as the gap in percentage between C_v and C_{ref}. For a given instance, the gap is calculated by:

$$gap(v) = \left|C_v - C_{ref}\right| \cdot \frac{100}{C_{ref}} \tag{4.3}$$

The average gap for all the 264 instances with different v value at different times during the evaluation is shown in Table 4.1. In order to appreciate also the speed of convergence, Figure 4.2 shows the corresponding graph. We observe that the average gap drops dramatically below 10% in 30 seconds for all v values except for $v = 1$. In general, $v = 1$ gives the worst results, $v = n$ and $v = 0.5n$ perform the best, and flexible values are better than fixed values. This suggests that the choice of tolerance value y is quite instance dependent.

As described in Section 3.1.2, the characteristics of the POP instances are: DIMENSION, DEADLINE TYPE, PRIZE TYPE, PROBABILITY TYPE, and EDGE WEIGHT TYPE. The last characteristic is inherited from the TSP library thus not specially mentioned for the POP characteristics in Section 3.1.2. In order to figure out which are the more significant characteristics that influence the results, a multivariate analysis has been done using R[1].

First, we fit an analysis of variance model using the "aov" function, which produces regres-

[1]https://www.r-project.org/

Table 4.1. Average gap (%) for 264 POP instances

v\ time	30s	60s	180s	300s	600s
1	11.4	10.4	9.5	8.9	8.4
5	9.9	8.8	7.7	7.3	6.7
10	9.7	8.6	7.6	7.1	6.6
20	9.5	8.6	7.6	7.2	6.8
0.5n	8.3	7.6	6.5	6.1	5.6
0.75n	8.4	7.7	6.7	6.3	5.8
n	8.1	7.4	6.5	6.2	5.6

```
> summary(y1.aov)
             Df Sum Sq Mean Sq F value   Pr(>F)
X[, 1]        1  22986   22986 119.820  < 2e-16 ***
X[, 2]        2    545     273   1.422 0.243245
X[, 3]        1   2601    2601  13.559 0.000282 ***
X[, 4]        1      9       9   0.050 0.824104
X[, 5]        3  16062    5354  27.909 1.22e-15 ***
Residuals   255  48920     192
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 4.3. Analysis of Variance of the POP characteristics

sion coefficients, fitted values, residuals, etc. Then we produce a type I (sequential) ANOVA table, which is an extension of the independent samples t-test for comparing in a situation where there are more than two groups [64]. For each of the characteristics, the hypothesis $\mathcal{H}_0$ is that "there is no difference between results obtained by this characteristic and other characteristics". The alternative hypothesis $\mathcal{H}_1$ is the opposite. When the p-value is less than 0.05, it indicates strong evidence against the null hypothesis, then we reject the null hypothesis. Figure 4.3 shows the output of the test, with 1 to 5 referring to the five characteristics:DIMENSION, DEADLINE TYPE, PRIZE TYPE, PROBABILITY TYPE, and EDGE WEIGHT TYPE.

In Figure 4.3 we observe that DIMENSION, PRIZE TYPE and EDGE WEIGHT TYPE are significant different from the other characteristics, therefore we consider them as the main factors. Since EDGE WEIGHT TYPE influences mainly the way of calculating distances (e.g. Euclidean distance or Manhattan distance, 2D or 3D, etc), it is relatively less relevant for our study on POP. In the following subsections, we study separately the influence of the tolerance value y regarding the two factors: "DIMENSION" and "PRIZE TYPE". This discovery matches the assumption that the tolerance value y has effect mainly on certain instances that has either long travelling time in between the customers or small prize values, which might cause negative profit when visiting a customer.

4.3.2 Tuning of the tolerance value y based on Dimension

The dimension n of the POP instances varies from 14 to 99, we divide the 264 instances into 3 groups, as in [1]:

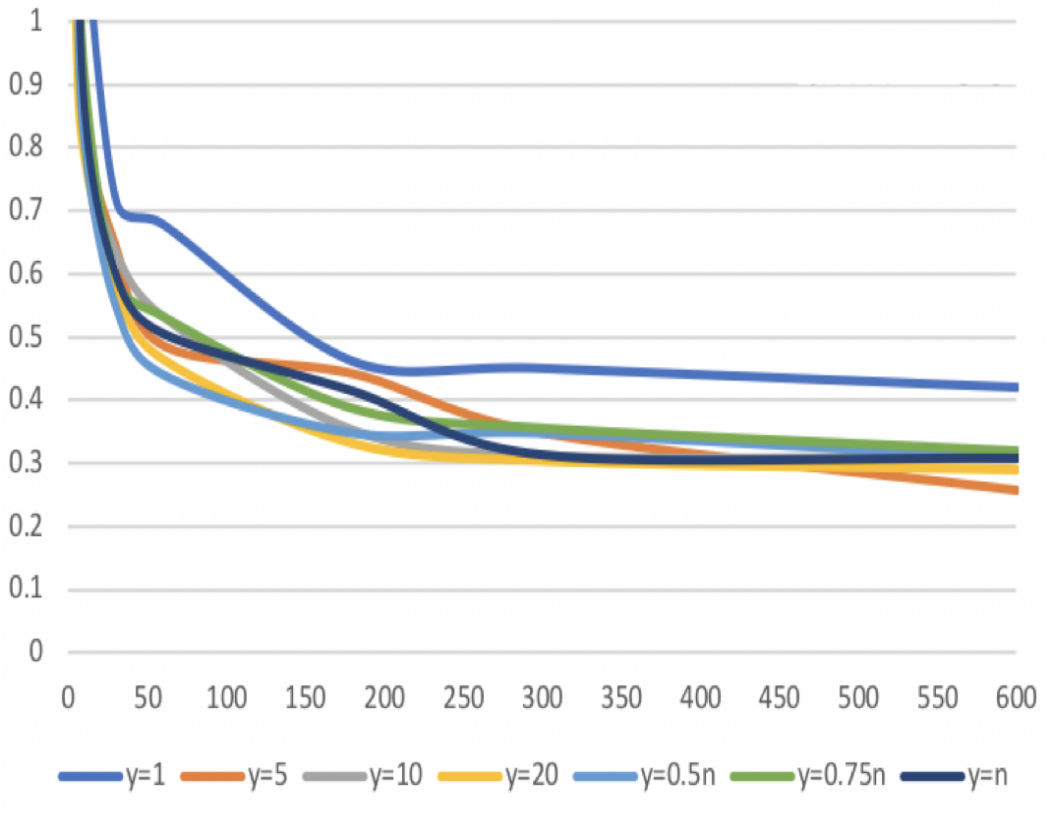

Figure 4.4. Average gap (%) over time for 84 POP instances with $n < 30$

- $n < 30$ (84 instances)
- $30 \leq n < 50$ (84 instances)
- $50 \leq n < 100$ (96 instances)

In Figure 4.4, Figure 4.5 and Figure 4.6 each curve presents the average gap (4.3) over the evaluating time obtained with a certain $v \in \{1, 5, 10, 20, 1/2n, 3/4n, n\}$ representing the tolerance value y. Each figure shows the results obtained for a certain dimension group.

From the perspective of dimension we found that, for instances with dimension $n < 30$, the average gap drops below 1% in 10 seconds. The influence of y value is not significant, $v = 1$ is the worst and $v = 5$ is slightly better at the time limit. For instances with dimension $30 \leq n < 50$, the average gap drops below 5% after around 150 seconds, and reaches approximately 4% in 600 seconds. The influence of y value is not significant either, $v = 1$ is the worst and $v = 10$ is slightly better at the time limit. For dimension $50 \leq n < 100$, the average gap becomes larger and the influence of y value is significant. We can observe that $v = 1$ is the worst and $v = 20$ is the best.

The conclusion is that larger values are needed for y for large dimension instances to maintain an acceptable accuracy. For small dimension instances, the gap is small whatever the tolerance value y is, therefore we can simply pick the one with the highest calculation speed to save computing time.

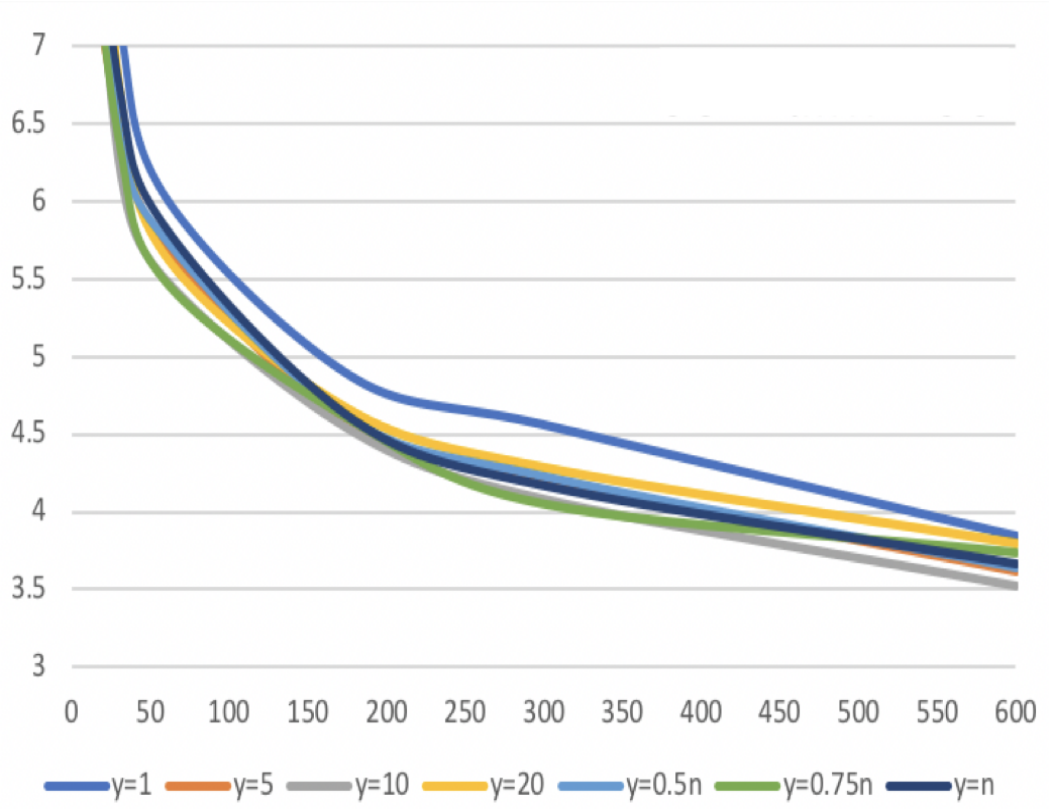

Figure 4.5. Average gap (%) over time for 84 POP instances with $30 \leq n < 50$

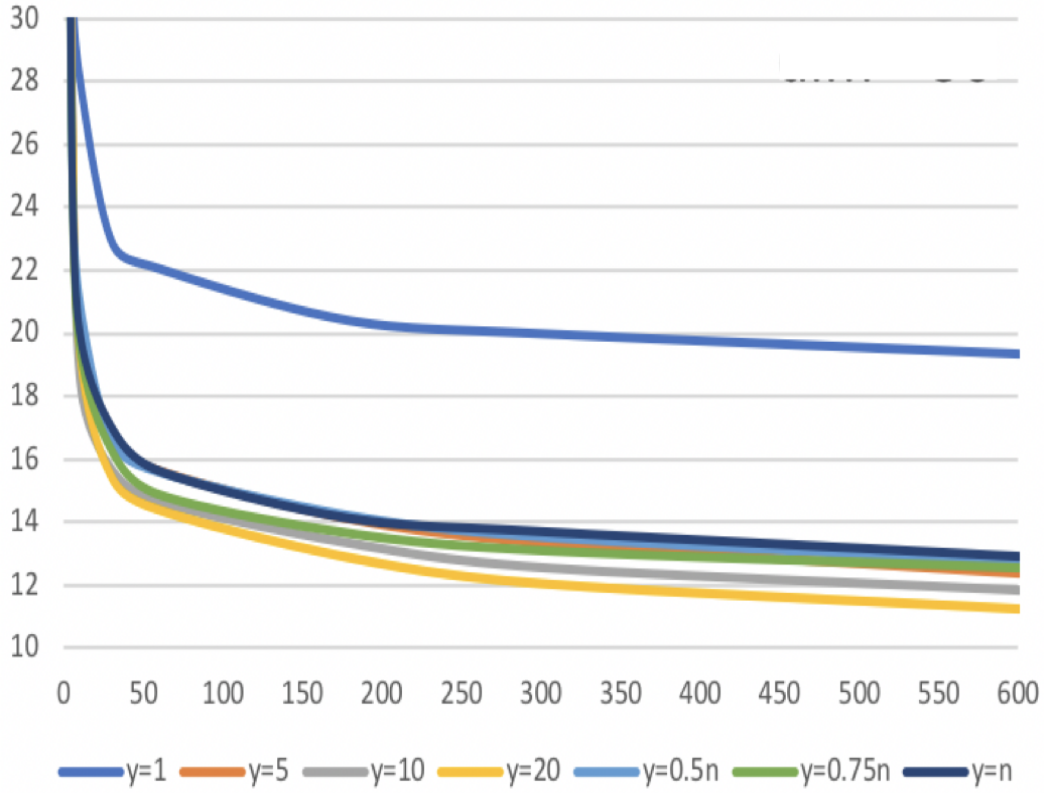

Figure 4.6. Average gap (%) over time for 96 POP instances with $n \geq 50$

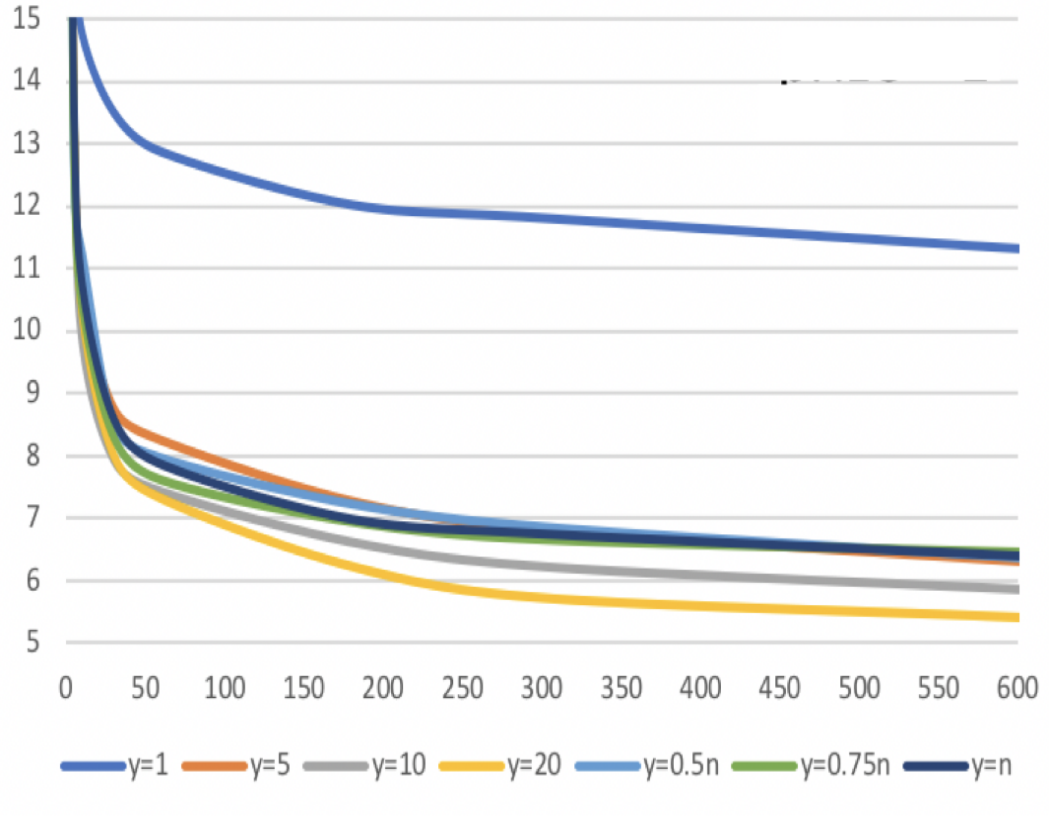

Figure 4.7. Average gap (%) over time for 132 POP instances with $\pi_i = 1$

4.3.3 Tuning of the tolerance value y based on Prize Type

In this part we study the influence of the tolerance value y based on the "PRIZE TYPE" of the instances (see Section 3.1.2). The 264 instances are divided into 2 groups:

- PRIZE TYPE g1: all the prizes are equal to 1 for all customers (132 instances)
- PRIZE TYPE g2: the prize of each customer is a pseudo-randomly generated integer in $\{1, 2, ..., 100\}$ (132 instances)

In Figures 4.7 and Figure 4.8, curves represent the average gap obtained with a certain $v \in \{1, 5, 10, 20, 1/2n, 3/4n, n\}$ being the tolerance value y. Each figure shows the result of a certain PRIZE TYPE group.

Figure 4.7 shows the results obtained on instances with PRIZE TYPE g1. The prizes are all equal and small, thus negative profit issue is high probable to appear. Not surprisingly, $v = 1$ performs bad in this case and $v = 20$ leads to the most accurate results. Figure 4.8 shows the results obtained on instances with PRIZE TYPE g2. The results obtained with different values of v are similar, thus the influence of parameter y is not significant.

From the perspective of Prize types, we can conclude that for instances with small prize values, large values are needed for the tolerance value y to maintain the accuracy. For instances with random prize, the heuristic speed-up criterion is quite applicable. The difference between

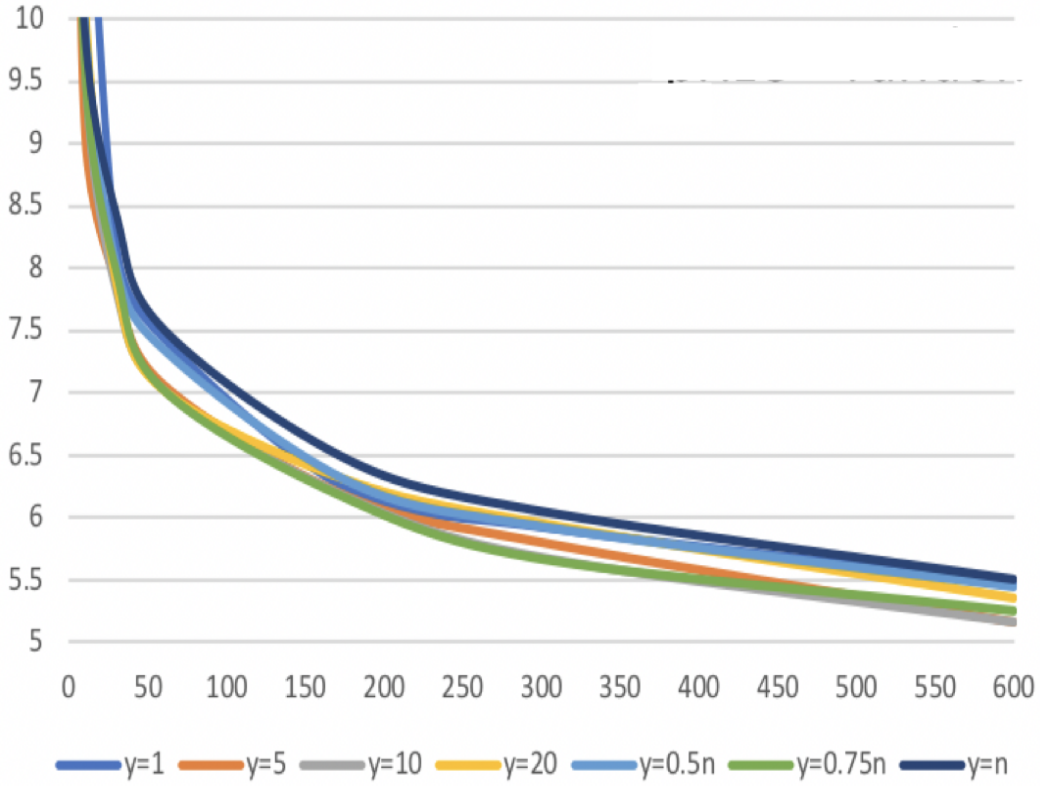

Figure 4.8. Average gap (%) over time for 132 POP instances with $\pi_i = random$

gap with different tolerance values is quite small, therefore, we can pick the one with the best calculation speed to save computing time, e.g. $\nu = 1$.

4.3.4 Summary and considerations

From the previous experiments we can conclude that, for instances with dimension greater than 50, large values should be taken for the tolerance value y to ensure the accuracy, e.g. $y = 20$. This value is not strictly specified for the reason that, even though higher value of y means higher accuracy in the Monte Carlo evaluator, lower value of y has higher computation speed, which will result in more iterations in RRLS, that possibly improves the probability of finding a better tour in the metaheuristic algorithm for the POP. For small instances with dimension no greater than 50, when the prize values are small, we should consider large tolerance values y as well for the same reason, and for other cases we can take $y = 1$. In such a way a good balance between speed and precision can be achieved, and the POP can be solved by using the RRLS method fast, simple, and with reasonably small gaps in practice with respect to the best known solutions for the instances.

With the parameters taking values as described, the average gap with respect to the best known solutions drops below 6% in 250s with RRLS method (Figure 4.2). For dimension $n < 30$, the average gap drops below 1% in 10 seconds (Figure 4.4). The conclusion is that RRLS performs particularly well on small instances. For large instances, 2-opt local search might not be sufficient, stronger local search methods and further improvements are to be employed for more accurate results. In Section 4.4, a different initialization method is proposed to improve the performance of the RRLS algorithm.

4.4 A wiser selection of starting solutions

In this section we study the use of a wiser selection of starting solutions after restart (not completely at random as it happens in the previous version of the method described in Algorithm 1). A method has been proposed to effectively select such (re-)initialising solutions, and we present an empirical study to validate the idea. Our purpose is to show the important role of an effective restarting strategy in achieving better results within a RRLS algorithm.

4.4.1 Generation of initial solutions for the random restart phase

In Section 4.2, the Random Restart Local Search algorithm was starting with a complete tour τ_0 generated at random, and the searching procedure was associated with a new initial solution for the optimization when no improving tour can be found in the neighborhood, generated again at random. The configuration of these solutions from which the RRLS methods re-optimize after a restart phase plays an important role in finding out a good solution using local search based heuristics. Therefore, we propose to modify the generation of the re-initialising solutions inside the RRLS algorithm.

The same algorithm is used with slightly adjustment on a more generalised version. A function *Initialising()* takes the place of *RandomTour()*, which represents different initialization methods under investigation. The pseudo code of the algorithm is outlined in Algorithm 2.

Algorithm 2: Random Restart Local Search version 2

```
τ_0 = Initialising();
τ_best = τ_local = τ_0;
while runtime ≤ max_runtime do
    for τ ∈ N(τ_local) do
        if u(σ(τ)) > u(σ(τ_local)) then
            τ_local = τ;
        end
    end
    if u(τ_local) > u(τ_best) then
        τ_best = τ_local;
    end
    if no improving 2-opt move exists then
        τ_local = Initialising();
    end
end
return σ(τ_best);
```

In the implementation of the Algorithm 1 in the original RRLS algorithm, a *Random Insertion* method [8] is adopted for the initial solution. The method starts from a tour initially containing only the depot 0, then proceeds iteratively. Nodes not included in the current partial tour are randomly inserted into the tour. The procedure is repeated on the incrementally larger partial solution and the method stops when all nodes in the graph have been included in the tour, and we add the destination $n+1$. Solutions generated in this way are fully randomised, thus the searching space is explored more thoroughly comparing to other initialization methods, however, there is a drawback: the method takes longer time to converge due to the low quality of such initial solutions.

We here propose to use a *Nearest Neighbour* method [8] to generate initial solutions. Such a method represents a simple way to generate an initial solution which offers fast converging speed for local searches, but on the other hand is more prone to get stuck in local optima, that are often clearly worse than the global optimum. The method starts also from a tour initially containing only the depot 0. Then at each iteration the nearest feasible node (not yet part of the tour) from the last added node in the current partial tour is added to expand the partial tour itself. The procedure is repeated until all nodes in the graph have been included in the tour, the destination $n+1$ is finally added.

Figure 4.9 shows an example of evolution of the best cost retrieved by the RRLS algorithm over time, when different initialization methods are considered. The results are reported for a representative instance (*att48FSTCII_q1_g1_p1*) from the POP benchmark set introduced in [1]. The objective function value $u(\tau)$ of the instance is depicted on the y-axis, presenting the evolution of the optimal solution found over time on the x-axis (seconds). At this stage we focus on the trajectories obtained while using *Random Insertion* or *Nearest Neighbour* as initialization method. When *Nearest Neighbour* is employed, the search process converges extremely fast to a local maximum. However, no improvement takes place for a long time afterwards, and the best known solution is not achieved in the given time. On the other hand, when the Random Insertion method is employed, the search process converges substantially slower, but

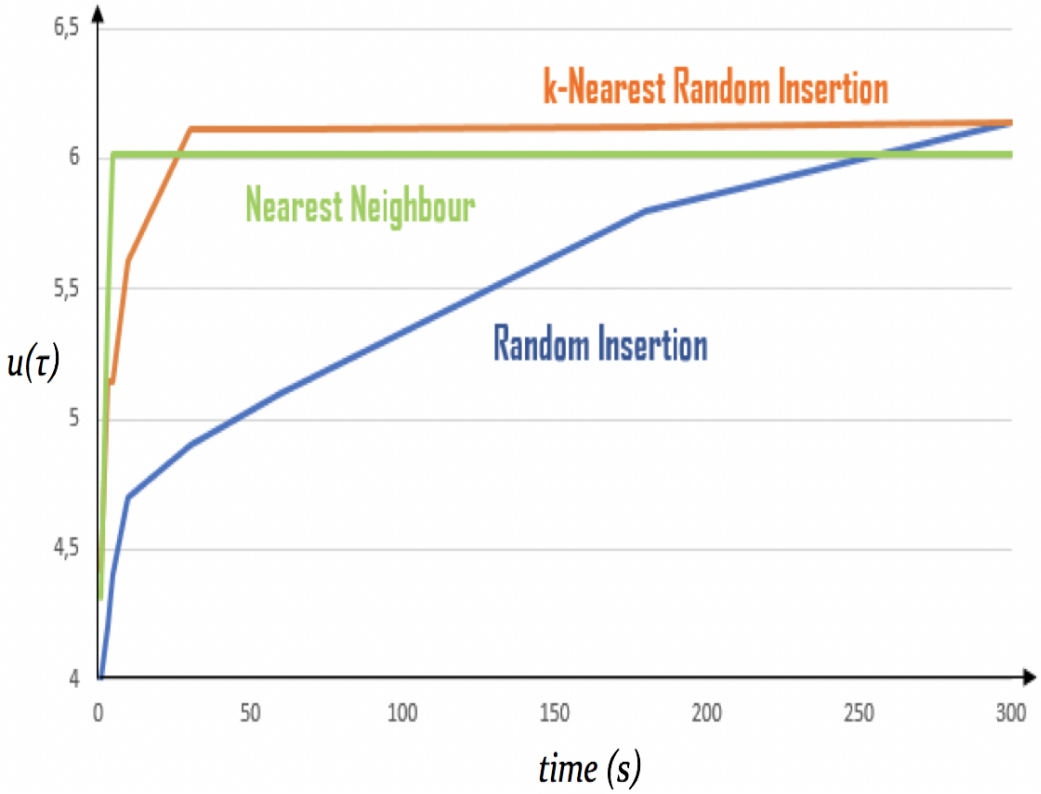

Figure 4.9. Evolution of the best solution retrieved by RRLS over time with three initialization methods (instance att48FSTCII_q1_g1_p1)

the best known solution can be achieved in the given time, notwithstanding the computation time required is long.

The ideal initialization method needs to guarantee a fair exploration of the search space, while delegating the identification of local minima to the embedded local search algorithms. Based on the observations above regarding the characteristics of the *Nearest Neighbor* and *Random Insertion* methods, and their impact on the results retrieved by the RRLS procedure, we here introduce a third initialization method, that in our vision should maintain the best characteristics of the previous two approaches, while minimizing their drawbacks.

The *k-Nearest Random Insertion* method is inspired by the analogous one described in [53]. In the new method, while generating an initial solution, instead of picking the Nearest feasible Neighbour at each step, we carry out a Random Insertion where the candidate nodes are the k feasible ones closest to the last node added in the previous iteration. In this way, rapid convergence and enough randomness can be retained at the same time upon an appropriate tuning of parameter k. In Figure 4.9 the behaviour of a *k-Nearest Random Insertion* with $k = 2$ is also depicted, and the trajectory is clearly the most desirable of the lot, being able to guarantee a fast and precise convergence to the best value.

Note that when $k = 1$, the initial solution is a greedy solution generated by the standard

Nearest Neighbour method. When $k > 1$, the larger the k value is, the more randomized the initial solution will be. In the extreme case when $k = n$, the initial solution is a random solution generated by the standard Random Insertion method.

In the following Section 4.4.2 we study from an experimental point of view the impact of this re-initialization strategy on the performance of the Random Restart Local Search algorithm for the POP. In particular, we assess the performance of the new re-initialization procedure for different values of parameter k.

4.4.2 Effectiveness of parameter k

The data set used for this set of experiments are the same as described in Section 3.1.2. In the experiments we are more interested in the searching procedure, therefore less samples are used comparing to Section 4.3.1. We select $s = 50$ samples to be used by the Monte Carlo evaluator embedded in the RRLS method. According to the previous study in Section 3.2, this number of samples is in the range that provides a good balance between the quality of the approximation of the objective function value, and the computation speed. Meanwhile the computational speed is faster, more iterations are expected and the search space is more likely to be better explored.

The main study we present is about parameter k, used by the k-Nearest Random Insertion method adopted for re-initialising solutions within the RRLS metaheuristics. We test the following values for parameter k: 1,2,3,4,5. A maximum computation time of 180 seconds is allowed to each run in all the experiments reported.

The objective function value of the best solution retrieved by the RRLS method with a certain value of k for the k-Nearest Random Insertion component, at a certain time t for a certain instance i is denoted as $X_{k,t,i}$. The quality of the best solutions retrieved is compared with a baseline X_i^{ref} given by the results reported in [1]. For an in-depth and more precise study, the 264 instances are partitioned into different groups as in [1] according to their characteristics (see Section 3.1.2), and a separate analysis is reported for each group in Table 4.2. To be more specific, when considering the dimension, there are 84 small instances with $n < 30$, 84 medium instances with $30 \leq n < 50$, and 96 large instances with $50 \leq n \leq 98$. When considering Deadline values, there are three groups with 88 instances in each with $\omega = \{\frac{1}{4}, \frac{2}{4}, \frac{3}{4}\}$ representing short, medium and long deadlines. The instances are finally partitioned with respect to the distribution of the prizes and to the distribution of the probability of requiring a visit.

For every subgroup sub with a certain number of instances we calculate, for each setting k, the average error $\overline{e_k}$ over the instances of the subgroup at time t. The error is calculated as the average of the relative difference between $X_{k,t,i}$ and X_i^{ref} calculated for each instance i, as follows:

$$\overline{e_k}(\%) = \frac{\sum_i^{|sub|} \left| X_{k,t,i} - X_i^{ref} \right| \cdot \frac{100}{X_i^{ref}}}{|\text{sub}|} \tag{4.4}$$

First, we are interested in the final results obtained with different values of k when the maximum computation time is reached. The detailed average error $\overline{e_k}(\%)$ for each subgroup and for different k values obtained at time limit is presented in Table 4.2.

Table 4.2. Detailed average error $\overline{e_k}(\%)$ for different k values

	#instances	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$n<30$	84	13.320	0.632	0.436	0.492	0.486
$30\leq n<50$	84	29.417	4.674	5.765	5.836	5.303
$n\geq 50$	96	37.433	10.730	13.904	15.191	15.675
$\omega=1/4$	88	28.716	4.298	5.052	5.516	5.423
$\omega=2/4$	88	29.127	6.228	8.012	8.836	8.923
$\omega=3/4$	88	23.787	6.245	8.023	8.260	8.279
$p_i=1$	132	31.215	4.897	6.711	7.519	7.256
$p_i\in\{1,\ldots,100\}$	132	23.205	6.284	7.347	7.556	7.828
$\pi_i=0.5$	132	27.590	5.063	7.045	7.393	7.168
$\pi_i\in[0.25,0.75]$	132	26.830	6.118	7.013	7.682	7.916
All	**264**	**27.210**	**5.590**	**7.029**	**7.537**	**7.542**

In Table 4.2 we observe that the results obtained by re-initialising solutions with the *Nearest Neighbor* method ($k=1$) performs bad. This is because every time the searching procedure restarts from very similar solutions, and the searching space is not explored well. As described in Section 4.4.1, the larger the k value is, the more randomized the initial solution will be. In the table we observe a trend that for $k\geq 2$ the average error in general goes up as k increases. A possible explanation is that with more randomness, it relatively takes longer to converge, thus within the given fixed time limit, larger k values perform worse.

Preliminary observation from Table 4.2 shows that the setting $k=2$ is the best in general. It performs well in multiple subgroups, except for the group of small instances, where $k=3$ is better. In the following Section 4.4.3 we calculate the p-value between $k=2$ and $k=3$ to check if the difference is significant. In order to check if it is possible that better values can be found when k increases and larger than 5, we also perform a hypothesis test between each pair of k values over time.

4.4.3 Statistical significance of the results on parameter k

First, a hypothesis test is performed on the results obtained with $k=2$ and $k=3$. The hypothesis $\mathcal{H}_0$ is that "there is no difference between results obtained with $k=2$ and $k=3$ for a certain sub-group". The alternative hypothesis $\mathcal{H}_1$ is the opposite. When the p-value is less than 0.05, it indicates strong evidence against the null hypothesis, then we reject the null hypothesis. We carried out a p-value test for all the subgroups and the results are presented in Table 4.3.

In Table 4.3 we observe that the difference between $k=2$ and $k=3$ for small instance groups with $n<30$ and deadline type group $\omega=\frac{1}{4}$ is not statistically significant, while for all the rest characteristics the difference is significant. This indicates that even though $k=3$ is better than $k=2$ on small instances, this difference is not significant, but in other groups where $k=2$ is better than $k=3$ the difference is significant. Therefore, we can conclude that $k=2$ is the best choice among the 5 values we tested for k-Random Insertion method for the POP in general.

As mentioned in Section 4.4.2 we also observe a trend that for $k\geq 2$ the average error in general goes up as k increases. In order to check if this indicates that no better values can

Table 4.3. Detailed p-value between $k = 2$ and $k = 3$

	#instances	p values: 2 vs 3
$n < 30$	84	0.656671275
$30 \leq n < 50$	84	0.009421352
$n \geq 50$	96	0.000138711
$\omega = 1/4$	88	0.132253417
$\omega = 2/4$	88	0.002397467
$\omega = 3/4$	88	0.003500878
$p_i = 1$	132	2.02315E-06
$p_i \in \{1, \ldots, 100\}$	132	1.45938E-05
$\pi_i = 0.5$	132	0.000254197
$\pi_i \in [0.25, 0.75]$	132	0.01019796
All	**264**	**1.37962E-05**

Table 4.4. p-value over time for different k values

k values $\backslash t$	1s	10s	30s	60s	180s
1 vs 2	1.4E-12	2.4E-17	9.0E-20	4.0E-21	5.1E-21
1 vs 3	4.0E-12	5.8E-17	1.9E-20	2.6E-21	1.7E-21
1 vs 4	2.0E-12	1.3E-15	1.7E-19	1.8E-21	1.1E-20
1 vs 5	1.4E-11	1.3E-17	9.3E-21	1.4E-21	5.7E-21
2 vs 3	7.4E-08	2.2E-07	1.0E-05	2.5E-05	1.4E-05
2 vs 4	2.0E-08	4.9E-08	8.3E-07	1.3E-06	3.0E-06
2 vs 5	3.0E-10	8.2E-09	1.2E-07	5.1E-08	6.5E-07
3 vs 4	0.0008	0.0036	0.0114	0.0776	0.3555
3 vs 5	7.0E-07	8.4E-05	9.9E-06	0.0003	0.0203
4 vs 5	0.0068	0.3094	0.0174	0.0213	0.1508

be found when k increase, we perform a hypothesis test between each pair of k values over time. The hypothesis $\mathcal{H}_0$ is that "there is no difference between results obtained with $k = i$ and $k = j$ at a given time". The alternative hypothesis $\mathcal{H}_1$ is the opposite. By calculating p-value over time between each pairs of k, we have a better view of if the difference between k values are significant and whether the difference is influenced by calculation time. In Table 4.4 we perform p-value test between $X_{k,t,i}$ for $i \in$ {all the 264 instances} in pairs of k value at some relevant times $t \in \{1s, 10s, 30s, 60s, 180s\}$.

The result in Table 4.4 shows that $k = 1$ is highly significant different from all $k > 1$ of all time. $k = 2$ is also significant different from all $k > 2$ of all time. $k = 3$ is significant different from $k > 3$ from the start, but as time goes, the results are no longer significant different. The results between $k = 4$ and $k = 5$ are not significant different of all time. The results indicate that there is no much difference after $k \geq 4$. It is also possible to extrapolate that considering even larger values of k, which means going towards a Random Insertion initialization method, would lead to worsening results in the given time, due to a much slower convergence.

In general, we can draw the conclusion that $k = 2$ is the best choice for the *k-Nearest Random Insertion* re-initialising method to improve the performance of the Random Restart Local Search algorithm for POP.

4.5 Conclusions

In this chapter we have solved the Probabilistic Orienteering Problem by using a metaheuristic algorithm based on a Monte Carlo sampling objective function evaluator with a speed-up criterion and a Random Restart Local Search method. Computational studies on precision and speed with different values of a tolerance parameter y to be used in the algorithm have been performed. We have discussed separately the appropriate choice of the tolerance value to be used inside a metaheuristic algorithm for different types of instances, in such a way that the Probabilistic Orienteering Problem can be solved even faster with the same accuracy.

In this work we have also shown how a more effective re-initialization procedure, achieved by a k-Nearest Random Insertion method, can significantly improve the performance of the overall metaheuristic method. A detailed computational study has also revealed the influence of parameter k in the performance of the overall method. This method will also be used for the Tabu Search algorithm, that we will present in Chapter 5.

Chapter 5

A Tabu Search Heuristic Algorithm

In this Chapter, we embed the Monte Carlo evaluator described in Chapter 3 into a Tabu Search (TS) algorithm. The method is designed to be fast, simple, and with reasonably small optimality gaps. A detailed computational study of the new approach is presented, with the aim of studying the performance of the algorithm in terms of precision and speed, while positioning the new method within the existing literature. The work presented has appeared in [13] and [16].

5.1 The role of Monte Carlo sampling within the heuristic

As mentioned in Chapter 2, solving the POP is computationally demanding. In this work, we use again the Monte Carlo sampling method described in Chapter 3 as an interrelated part of a simple heuristic to solve the POP. Due to the heuristic nature of the approach, the approximation of the objective function value $u(\tau)$ is more accurate when more scenarios are used, but this requires more computational time. Previous experimental results from Section 3.1.4 show that in general the approximation error for $u(\tau)$ is less than 1% when $s \geq 50$, and the computational speed is over 10^5 evaluations per second when $s \leq 250$ on a normal modern personal computer. A number of samples in this interval is a considerable choice. When the Monte Carlo approximation is embedded in a metaheuristic algorithm for the POP, high speed is desirable for a better exploration of the searching space, while precision is less crucial, since the overall approach is heuristic in native. Therefore, $s = 50$ is adopted as setting for all the experiments presented in this work.

By design, the TS algorithm treats complete tours touching all customers, while a POP solution only visits a subset of customers that can be served before the deadline D. As described in Chapter 3, the Monte Carlo sampling component is also delegated to extract a POP solution $\sigma(\tau)$ as a prefix of a complete tour τ.

5.2 Comparison between the TS and RRLS methodologies

The main idea of the Tabu Search method we use is similar to that of the Random Restart Local Search algorithm described in Chapter 4. Similarly as in the Random Restart Local Search method, we assume that every time a complete tour with all customers in a given instance is

considered. The Tabu Search algorithm embeds with the same Monte Carlo component to evaluate and extract feasible solutions out of complete tours. Both the RRLS and the TS algorithms are designed to escape from local extrema during the local search procedure. The basic idea of TS is to accept a sequence of non-improving solutions to achieve this goal. This is realized by the use of an explicit memory called Tabu List during the search process. It avoids visiting solutions that have been visited recently.

To be more specific, an initial complete tour will be generated by the k-Nearest Random Insertion method (see Section 4.4) and fed to the TS solver. Experimental results in Section 4.4.2 show that $k = 2$ works well for the POP, therefore we adopt this value for the experiments in this work. The initial tour is then reordered with the a 2-opt local search method (see Section 4.1). In this way a group of neighboring tours is generated. With the Tabu List as a filter, for each of the remaining neighboring tours that has not been visited recently, a POP feasible solution is extracted and its objective function value is evaluated by the Monte Carlo evaluator. The best tour in the neighborhood is selected as the tour for the next iteration. Differently from the RRLS method, the new neighbourhood is generated from the best tour of the previous neighbourhood, which is not necessarily the best tour overall, due to the Tabu list. The attributes of the best tour is then stored in the Tabu List. The searching process continues until a given time limit is reached.

5.3 Memory Structure

Figure 5.1 shows the evolution of the objective function value $u(\tau)$ (on y-axis) over time (on x-axis) for an example instance $bayg29FSTCII\_q1\_g2\_p1$ in the 264 POP benchmarks introduced in Section 3.1.2. With the RRLS method (with 2-opt operator), the result converges very close to the exact value in 2 seconds, but finally reaches the exact value in 39 seconds. With an appropriate tuning of the TS algorithm (with 2-opt operator), the result converges to the exact value in 0.31 seconds. In the following section, we explain in details the memory structure of the TS algorithm.

The function of the memory structure is to filter out some tours and define which tours in the neighborhood $\mathcal{N}(\tau)$ can be explored by the search. The memory commonly consists of tours recently encountered when moving from one tour to another, which remain consequently forbidden for a certain amount of time (iterations).

In order to reduce storage requirements, usually a set of rules, such as a collection of forbidden moves, are stored in the list instead of complete banned tours. The list is called Tabu List. Thus, finding an appropriate way to describe the forbidden tours in a compact structure is also part of the design of a TS algorithm. In this work, the attribute we store in the list is the pair of customers swapped by the 2-opt heuristic (see Section 4.1).

During the searching procedure, each time we select the best non-forbidden tour in the neighborhood $\mathcal{N}(\tau)$ as the incumbent solution τ_{inc} (no matter it is better or worse than current best solution). The two customers j and k between which the route is reversed by the 2-opt operation transforming τ into τ_{inc} are stored in the Tabu List. Any 2-opt operation involving customers in the Tabu List as pivots j and k is consequently now forbidden.

The length of the Tabu List l (also known as Tabu tenure) decides the number of Tabu Search iterations for which customers remain forbidden. When the TS algorithm embeds a

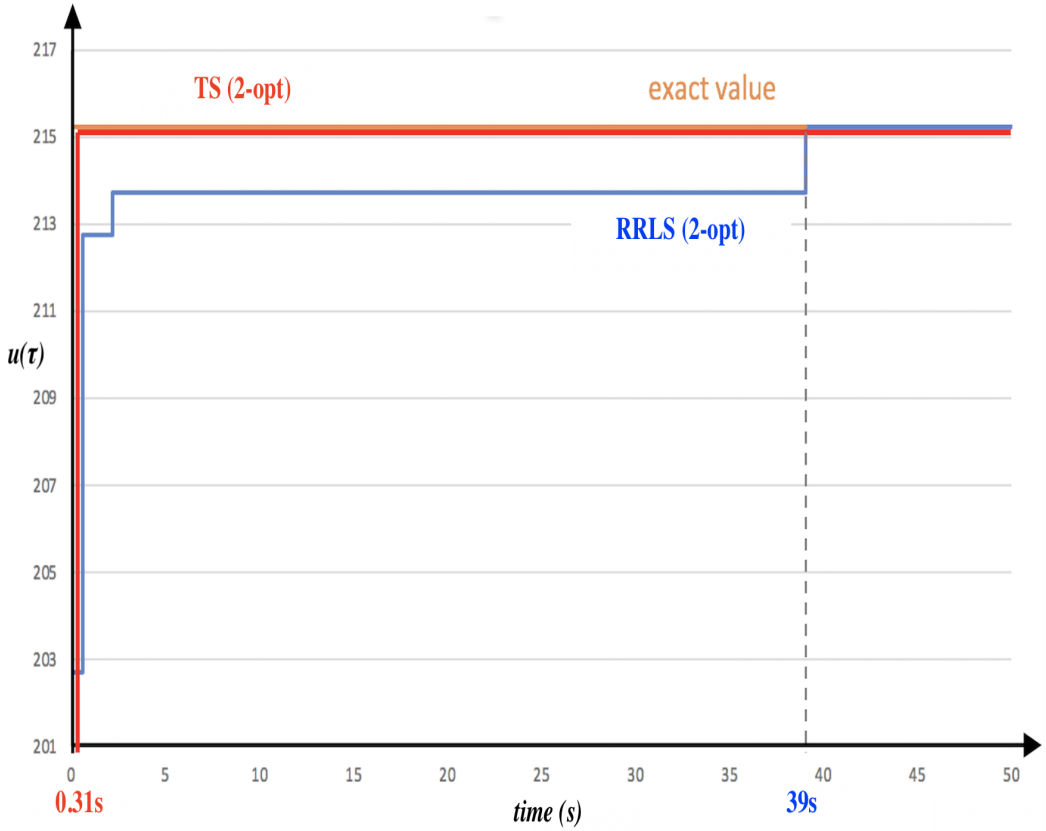

Figure 5.1. Comparison between the TS algorithm and the RRLS algorithm for an example instance

2-opt operator, l pairs of customers are forbidden to move at the same time when the Tabu List is full, thus the maximum value of l should be less than $\frac{1}{2}n$. In different Tabu Search implementations the Tabu tenure can either be a fixed number or a flexible value that changes deterministically with problem parameters [4]. Detailed experimental results about the tuning of parameter l can be found in Section 5.6.2.

5.4 Local Searches

In the solving architecture we propose, local search methods are used to provide heuristic solutions in a limited time. The 2-opt heuristic operator introduced in Section 4.1 is a simple climber that has been proven to be efficient for Orienteering Problems, therefore we adopted it in this work for the POP. However, the operator can actually be replaced with any other operator (for example, 3-opt, 4-opt [6]) as a consequence of the flexibility design of our algorithm. In this section, we take a 3-opt operator as an example for the possible replacement, and to understand if it can improve the results of 2-opt.

In a 3-opt move of out implementation, three non-adjacent edges of a tour are deleted and the tour is reconnected in seven different ways. An example of all the possible combination

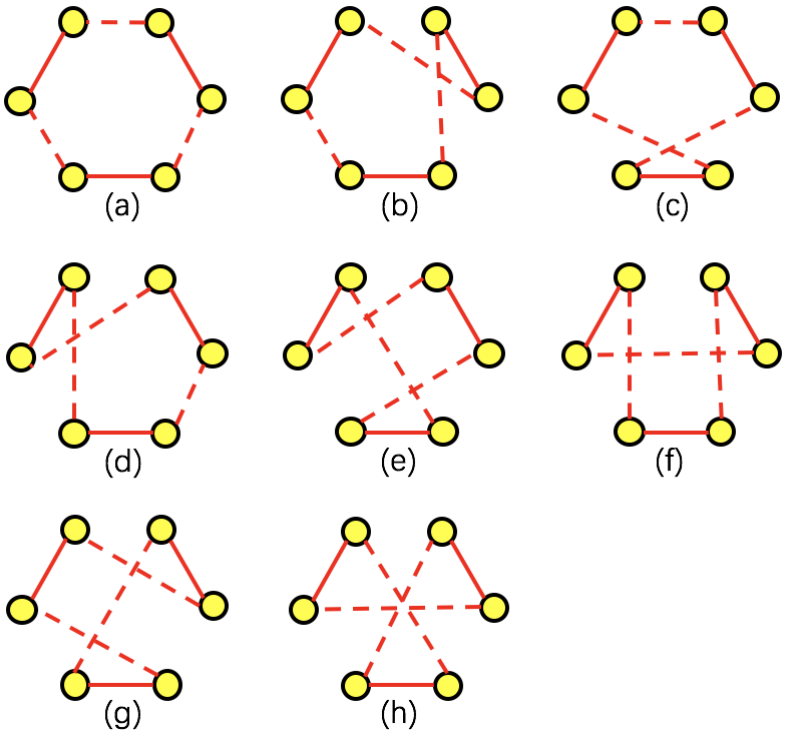

Figure 5.2. All the possible combination cases of a 3-opt move

cases of a 3-opt move is shown in Figure 5.2. In tour (a), the three edges to break are shown in dotted line. Among all the possible recombination, it happens when one edge keeps the original connection and the other two are reconnected, in this case we can obtain the tours (b), (c) and (d). These three tours are equivalent to the results of a single 2-opt move, which picks two customers of the given tour (a) and reverse the path between two chosen customers. By continue doing 2-opt moves on tours (b), (c) and (d) we can obtain tours (e), (f) and (g). Similarly, tour (h) can be obtained by doing three subsequent 2-opt moves. The complexity of a 3-opt execution for a tour with n customers is $O(n^3)$.

Some theoretical results about the number of complete tours present in the neighbourhood generated by the 3-opt move are considered.

Given a complete tour τ with n customers, with 0 being the depot and destination, there are $n+1$ edges in total. In a single 3-opt move, when removing 3 edges there are 7 alternative tours that can be generated. Theoretically, the total number of 3-opt moves that can be operated in tour τ is determined by the number of combinations to pick 3 edges out of $n+1$ edges where no two adjacent edges can be selected. This is similar to the famous problem of selecting k people from m people sitting around a round table, where no two adjacent people can be selected [7]. The problem was initially introduced in Lucas [43]. The formula for the total number of ways to select is already known as:

$$C^{k}_{m-k+1} - C^{k-2}_{m-k-1} \tag{5.1}$$

By taking $k = 3$ and $m = n+1$ into the formula (5.1), the total number of 3-opt moves that can be operated in τ is given by:

$$C_{n-1}^{3} - (n-3) \tag{5.2}$$

With straightforward expansion of formula (5.2) and multiply it by 7, the total number of complete tours contained in the 3-opt neighbourhood $\mathcal{N}(\tau)$ is then determined by:

$$|\mathcal{N}(\tau)| = \frac{7(n^3 - 6n^2 + 5n + 12)}{6} \tag{5.3}$$

Note that the 3-opt operator works for instances with dimension $n \geq 5$.

With the help of formula (5.1), more choices for the operators such as 4-opt up to k-opt can also be considered with intuitive theoretical results. However, due to the complexity of these k-opt operators with $k \geq 3$, in this Chapter we will only focus on the 2-opt and 3-opt operator for the Tabu Search algorithm, since being simple and efficient is the original intention.

When the memory structure described in Section 5.3 is applied on top of the 3-opt heuristic, we use the same memory as for the 2-opt heuristic, which forbids the customers in the position that have been swapped to obtain the current best tour in the neighborhood. Since there are three positions in the 3-opt move, each time when a better solution is found, three customers will be forbidden for a certain amount of time (iterations).

As mentioned in Section 5.3, a main parameter to tune in the Tabu Search is the length of the Tabu List. When a 3-opt operator is embedded in the TS algorithm, l triples (instead of pairs) of customers are forbidden to move at the same time when the Tabu List is full, thus smaller value of the Tabu tenure should be considered. Detailed experimental results about the tuning of parameter for the 3-opt approach can be found in Section 5.6.3.

5.5 The Complete Tabu Search Algorithm

First of all, we propose a general framework for the local search procedure and the restart conditions we use for the Tabu Search algorithm. It is a Random Restart Local Search algorithm slightly different from the RRLS algorithm in Chapter 4, but more adaptable to the Tabu Search structure. The precise differences will be detailed later with explanations. The pseudo code of the algorithm is outlined in Algorithm 3. By taking experimental results of this algorithm as a reference, we are able to understand how the local search approaches can benefit from the use of a Tabu Search paradigm.

The input of the algorithm is a POP instance. The initial tour τ_0 is a *complete tour* generated by a k-Nearest-Random-Insertion method (see Section 4.4) with $k = 2$. To be more intuitive, while constructing a solution, a random customer is iteratively picked between the two nearest customers from the current customer. By using the Monte Carlo evaluator (see Chapter 3), a corresponding feasible solution $\sigma(\tau_0)$ is obtained together with an evaluation value of the objective function $u(\sigma(\tau_0))$. We then apply a local search (2-opt or 3-opt in our case) to the initial solution obtaining the group of neighboring tours $\mathcal{N}(\tau_0)$. For each tour of the neighborhood, we approximate the value of the objective function (of the feasible solution) by using the Monte Carlo evaluator. The current best incumbent tour τ_{inc} is updated consequently. After comparing all the tours in the neighborhood, a new group of neighboring tours will be generated from the best tour τ_{inc} in the previous neighborhood. When no improving tour can be

Algorithm 3: Random Restart Local Search (TS)

```
iterations = 0;
τ_0 = k_Nearest_Random_Insertion();
τ_best = τ_current = τ_0;
while runtime ≤ max_runtime do
    τ_inc = −∞;
    for τ ∈ N(τ_current) do
        if u(σ(τ)) > u(σ(τ_inc)) then
            τ_inc = τ;
        end
    end
    τ_current = τ_inc;
    if u(σ(τ_inc)) > u(σ(τ_best)) then
        τ_best = τ_inc;
        iterations = 0;
    end
    iterations = iterations + 1;
    if iterations ≥ max_non_improving_iterations then
        τ_current = k_Nearest_Random_Insertion();
        iterations = 0;
    end
end
return σ(τ_best);
```

found after a certain number of iterations ($max\_non\_improving\_iterations$, set to 50 in this work), a new tour is generated again by k-Nearest-Random-Insertion method and the process is restarted until a given time limit is reached.

The main difference between Algorithm 3 and the RRLS algorithm described in Algorithm 2 (see Chapter 4) is that, in Algorithm 2 the new neighborhood is generated whenever a better solution is found. The restarting condition is different as a consequence, the searching process restarts when no improving moves can be found in the neighbourhood in Algorithm 2. In Algorithm 3, the new neighborhood is generated after searching the whole current neighborhood. This more consistent way is introduced to be fully aligned with the Tabu Search logic and to have a clear vision of the role of the Tabu memory. Pseudo codes for the full Tabu Search algorithm with 2-opt or 3-opt operator are outlined in Algorithm 4 and Algorithm 5.

The input of the algorithms is always a POP instance. A *complete tour* τ_0 is generated as initial tour by k-Nearest Random Insertion method and is evaluated by the Monte Carlo evaluator. A corresponding feasible solution $\sigma(\tau_0)$ is obtained by deciding to visit the customers that can be visited before the deadline. We then apply a local search (see Section 4.1 and Section 5.4) to τ_0, by considering a group of neighboring tours $\mathcal{N}(\tau_0)$. A memory structure (see Section 5.3) is used to prevent the local search from re-visiting the same tours that has been visited recently. For each neighboring tour $\tau_i \in \mathcal{N}(\tau_0)$ that has not been visited recently, its objective function value $u(\tau_i)$, the corresponding feasible solution $\sigma(\tau_i)$ and the objective function value $u(\sigma(\tau_i))$ are calculated by the Monte Carlo evaluator, and the incumbent solution τ_{inc} is updated consequently.

Algorithm 4: Tabu Search algorithm with a 2-opt operator

```
Tabulist = [];
iterations = 0;
τ_0 = k_Nearest_Random_Insertion();
τ_best = τ_current = τ_0;
while runtime ≤ max_runtime do
    τ_inc = −∞;
    for j = 1, ..., |σ(τ_current)| + 1 do
        for k = j + 1, ..., |σ(τ_current)| + 1 do
            if (τ_current[j] ∉ Tabulist) & (τ_current[k] ∉ Tabulist) then
                τ_i = 2-opt(τ_current) ;
                if u(σ(τ_i)) > u(σ(τ_inc)) then
                    τ_inc = τ_i;
                    (c_1, c_2) = (τ_current[j], τ_current[k]);
                end
            end
        end
    end
    τ_current = τ_inc;
    for j = |σ(τ_current)| + 1, ..., n + 1 do
        for k = j + 1, ..., n + 1 do
            τ_i = 2-opt(τ_current);
            if u(τ_i) > u(τ_inc) then
                τ_inc = τ_i;
            end
        end
    end
    if u(σ(τ_inc)) > u(σ(τ_best)) then
        τ_best = τ_inc;
        iterations = 0;
    end
    iterations = iterations + 1;
    if iterations ≥ max_non_improving_iterations then
        τ_current = k_Nearest_Random_Insertion();
        iterations = 0;
    end
    Insert (c_1, c_2) in the Tabulist ;
    if Tabulist is full then
        remove oldest pair;
    end
end
return σ(τ_best);
```

Algorithm 5: Tabu Search algorithm with a 3-opt operator

```
Tabulist = [];
iterations = 0;
τ_0 = k_Nearest_Random_Insertion();
τ_best = τ_current = τ_0;
while runtime ≤ max_runtime do
    τ_inc = −∞;
    for τ ∈ N(τ_current) do
        if (τ[i], τ[j], τ[k] ∉ Tabulist) then
            if u(σ(τ)) > u(σ(τ_inc)) then
                τ_inc = τ;
                (c_1, c_2, c_3) = (τ[i], τ[j], τ[k]);
            end
        end
    end
    τ_current = τ_inc;
    if u(σ(τ_inc)) > u(σ(τ_best)) then
        τ_best = τ_inc;
        iterations = 0;
    end
    iterations = iterations + 1;
    if iterations ≥ max_non_improving_iterations then
        τ_current = k_Nearest_Random_Insertion();
        iterations = 0;
    end
    Insert (c_1, c_2, c_3) in the Tabulist ;
    if Tabulist is full then
        remove oldest triple;
    end
end
return σ(τ_best);
```

In Algorithm 4, the local search proceeds in two steps. We first swap between the customers that can be visited before the deadline and compare the objective function value $u(\sigma(\tau_i))$. The pair of customers $(\tau_i[j], \tau_i[k])$ in the position j and k that have been swapped to obtain the incumbent solution τ_{inc} are saved consequently. After finish searching among the customers that can be visited before the deadline, a local search procedure without memory structure will run for the rest of the customers who are visited after the deadline incurred. The value of $u(\tau_i)$ is used here to discriminate between tours that differ only for the customers appearing after the deadline. In this way, when two different complete tours are associated to the same feasible solution, priority is given to the tour with a better tail after the customers included in the feasible solution. This strategy is very important in the economy of the metaheuristic strategy we developed, since promising chunks of tours are prepared in complete tours after the end of the respective feasible solution, and these parts are used by the 2-opt component to improve the feasible solutions in subsequent iterations.

After comparing all applicable tours in the neighborhood, a new group of neighboring tours $\mathcal{N}(\tau_{inc})$ will be generated from the incumbent solution τ_{inc}, and attributes of τ_{inc} are stored in the memory structure. When no improving tour can be found after a certain number of iterations (set to 50 in this work), a new random tour is generated and the process is restarted until a given time limit is reached.

Some theoretical results about the number of tours left in the 2-opt neighbourhood when a Tabu List is implemented, are presented.

Given a complete tour τ with $q = |\sigma(\tau)|$ customers in the associated feasible solution $\sigma(\tau)$, and a Tabu List of length l containing $2 \cdot l$ distinct customers, the total number of meaningful tours left in the 2-opt neighbourhood $\mathcal{N}^l(\sigma(\tau))$ is given by the number of combinations of picking 2 customers out of $q-2l$ customers:

$$L = \left|\mathcal{N}^l(\sigma(\tau))\right| = \frac{(q-2l)(q-2l-1)}{2} \tag{5.4}$$

L represents the number of meaningful tours left in the neighborhood when a Tabu List of length l is full. In general we expect $L >> 0$ in order to ensure that the search space is not (almost) empty, and the search can proceed fruitfully.

The actual value $q = |\sigma(\tau)|$ for the certain complete tour τ is given by the Monte Carlo evaluator. Given the value q and a Tabu List length l, L can be easily calculated, and the number of meaningful tours in the 2-opt neighbourhood can be easily checked. A closed formula for the total number of non-forbidden tours of the 2-opt neighbourhood of a specified complete tour τ forbidden by the Tabu mechanism is therefore given by:

$$|\mathcal{N}(\sigma(\tau))| - \left|\mathcal{N}^l(\sigma(\tau))\right| = (2q-2l-1) \cdot l \tag{5.5}$$

The logic of the Tabu Search algorithm is the same in Algorithm 5 with the 3-opt operator. We denote tours in the neighborhood of $\tau_{current}$ as τ, with i, j, and k being the position of the customers being swapped to obtain τ. The triple of customers $(\tau[i], \tau[j], \tau[k])$ in the relevant position i, j and k that have been swapped to obtain the incumbent solution τ_{inc} are saved consequently. Different from the algorithm for the 2-opt operator, the memory structure is not only applied on the customers visited before the deadline. For the reason that in a 3-opt move, even if all the broken edges are on the route visited after the deadline, the reconnecting of the

edges still could generate meaningful new tours. Therefore, the memory structure is working during the entire searching process. Theoretical computations on the number of tours left in the 3-opt neighborhood are not intuitive, thus we did not do it. Since l triples of customers are forbidden to move in the 3-opt heuristic instead of l pairs for the 2-opt heuristic, smaller value of the Tabu tenure should be considered.

5.6 Experimental Results

In this section we test the performance of the TS algorithm. First, we run the Random Restart Local Search algorithm designed for TS (without memory structure) with 2-opt and 3-opt operator. Then we tune the length of the Tabu List separately for the 2-opt and 3-opt heuristic in the TS algorithm. Finally the TS algorithm is compared with other heuristic methods for the POP.

The testing environment is computer equipped with an Intel Core i7-9700K processor running at 3.6 GHz and with 16GB of RAM. A single thread was used for all the experiments. The algorithm we propose was implemented in C++.

5.6.1 Random Restart Local Search TS: 2-opt and 3-opt

The benchmark instances used for experiments are those from Angelelli et al. [1] described in Section 3.1.2. A maximum computation time of 600 seconds is allowed to each run in all the experiments reported in this section, with number of samples $s = 50$ and $C = 0.001$ as the coefficient of the objective function (2.2).

Similar as in Section 4.4.2, the 264 instances are divided into different groups based on their characteristics. According to the dimension, there are 84 small instances with $n < 30$, 84 medium instances with $30 \leq n < 50$, and 96 large instances with $n \geq 50$. In terms of the deadline, there are also three groups. For Prize type and Probability type, there are two groups with 132 instances for each.

First, we run the TS algorithm with Tabu length $l = 0$ with 2-opt and 3-opt operators. Table 5.1 shows the results obtained by Algorithm 3 presented in Section 5.5. For each of the characteristic group, the average error together with the corresponding number of best-known solutions found ($best\_sol$, comparing to the best solutions disclosed in [1]) and the average computation time are reported. The computing time reported are the time at which the final solutions are found. The last row in the table contains the average results over all the instances.

In Table 5.1 we can observe that the average error of 2-opt and 3-opt heuristics are similar. However, the 2-opt heuristic finds more best-known solutions, taking advantage of more restarts and faster runs. The 3-opt requires shorter computing time in this perspective.

In Table 5.1 we can also observe that the 2-opt heuristic is better for a variety of instance characteristics such as small to medium dimension ($n < 50$), small to medium deadlines, random prizes ($\pi_i = random$) and equal probabilities ($p_i = 0.5$). While the 3-opt heuristic is better than 2-opt for certain instance characteristics. By using the 3-opt heuristics, the average error is smaller for instances with large dimension $n \geq 50$, long deadlines, small prizes ($\pi_i = 1$)

Table 5.1. RRLS - Comparison between 3-opt and 2-opt heuristics for the POP

operators / instances	**2-opt**			**3-opt**		
	error(%)	best_sol	time(s)	error(%)	best_sol	time(s)
$n < 30$	0.018	61	1.29	0.112	60	1.75
$30 \leq n < 50$	1.169	26	43.24	1.677	22	20.16
$n \geq 50$	9.056	15	72.29	8.529	14	55.47
$1/4T_{max}$	1.936	50	20.96	2.160	42	13.52
$2/4T_{max}$	4.711	34	39.95	4.996	29	27.29
$3/4T_{max}$	4.365	18	60.46	3.855	25	40.62
$\pi_i = 1$	5.022	54	29.39	4.502	52	31.02
$\pi_i = random$	2.320	48	51.52	2.839	44	23.26
$p_i = 0.5$	3.226	49	42.14	3.580	45	27.05
$p_i = random$	4.115	53	38.77	3.761	51	27.23
ALL	**3.671**	**102**	**40.46**	**3.670**	**96**	**27.14**

and random probabilities ($p_i = random$). The 3-opt heuristic works noticeable better on instances with long deadlines: more best-known solutions are found and in a shorter computing time comparing to the results obtained by 2-opt heuristic.

5.6.2 Tuning for the length of the Tabu List: the 2-opt case

In this section we consider the tuning of the Tabu list length for the case where 2-opt is used as an inner local search.

With the same data sets as described in Section 3.1.2, the experiments are run with a maximum solving time of 180s for each instance, with number of samples $s = 50$.

As alternatives for the length of the Tabu List l, in order to find a tuning that performs well in general case, four flexible values $l = f \cdot n$ are tested, with $f \in \{0.02,0.05,0.1,0.15\}$ being a coefficient and n being the number of customers of the instance under investigation (note that the number of customers ranges between 15 and 97 for the 264 instances from [1]).

For each final solution σ retrieved by the algorithm, its objective function value is evaluated by substituting the exact evaluation formulas (2.3) and (2.4) into the equation:

$$u(\sigma) = \sum_{k=1}^{q} \pi_k p_k - C \cdot \sum_{h=0}^{q} [\pi_h \sum_{k=h+1}^{q+1} (t_{hk} \cdot \pi_k \cdot \prod_{j=h+1}^{k-1} (1-\pi_j))] \tag{5.6}$$

where it is assumed that $\prod_{j=h+1}^{k-1}(1-\pi_j) = 1$ whenever $h+1 > k-1$.

The objective function value of the best solution retrieved with a certain Tabu List length l is denoted as X_l. Since an exact solution is not available for all the instances considered, the best solution reported in [1] for each instance is considered as a reference value X_{ref}. We calculate, for each setting l, the error in percentage by computing the relative difference between X_l and X_{ref}:

$$error(\%) = \left|X_l - X_{ref}\right| \cdot \frac{100}{X_{ref}} \tag{5.7}$$

Table 5.2. Average results for the TS algorithm (2-opt) with different tabu lengths

l / instances	0.02n		0.05n		0.1n		0.15n	
	error(%)	time(s)	error(%)	time(s)	error(%)	time(s)	error(%)	time(s)
$n < 30$	0.059	0.16	0.055	0.15	0.040	0.21	0.029	0.24
$30 \leq n < 50$	1.683	6.25	1.618	5.37	1.707	4.46	2.201	4.31
$n \geq 50$	6.172	10.82	6.012	11.10	6.871	6.73	8.420	4.26
$1/4T_{max}$	1.313	3.59	1.225	2.89	1.803	1.86	2.829	1.44
$2/4T_{max}$	3.626	6.31	3.507	6.31	3.681	4.64	4.522	3.19
$3/4T_{max}$	3.458	8.04	3.425	8.17	3.679	5.30	3.964	4.36
$\pi_i = 1$	2.724	5.78	2.721	5.62	3.053	3.91	3.834	2.95
$\pi_i = random$	2.874	6.18	2.716	5.96	3.056	3.96	3.710	3.05
$p_i = 0.5$	2.720	5.97	2.583	5.80	3.0345	4.24	3.662	2.91
$p_i = random$	2.878	5.99	2.854	5.78	3.074	3.63	3.881	3.08
ALL	**2.800**	**5.98**	**2.719**	**5.79**	**3.054**	**3.93**	**3.772**	**3.00**

In the experiment, for each Tabu List length l, we report the average and best error over 5 runs.

Table 5.2 shows the average error and computation time over 5 runs for different values of l. The last row of the table shows the average results for all the 264 instances. In Table 5.2 we observe that the choice of the Tabu List length l is different for each dimension group. For small instances with $n < 30$, the best result is obtained with $l = 0.15n$, for medium instances with $30 \leq n < 50$ and large instances with $n \geq 50$, the best result is obtained with $l = 0.05n$. It is worth noting that, when $l = 0.02n$, the Tabu List length is 0 for some of the small instances ($n < 30$). In this case the TS mechanism simply does not get activated. Apart from this, smaller values of l appear more indicated for larger n. This shows that the choice of l might be independent of n. In particular, fixed values like 2 or 3 seem to be a good choice according to the emerging trend.

As mentioned in Section 5.5, the memory structure for the 2-opt heuristic works only on the q customers visited before the deadline D. The value of q is mainly influenced by the instance size and deadline type. From the table we can also observe that the algorithm performs differently on relevant subgroups. Among the given POP benchmarks, the actual value of q normally varies from 4 to 21 for small instances, from 7 to 40 for medium instances, and from 1 to 85 for large instances. Considering equations (4.2) and (5.5), when appropriate values of Tabu List lengths are taken (e.g. $l = 2$ from the previous conclusion of Table 5.2), 57.6% of meaningful complete tours on average are forbidden by the Tabu mechanism for small instances, 32.5% for medium instances, and 20.2% for large instances. Theoretical and empirical results are therefore consistent.

In general, a smaller value of Tabu List length requires longer computation time, and the average computing time is less that 10 seconds for all settings. This shows that the TS algorithm with 2-opt heuristic is able to give a relatively good result for a POP instance within a very short time.

In Table 5.3 we present the best results over 5 runs obtained by the TS algorithm with 2-opt operator. For each characteristic group, the error with the corresponding number of best known solutions found *bs* (comparing to the best solutions disclosed in [1]) and the computation time

Table 5.3. Best results obtained by TS algorithm (2-opt) for different tabu lengths

l / instances	0.02n			0.05n			0.1n			0.15n		
	error	*bs*	time	error	*bs*	time	error	*bs*	time	error	*bs*	time
$n < 30$	0.049	74	0.10	0.024	77	0.09	0.024	73	0.08	0.024	79	0.12
$30 \leq n < 50$	1.207	41	5.99	1.032	39	6.29	0.964	41	4.99	1.491	32	4.03
$n \geq 50$	4.337	16	11.00	4.255	20	11.66	5.016	12	8.73	5.894	12	5.84
$1/4T_{max}$	0.703	60	3.57	0.631	66	2.70	0.944	62	2.25	1.711	55	1.52
$2/4T_{max}$	2.626	42	6.33	2.498	42	7.27	2.676	38	5.65	3.392	39	3.86
$3/4T_{max}$	2.602	29	7.92	2.521	28	8.84	2.795	26	6.47	2.773	29	4.95
$\pi_i = 1$	1.696	75	6.78	1.784	77	6.37	1.935	73	4.82	2.443	69	3.87
$\pi_i = random$	2.258	56	5.10	1.983	59	6.16	2.341	53	4.75	2.807	54	3.02
$p_i = 0.5$	1.978	67	5.99	1.755	69	6.41	2.101	65	5.33	2.591	61	3.37
$p_i = random$	1.975	64	5.89	2.012	67	6.13	2.176	61	4.24	2.660	62	3.53
ALL	**1.977**	**131**	**5.94**	**1.883**	**136**	**6.27**	**2.138**	**126**	**4.79**	**2.625**	**123**	**3.45**

are reported. Again, the last row contains the average results over all the instances. The best choice of Tabu List length is again slightly different for different characteristics. The algorithm performs extremely well on small instances with $n < 30$, with $l = 0.15n$ being the best with an error of 0.024% and 79 out of 84 instances solved to optimal. For medium size instances with dimension $30 \leq n < 50$, $l = 0.1n$ performs the best, and the instances can be solved with an average error of 0.964%. For large instances with dimension $n \geq 50$, $l = 0.05n$ performs the best with an error of 4.255%. For all the rest of characteristics $l = 0.05n$ performs the best, with 30%-40% of the meaningful complete tours forbidden. Therefore, at this stage we choose $l = 0.05n$ as representative of the TS algorithm (with 2-opt operator) for the comparison with other heuristic methods from the literature provided in Section 5.6.4.

5.6.3 Tuning for the length of the Tabu List: the 3-opt case

In this section, we consider the tuning of the Tabu list length for the case where 3-opt is used as an inner local search.

Similar to what done in Section 5.6.2, four flexible values $\{0.01n, 0.05n, 0.1n, 0.15n\}$ are proposed as alternatives for the length of the Tabu List, with n being the dimension of the instances. Note that the smallest candidate of Tabu list length for 2-opt was $0.02n$. As explained in Section 5.4, the length of the Tabu List should be smaller for the 3-opt, therefore we take $0.01n$ as the smallest candidate. We report results over one run only.

The results of the 3-opt heuristic with different lengths of the Tabu List is shown in Table 5.4. By comparing results in Table 5.4 and Table 5.1 and Table 5.2 we can observe that Tabu Search does not offer impressive results with the 3-opt heuristic as it was happening with the 2-opt case, but it can improve the results for certain instance characteristics. For instances with small dimension $n < 30$, an improvement on both error and computing time can be observed for length $l = 0.1n$. For medium size instances with $30 \leq n < 50$, improvements on quality of the solutions can be observed for length $l = 0.01n$, but the computing time increases. For large instances with $n \geq 50$, no improvements can be observed when the Tabu mechanism is activated.

In the perspective of Deadline values, there is no improvement on instances with short deadlines. The Tabu Search mechanism improves the quality of solution with length $l = 0.01n$

Table 5.4. Average results for the TS algorithm (3-opt) with different tabu lengths

l / instances	0.01n			0.05n			0.1n			0.15n		
	error	*bs*	time	error	*bs*	time	error	*bs*	time	error	*bs*	time
$n < 30$	0.07	57	2.1	0.10	55	1.6	0.08	59	1.3	0.05	59	2.1
$30 \leq n < 50$	1.53	25	45.9	2.16	23	19.8	2.14	16	20.0	2.24	18	20.0
$n \geq 50$	9.80	12	124.9	10.60	11	50.4	10.26	9	47.0	11.09	10	44.5
$1/4T_{max}$	4.37	41	24.9	3.62	41	11.6	2.83	37	13.7	3.93	39	12.5
$2/4T_{max}$	4.43	31	63.8	5.31	24	26.3	5.65	25	25.8	5.39	27	23.2
$3/4T_{max}$	3.42	22	93.2	4.79	24	37.5	4.83	22	32.1	4.97	21	33.9
$\pi_i = 1$	5.93	55	63.0	5.57	52	28.7	5.23	45	27.3	5.95	47	24.1
$\pi_i = random$	2.22	39	58.2	3.57	37	21.6	3.64	39	20.4	3.58	40	22.4
$p_i = 0.5$	3.96	45	57.6	4.40	46	25.1	4.36	41	23.8	4.55	43	23.4
$p_i = random$	4.18	49	63.7	4.75	43	25.1	4.50	43	23.9	4.98	44	23.1
ALL	4.07	94	60.6	4.57	89	25.1	4.43	84	23.8	4.76	87	23.2

on instances with medium to long deadlines, but the computing time increases as well. There is not much improvements for instances with different Prize type or Probability type: the error of the solution is slightly improved with length $l = 0.01n$ for instances with random probabilities, but the number of best known solution found decreased and the computing time increased as well, therefore it is not a considerable choice.

In general, from Table 5.4 we can conclude that Tabu Search improves both the quality and the speed of the 3-opt heuristic on instances with medium to long deadlines. It can also improve the quality of solutions for instances with small dimension and random probabilities.

A further study can be carried out on all the 264 POP instances. We compare the best results obtained by 3-opt with Tabu Search and 2-opt with Tabu Search for each instance. The results show that the 3-opt heuristic achieves better results than the 2-opt on 42 instances in terms of accuracy. The 42 instances are listed out in Table 5.5. The majority (90%) of these instances are of medium to long deadline types. Among the 42 improved instances, there are also 9 instances that have reached the best-known solution that have not been reached by the 2-opt heuristic. However, the 3-opt heuristic is slower than 2-opt in most of the cases. A detailed comparison between the 2-opt and 3-opt operators within the Tabu Search algorithm on all the 264 instances can be found in Appendix. Among the 222 instances left, there are 116 instances on which the 2-opt heuristic achieves better results than the 3-opt, and 106 instances that the 2-opt and 3-opt heuristics are equally good in terms of accuracy, but the 2-opt heuristic is faster in computing speed. Therefore, we can conclude that the 3-opt heuristic and the 2-opt heuristic have different adaptability on the POP instances. The 2-opt heuristic works better on the majority of the POP instances generated in [1], while the 3-opt heuristic works better especially on instances with medium to long deadlines.

We conclude that 2-opt is more suitable to be used within the TS algorithm being faster and taking more advantage of the memory structure. For this reason we will only consider 2-opt in the remainder of the chapter.

Table 5.5. The 42 instances on which the 3-opt heuristic achieves more accurate results than 2-opt for the TS algorithm

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
att48FSTCII_q3_g2_p2	4.64	12.85	1.55	5.62
bayg29FSTCII_q3_g1_p2	0.41	4.59	0.00	3.32
berlin52FSTCII_q2_g1_p2	0.57	15.30	0.19	79.05
berlin52FSTCII_q3_g2_p2	2.90	9.97	2.75	32.72
brazil58FSTCII_q1_g2_p2	2.73	0.38	1.36	10.51
brazil58FSTCII_q3_g2_p1	0.003	17.14	0.00	101.70
brazil58FSTCII_q3_g2_p2	0.01	15.34	0.00	32.23
dantzig42FSTCII_q3_g1_p1	0.002	13.65	0.00	3.01
dantzig42FSTCII_q3_g1_p2	1.76	2.52	0.59	4.72
dantzig42FSTCII_q3_g2_p1	5.18	10.2	3.51	34.45
eil51FSTCII_q2_g1_p2	1.52	7.41	0.00	5.13
eil76FSTCII_q2_g1_p1	6.69	21.14	4.42	27.44
eil76FSTCII_q3_g1_p1	4.81	8.91	1.58	75.16
eil76FSTCII_q3_g2_p1	6.94	16.20	4.40	112.88
eil76FSTCII_q3_g2_p2	5.36	7.82	3.44	27.96
gr24FSTCII_q3_g2_p1	0.75	0.5	0.32	0.4
gr24FSTCII_q3_g2_p2	1.28	0.36	0.00	12.92
gr48FSTCII_q1_g2_p2	3.67	3.78	1.80	14.71
gr48FSTCII_q3_g1_p2	3.85	27.99	2.95	61.23
gr48FSTCII_q3_g2_p2	3.18	9.35	2.07	61.46
gr96FSTCII_q3_g2_p1	4.16	10.90	2.90	147.97
gr96FSTCII_q3_g2_p2	5.15	8.04	4.66	174.93
hk48FSTCII_q3_g1_p1	2.91	12.47	0.86	50.06
hk48FSTCII_q3_g1_p2	1.35	11.29	1.27	27.27
pr76FSTCII_q2_g2_p1	8.10	25.72	7.87	56.70
rat99FSTCII_q1_g2_p2	6.67	4.5	0.00	21.57
rat99FSTCII_q2_g2_p2	18.27	5.03	16.68	79.44
rat99FSTCII_q3_g1_p1	11.69	24.19	10.43	140.12
rat99FSTCII_q3_g1_p2	12.03	12.27	9.13	175.53
rat99FSTCII_q3_g2_p1	11.47	5.11	8.43	51.75
rat99FSTCII_q3_g2_p2	12.41	11.15	9.44	54.31
st70FSTCII_q2_g1_p2	5.62	10.34	1.19	70.98
st70FSTCII_q2_g2_p1	4.19	17.48	1.37	17.27
st70FSTCII_q2_g2_p2	5.66	13.86	4.24	24.01
st70FSTCII_q3_g1_p1	3.57	6.63	1.76	74.92
st70FSTCII_q3_g1_p2	3.73	3.94	2.63	43.08
st70FSTCII_q3_g2_p1	5.30	23.04	4.00	14.59
st70FSTCII_q3_g2_p2	5.23	33.05	3.78	51.02
swiss42FSTCII_q2_g2_p2	0.19	19.59	0.17	103.92
swiss42FSTCII_q3_g1_p2	1.99	6.15	0.73	43.47
swiss42FSTCII_q3_g2_p1	1.45	3.68	0.27	7.02
swiss42FSTCII_q3_g2_p2	0.63	26.65	0.41	42.71

Table 5.6. Comparison between different heuristics for the POP

methods / instances	**sChain**[1]			**sPath**[1]			**s2Path**[1]			**TS**		
	error	*bs*	time	error	*bs*	time	error	*bs*	time	error	*bs*	time
$n < 30$	2.52	63	21	3.93	56	120	2.72	60	168	0.02	77	0.09
$30 \leq n < 50$	0.38	49	27	0.75	37	6	1.05	34	5	1.03	39	6.29
$n \geq 50$	0.77	41	223	1.17	26	181	1.34	23	127	4.26	20	11.66
$1/4T_{max}$	1.02	56	40	1.25	50	11	1.57	48	14	0.63	66	2.70
$2/4T_{max}$	0.23	54	22	0.63	37	94	0.58	39	17	2.50	42	7.27
$3/4T_{max}$	0.35	43	232	0.61	32	209	0.65	30	288	2.52	28	8.84
$\pi_i = 1$	0.78	65	136	0.88	58	174	1.11	54	153	1.78	77	6.37
$\pi_i = random$	0.29	88	60	0.78	61	36	0.75	63	60	1.98	59	6.16
$p_i = 0.5$	0.37	82	126	0.72	58	127	0.87	61	123	1.75	69	6.41
$p_i = random$	0.70	71	71	0.94	61	82	1.00	56	89	2.01	67	6.13
ALL	**0.53**	**153**	**98**	**0.83**	**119**	**104**	**0.93**	**117**	**107**	**1.88**	**136**	**6.27**

5.6.4 Comparison of the TS algorithm with other methods from the literature

In this section we compare the general performance of the TS algorithm using the 2-opt heuristic with other heuristic methods for the POP. For each method we present the error in percentage (again with reference to the best results published in [1]), the number of best solutions found and the computing time for each of the characteristic groups, as well as the aggregated results for all the 264 instances.

Three matheuristics based on a branch-and-cut framework are proposed by Angelelli et al. [1]: Smart-Chain (**sChain**), Smart-Path (**sPath**), and Smart-Two-ways-path (**s2Path**). The Chain, Path and Two-ways-path are alternative branching strategies proposed for the POP focusing on chains starting in 0 or ending in $n+1$. The corresponding matheuristics are branch-and-cut algorithms with heuristic branching and variable fixing to reduce the solution space. We refer the interested reader to [1] for complete details of the methods. Up to our knowledge, these are the only relevant methods available in the literature.

The experiments presented in [1] are carried out on a Intel Xeon E5-1650 processor, 3.50GHz and 16GB of RAM with a max computation time of 7200 seconds. In order to have here a fair comparison, we normalized the computation times from [1], obtained on a <Xeon E5-1650 processor, 3.50GHz> to our machine, which according to CPU Benchmarks[1] is faster by a factor of about 38%. The results are reported in Table 5.6, where the results of **TS** with 2-opt operator are obtained with $l = 0.05n$ (see Section 5.6.2) and a time limit again of 180 seconds.

From Table 5.6 we can observe that the TS algorithm performs very well on instances with small dimension and short deadline. For instances with dimension less than 30, the TS algorithm retrieves more best known solutions with much shorter times when compared to the other methods. The three Matheuristic methods takes from 21 seconds up to 168 seconds, among which the smallest average error is 2.52%. The TS algorithm takes only 0.09 seconds on average, 77 out of 84 instances are solved to the best known solution, with an average error of 0.02%. For medium size instances, the TS algorithm is competitive with the other methods on

[1] https://www.cpubenchmark.net/

accuracy and computing time. sChain method performs the best on error (0.38%), but also requires longer computing time (27s). The TS algorithm has an average error of 1.03% in 6.29s, which is similar to the other two matheuristic methods. For large instances, the matheuristics from [1] are much more accurate (0.77% - 1.34%), but with longer computational times (127s - 223s). The TS algorithm converges in an average of 11.66s with an error of 4.26%. This shows that the 2-opt local search used in TS algorithm might not be sufficient, and stronger local search methods might lead to more accurate results. The TS algorithm remains anyway always much faster.

In general, we can conclude that the TS algorithm is extremely fast, and with small errors. In practical terms, TS could be a precious approach for application where you need to take quick decisions (e.g. within an online decision support system, see [18]).

5.6.5 Results of the TS algorithm on large instances

The TS algorithm considered in this section is again using the 2-opt as inner local search.

The dimension of the 264 POP benchmarks for the previous experiments are up to 96 customers. In order to have a vision of how our approach performs on instances with more than 100 nodes, we generated more POP instances from the TSPLIB95 in the same way as described in [1]. The 24 new POP instances are based on 2 TSP instances of dimension 120 and 535, considering different characteristics (i.e. Deadline type q, Prize type g and Probability type p).

We solve the 24 new POP instances with the TS algorithm presented in Algorithm 4. The computation time limit is set at 1200 seconds for these experiments. To show the converging speed, we calculate the relative difference (%) between the objective function value $u(\sigma_t)$ of the best feasible solution σ_t retrieved at a certain time t and the objective function value $u(\sigma_{1200})$ of the best solution found at time limit, as follows:

$$gap\%(t) = |u(\sigma_t) - u(\sigma_{1200})| \cdot \frac{100}{u(\sigma_{1200})} \tag{5.8}$$

we use $u(\sigma_{1200})$ as a reference because no other known solution is available.

The converging speed of the 24 new POP instances are shown in Figure 5.3, with $gap\%(t)$ of the instance on y-axis and the time t on x-axis. We can observe that the differences drop below 3% within 60s and below 1% within 180s for all the example instances. This shows that the computation converges very fast in general even for large instances, indicating that scalability is promising for the method. We speculate that when larger instances without requirements for extremely high accuracy are considered, the TS is the only approach currently available able to deal with the optimization. The relevant best results found by the TS algorithm are reported in Table 5.7, for completeness.

5.7 Conclusions

In this chapter we have solved the Probabilistic Orienteering Problem by using a Tabu Search algorithm based on a Monte Carlo sampling objective function evaluator. The integration of the two methods allows the design of a simple Tabu Search metaheuristic. We compared the 2-opt heuristic with the 3-opt heuristic as inner local searches for the Tabu Search algorithm.

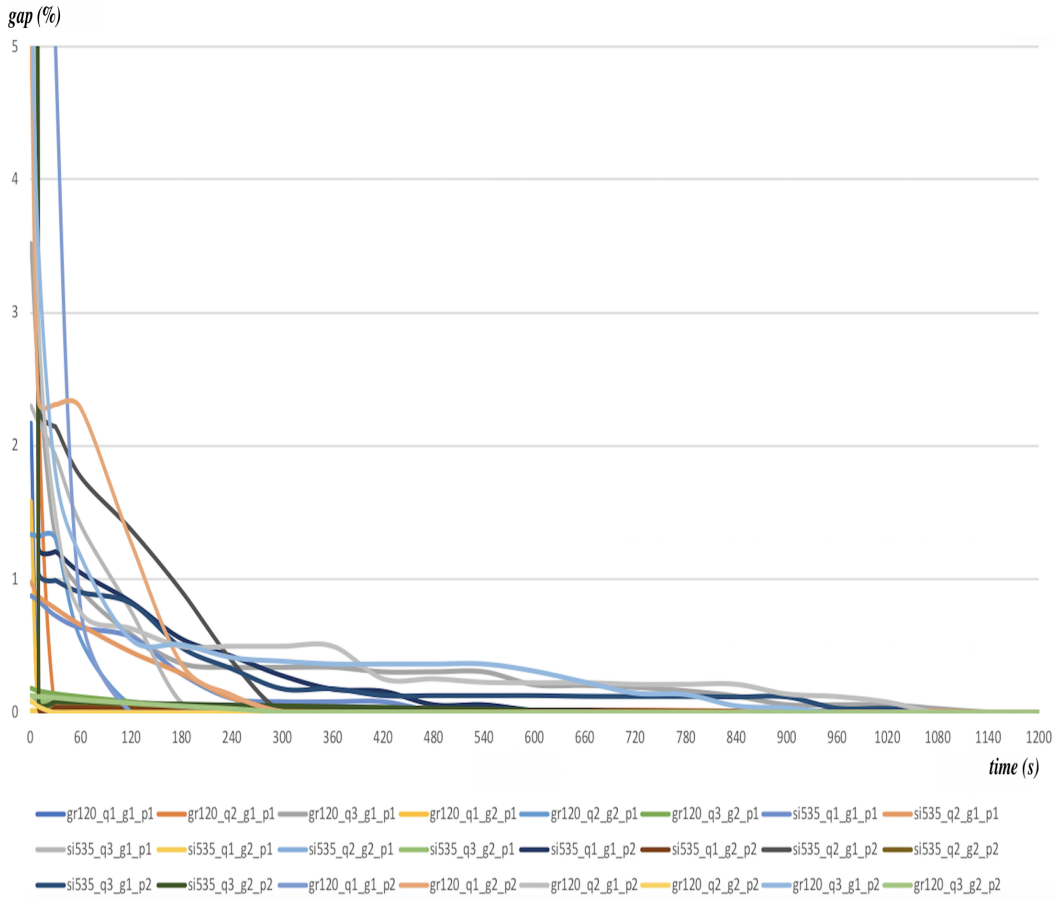

Figure 5.3. Converging speed of the TS algorithm on the new 24 POP instances with different dimension

Table 5.7. Best results found by the TS algorithm for the 24 new POP instances

instance	result	instance	result
gr120_q1_g1_p1	57.68	si535_q1_g1_p1	85.94
gr120_q1_g1_p2	56.20	si535_q1_g1_p2	82.93
gr120_q1_g2_p1	2904.35	si535_q1_g2_p1	4628.16
gr120_q1_g2_p2	2815.31	si535_q1_g2_p2	4502.85
gr120_q2_g1_p1	56.91	si535_q2_g1_p1	84.01
gr120_q2_g1_p2	56.02	si535_q2_g1_p2	82.71
gr120_q2_g2_p1	2936.01	si535_q2_g2_p1	4614.98
gr120_q2_g2_p2	2824.68	si535_q2_g2_p2	4481.41
gr120_q3_g1_p1	57.29	si535_q3_g1_p1	83.90
gr120_q3_g1_p2	55.76	si535_q3_g1_p2	84.87
gr120_q3_g2_p1	2902.13	si535_q3_g2_p1	4628.27
gr120_q3_g2_p2	2768.52	si535_q3_g2_p2	4521.03

Computational studies on the performance of the algorithm both in terms of precision and speed were carried out. We discussed separately the performance of the algorithms for different types of instances, that gave us a vision on how the 2-opt and 3-opt heuristic can improve the speed and efficiency on some classes of POP instances.

With appropriate parameter tuning, the Tabu Search algorithm is overall extremely fast, and with errors comparable with those of considerably more complex heuristics from the literature. This ability of offering high quality solutions in a short calculation time is going to be very practical for applications where real-time decisions need to be made. Furthermore, the algorithm performs also well on the new POP instances we generate with large dimension. The promising scalability opens up more possibilities on the development in both research and applications.

Chapter 6

Parameters Tuning and Machine Learning

In this chapter, we adopt Machine Learning tools to help evaluate efficiently and effectively the solutions of the Probabilistic Orienteering Problem. A crucial parameter in the Monte Carlo evaluator described in Chapter 3 is the number of samples to be used. More samples mean high precision, while less samples mean high speed. In this work, we shifted the problem of selecting the appropriate number of samples to that of finding a trade-off between precision and speed, the problem is then solved with Machine Learning models. The work has appeared in [45].

6.1 Introduction

In Chapter 3 we introduced a Monte Carlo sampling technique that creates a set of alternative realistic scenarios, evaluate the objective function over them and return the average as an approximation for the objective function of the POP. Since the aim of such a Monte Carlo-based evaluator is to be used inside Heuristic methods (see Chapter 4 and Chapter 5), speed is an important factor and a trade-off between speed and precision has be to found. In this chapter we propose techniques to use Machine Learning to predict such a parameter.

Our main contribution of this work is represented by two Machine Learning-based methods: one is more intuitive, the other more driven by technical considerations. By taking the representative features of the POP instances into the neural network-based methods, both methods able to predict the number of samples required to obtain the desired trade-off between speed and precision for the Monte Carlo evaluator.

In the following sections we are going to present the two approaches to predict the correct number of samples and some experimental results to validate the predictors.

6.2 Features Selection

Feature Selection is an important step in the model design. The aim is to select features that influence most the prediction and guide it. With such a process, the modeling tend to be more accurate due to less misleading data. Given the problem characteristics, we need to select the

Table 6.1. Features selected to feed the predictors

Name	Description
Nr. of customers	The number of customers of the given instances (before selecting how many customers to actually visit)
Deadline	The duration of the working day D
Avg Travel Time	The average of the travel times t between pairs of customers (and the depot)
St Dev Travel Time	The standard deviation of the travel times t between pairs of customers (and the depot)
Avg Presence Probability	The average of the presence probability π associated with customers
St Dev Presence Probability	The standard deviation of the presence probability π associated with customers
Avg Prize	The average of the prizes p associated with customers
St Dev Prize	The standard deviation of the prizes p associated with customers

appropriate features as an input for the neural network-based models. The features selected are reported in Table 6.1.

The eight features selected should synthesize the characteristics of each instance in the most possible complete way without introducing more complex features with the risk of confusing the neural networks. The predictors themselves are able to learn and exploit complex interactions of the input parameters if such interactions exist, so it is correct to delegate such a task to the networks themselves.

Understanding eventual mutual interactions among the selected features, as well as having an estimation of how much each of them is important in the economy of the whole learning process, is beyond the scope of the present study, which could be the topic of a future work. Eliminating features that prove to be of marginal utility might lead to a slightly more efficient learning, but in terms of computational performance and global error of the predictor, we do not expect great differences, especially taking into account that the neural networks we propose are not large (see Section 6.3).

6.3 Predicting the best number of samples for an instance

In the Monte Carlo sampling method, more samples means precision, while less samples means speed. In order to predict the best number of samples for an instance, a trade-off between precision and speed has to be found. Therefore, we shift the problem from samples to a concept of satisfaction.

6.3.1 The Concept of Satisfaction

Given an instance I and a generic solution with known objective function value $TOF(I)$ for this instance and a range of possible values for parameter $s \in S = \{10, 11, \ldots, 1000\}$, we define:

- $speed(I,s)$ between 0 (low) and 1 (high) indicating the speed achieved by the Monte Carlo evaluator with s samples on instance I. Let $sec(I,s)$ be the computation time in seconds required by the evaluator on instance I with s samples, then

$$speed(I,s) = \frac{sec(I,1000) - sec(I,s)}{sec(I,1000) - sec(I,10)}$$

- $prec(I,s)$ is a number between 0 (low) and 1 (high) measuring the quality achieved by the Monte Carlo evaluator with s samples on instance I. Let $AOF(I,s)$ be the approximated objective function value returned by the Monte Carlo evaluator on instance I with s samples (average over 10 runs), and $E(I,s) = |AOF(I,s) - TOF(I)|$ be the approximation error, then

$$prec(I,s) = \frac{\max_{x \in S}(E(I,x)) - E(I,s)}{\max_{x \in S}(E(I,x)) - \min_{x \in S}(E(I,x))}$$

Note that the definition of $speed(I,s)$ here is different from the speed evaluation in Chapter 3. With $sec(I,s)$ being the actual computation time of a given instance I with s samples, in Chapter 3 we use the number of evaluations per second to evaluate speed, which actually represents the value of $\frac{1}{sec(I,s)}$. When the number of samples s increases, the number of evaluations per second decreases gradually as a consequence. Differently in this chapter, the definition of $speed(I,s)$ actually represents the value of $-sec(I,s)$ with some coefficients. Therefore, when the number of samples s increases, the $speed(I,s)$ decreases linearly as shown in Figure 6.1. The choice of changing the concept of speed is motivated by the different context.

The values of $prec(I,s)$ for different values of s for an example instance is shown in Figure 6.2. We can observe that when the number of samples s increases, the $prec(I,s)$ increases logarithmically.

We will search for a trade-off between speed and precision that varies from 0 (only speed is important) to 1 (only precision is important) to help find the best number of samples to use. The trade-off is finally defined as the satisfaction level $sat(I,s)$ provided by using s samples on instance I :

$$sat(I,s) = \alpha \cdot speed(I,s) + (1-\alpha) \cdot prec(I,s)$$

where α is a parameter defined by the user and expressing the derived relative importance of speed with respect to precision.

Figure 6.3 shows the relationship between $speed(I,s)$, $prec(I,s)$ and $sat(I,s)$ for an example instance, with $\alpha = 0.5$. We can observe that the highest satisfaction score is obtained at $s = 120$, which is also the cross point of precision and speed.

With the concept of the satisfaction score, we want to predict the best number of samples to use to maximize this indicator. The target is to create a Neural Network that, given in input a set of *Features* F able to characterize a POP instance I (see Section 6.2), is capable of predicting a value for the number of samples s required by the Monte Carlo evaluator to provide the highest possible value of $sat(I,s)$. The Neural Networks will therefore be regressor models capable of combining the values of the input features into a proper (not necessarily linear) function.

Two alternative methods are designed for the prediction, both based on Feed-Forward Neural Networks. They are described in the following subsections.

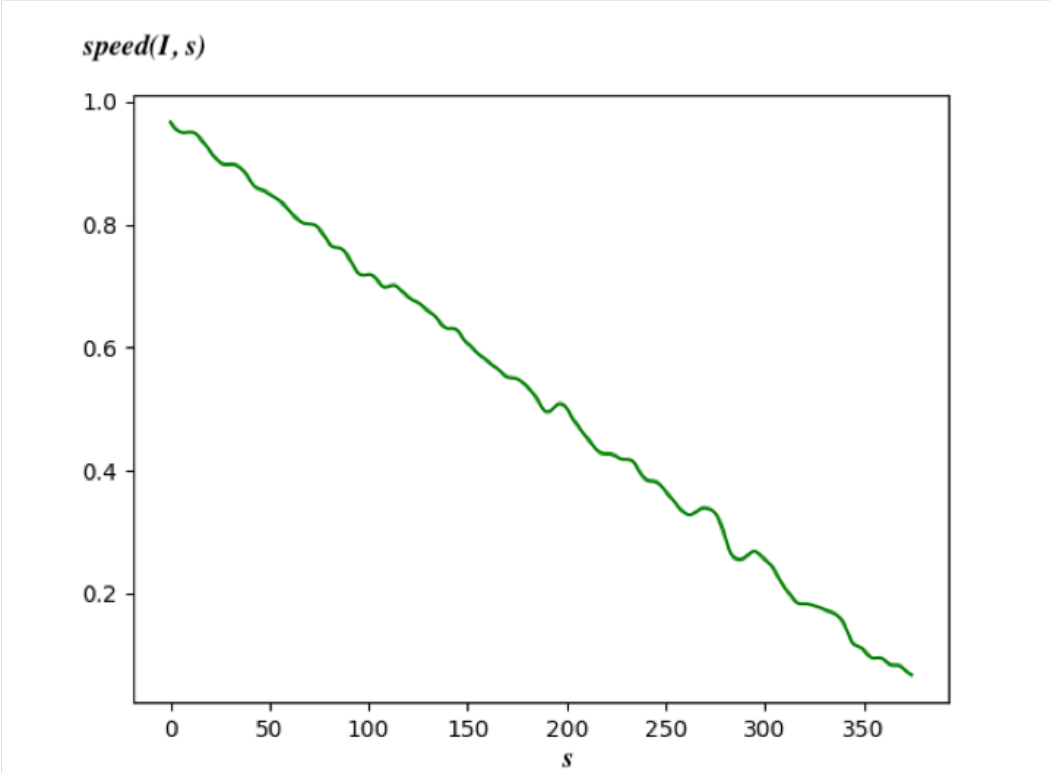

Figure 6.1. Values of $speed(I,s)$ for different values of s for an example instance

6.3.2 Method NN1

The first Neural Network takes the features F from the instance I under investigation as input. There are two inner layers with 32 neurons each and a ReLu activation function [57] and an output layer with one neuron and a Linear activation function (as recommended for regression tasks) [57]. The output of the network provides a prediction for s, the number of samples of the Monte Carlo evaluator. During the training of the method, the Neural Network adapts its weights to predict s with the highest possible precision for each training instance. The adaptation of the weights is based on the distance from the *predicted* s to the *desired* s. The model is shown in Figure 6.4. The training is carried out as a standard back propagation neural network [28].

This architecture has however a drawback. For a given instance I, the predicted value s_p might be relatively far away from the desired value s_d, and the training would modify the weights of the network substantially if s_p and s_d are substantially different from each other. To be more specific, an example is provided in Figure 6.5.

Figure 6.5 shows the chart of the sat function for a given instance I, on the x-axis the different values of s are reported, together with the respective values of $sat(I,S)$ on the y-axis. Our target is to predict the number of samples $s = \arg\max_{z \in S}\{sat(I,x)\}$ for a given instance I, considering equal importance of precision and speed, i.e. $\alpha = 0.5$. Note that the value of α

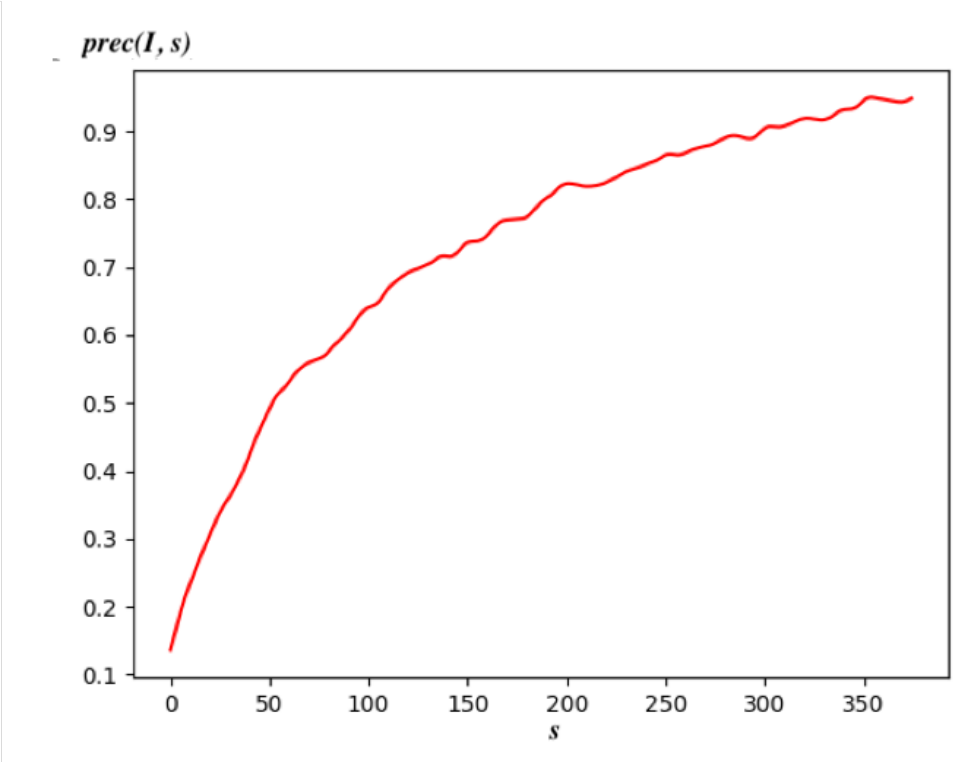

Figure 6.2. Values of $prec(I,s)$ for different values of s for an example instance

does not affect the conclusions we will draw about the prediction of the best value of samples s. Margin (1) is the difference between two s values (s_p and s_d) and margin (2) is the corresponding difference in terms of $sat(I,s)$ (difference between $sat(I,s_p)$ and $sat(I,s_d)$). We can observe that the value of $sat(I,s_p)$ might be very similar to that of $sat(I,s_d)$, notwithstanding the big difference between s_p and s_d. In such a case, s_p could already be classified as a promising solution and the weights considered already as fairly good. Therefore, another method based on the learning of the satisfaction function sat for a given value of s instead of the position of its maximum value only is designed, and will be described in Section 6.3.3.

6.3.3 Method NN2

In the second Neural Network, the input layer receives the features F extracted from the instance I under investigation and also a value of s. There are also two inner layers with 32 neurons each and a ReLu activation function and an output layer with one neuron and a Linear activation function. The output is a prediction for the satisfaction level $sat(I,s)$. The desired value of s will therefore be the one producing the highest output of the associated network (likely, the highest satisfaction value). This method will have to evaluate $|S|$ different values of s through the $|S|$ corresponding neural networks, and choose the most promising one. The model is shown in Figure 6.6. The training is carried out again as a standard back propagation network [28].

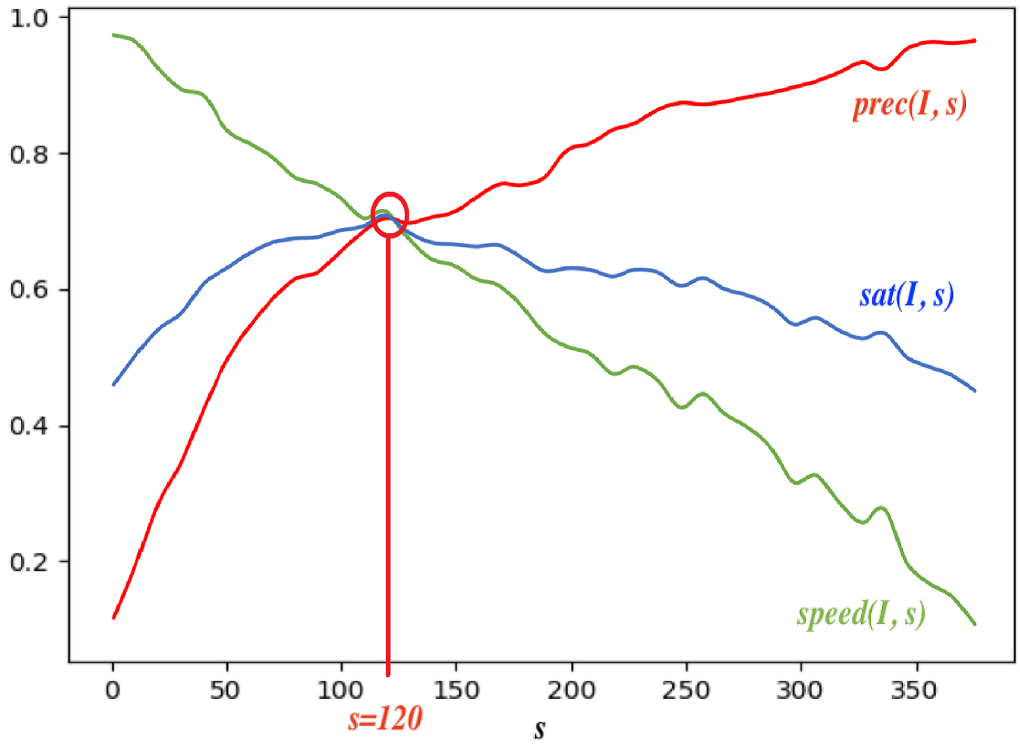

Figure 6.3. Values of $speed(I,s)$, $prec(I,s)$ and $sat(I,s)$ for different values of s for an example instance

The second method is less intuitive comparing to the first, but the methodology is based on the learning of the satisfaction function sat instead of the number of samples s, which allows the possibility of selecting a smaller sample size within a certain range of the satisfaction score. The *Method NN2* has however additional computational complexity when compared to *Method NN1* both during training ($|S|$ independent networks have to be trained instead of just one) and during prediction ($|S|$ predictions have to be done and the best selected). This is however not a problem, since the training phase is carried out once only, while prediction has negligible computation times anyway (see Section 6.4.1).

6.4 Computational Experiments

In this section we study the performance of neural networks proposed in Section 6.3. The experiments have been run on a computer equipped with a quadcore 2.6GHz Intel Core i7 processor and 32GB of RAM. All Neural Networks have been implemented in Keras[1] 2.2.0 and all the data processing has been carried out using Python 3.5.2.

[1] https://github.com/keras-team/keras

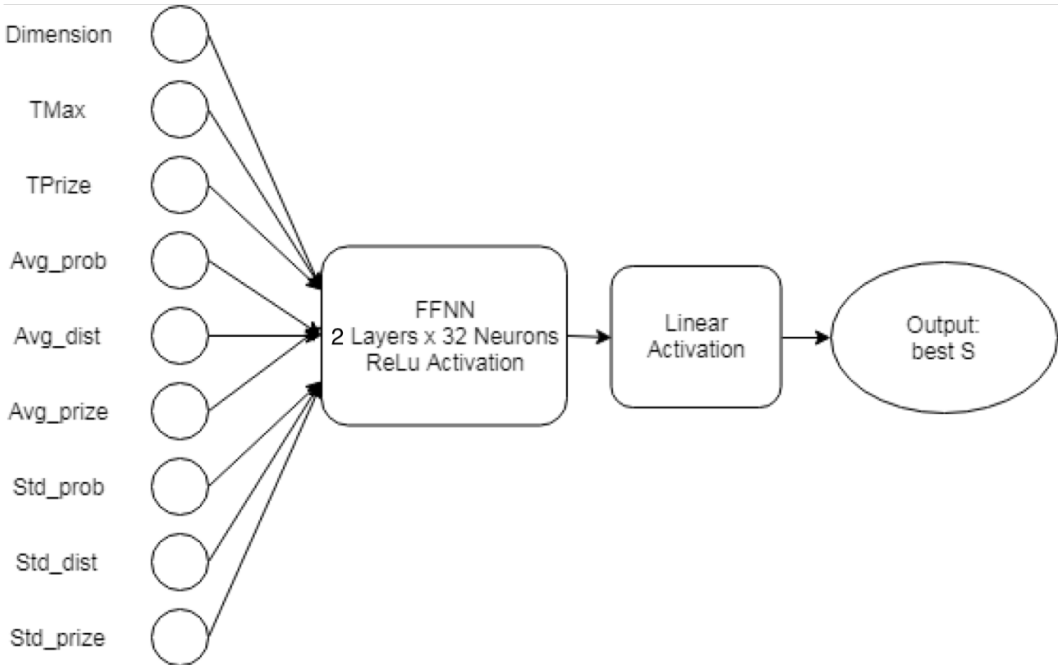

Figure 6.4. Feed Forward Neural Network: Architecture NN1

6.4.1 Training

For each instance I of the 264 instances available, we pre-calculate the value of $sat(I,s)$ (see Section 6.3) for a given solution and each value of $s \in S$ as the average of 10 runs of the Monte Carlo evaluator (see Chapter 3) with s samples. This preprocessing task is at the basis for the training and evaluation of our Machine Learning-based regressors.

We select 85% random instances among the 264 from [1] to be used for training purposes, forming the training set T. The training time is below 500 epochs for the Neural Network of *Method NN1* and between 2000 and 10000 epochs for each of the $|S|$ Neural Network of the *Method NN2*. This translate in a total time for training of about 10-12 seconds for *Method NN1* and 10-20 minutes for *Method NN2*.

Note that the substantial difference in the training steps is motivated by the different complexity of the information learnt by the two methods. We would like also to remark again that the longer training time for *Method NN2* is acceptable, since it has to be carried out only once, and offline.

6.4.2 Testing

The evaluation of the networks is carried out on the 264 instances from [1], both seen and unseen. The error for the prediction of s on the instances of a set Ev is calculated as:

$$\text{Average Error} = \frac{1}{|Ev|} \sum_{I \in Ev} |Pred(I,M) - True(I)|$$

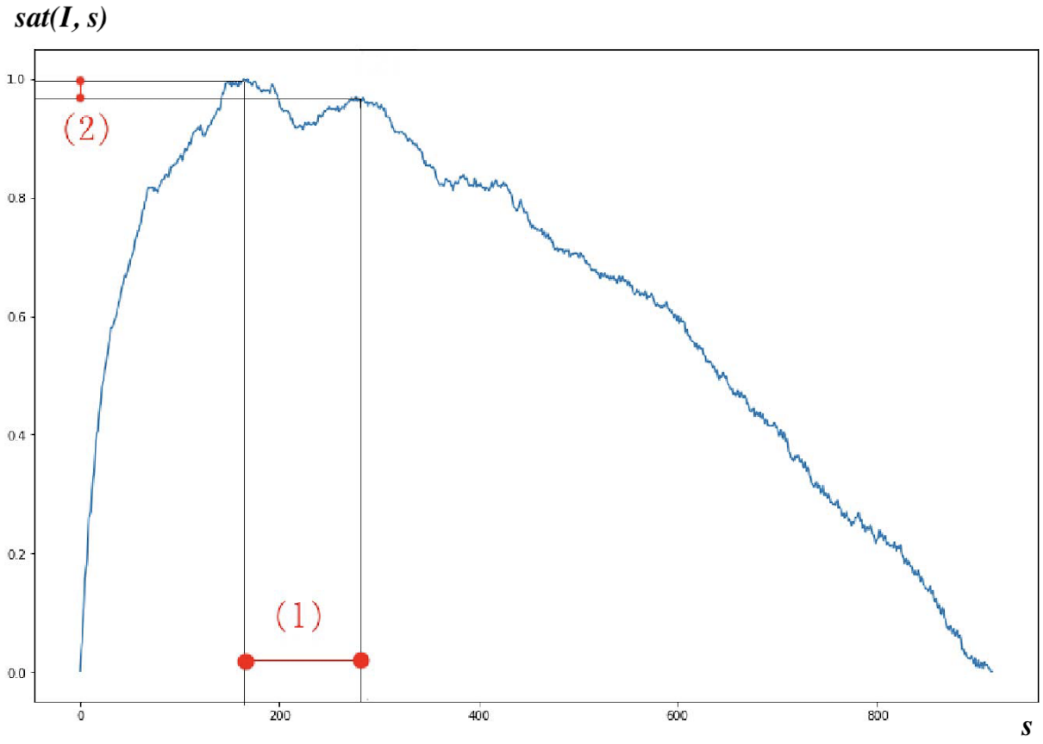

Figure 6.5. Values of $sat(I,s)$ for different values of s for an example instance

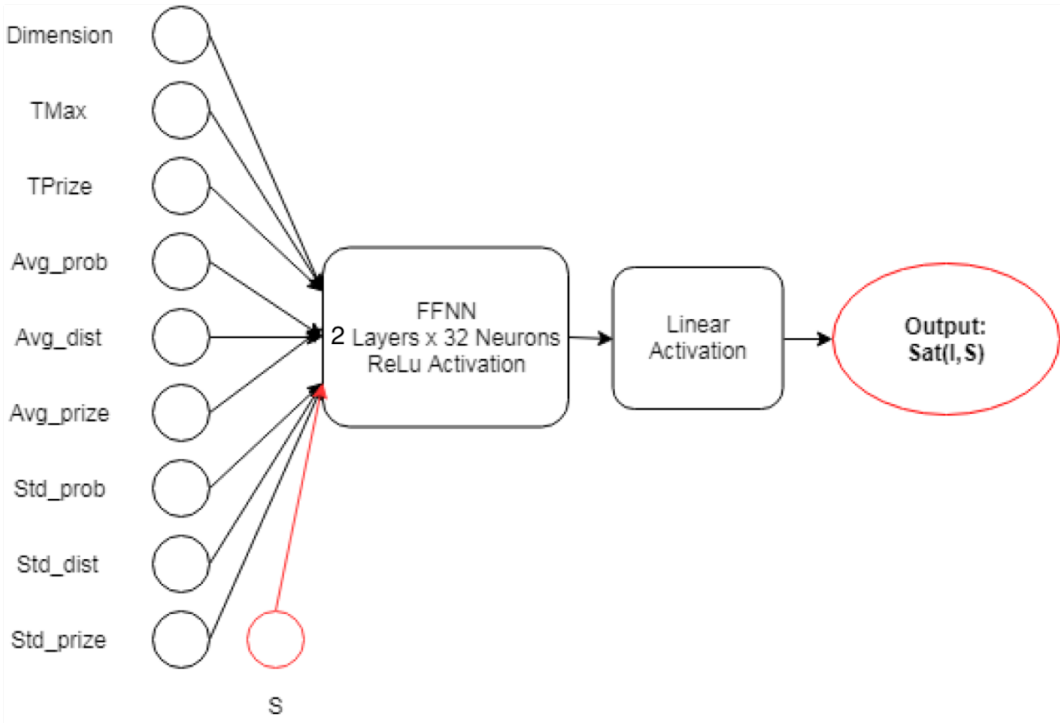

Figure 6.6. Feed Forward Neural Network: Architecture NN2. The differences with respect to architecture NN1 are highlighted in red.

where $Pred(I, M)$ is the value of s predicted by the method M (either *NN1* or *NN2* in our case) for the input associated with instance I and $True(I)$ is the actual best value of s for the instance I, according to the preprocessing and function sat (see Section 6.3). In practice we measure by how many units we fall apart from the optimum value of s on average.

The distribution of the best number of samples predicted are present in Figure 6.7. The results for the two methods proposed in Section 6.3.2 and Section 6.3.3 are shown in Figure 6.8 and summarized in Table 6.2.

First, we analyse the distribution of the the best number of samples predicted for the POP instances. In Figure 6.7 we can observe that the choice of the best number of samples to use is instance independent. With an average value of 166, the best number of samples predicted are mainly distributed in the range [0,110] and [270,330]. The peak frequency is in the range [0,10], however it contains several outliers that are actually instances with large dimension and hard to compute, thus not representative. They also coincide with the units that fall apart from the optimum value of s mentioned in Table 6.2. The second high frequency appears in range [300,310]. It is worth noting that, for several instances based on one same TSP instance with the same location information (see Section 3.1.2), the best number of samples to use is actually different according to different characteristics. A significant result is that the average best number of samples predicted for instances with PRIZE TYPE g2 (see Section 4.3.3) is 111, which is much smaller than PRIZE TYPE g1 with the average value 225.

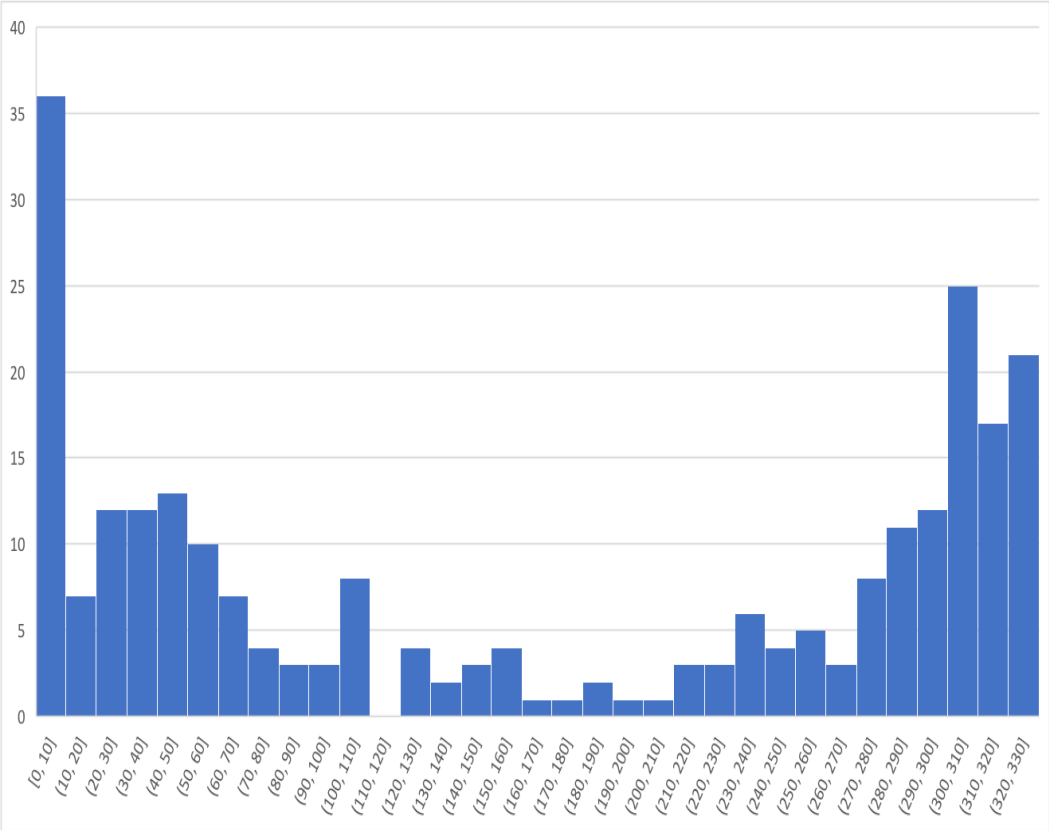

Figure 6.7. Distribution of the best number of samples predicted for 252 POP instances

We then move on to the satisfaction level of the prediction. In Figure 6.8, the 264 instances are ordered by increasing number of best samples required on the x-axis, and the corresponding satisfaction score is shown on the y-axis. We can observe that the bad predictions happen mainly at both ends. The models appear to be confused by outliers with either very small or big number of samples, which is probably due to not enough instances. Considering the distribution of the best number of samples predicted in Figure 6.7, the values at both ends actually account for a large proportion, while in Figure 6.8 only very few instances have bad prediction results. This indicates that the problem occurs only with certain extreme outliers. In Figure 6.8 we can also observe that the majority instances in the middle have very high satisfaction scores. This shows that both methods are able to predict the number of samples s required to maximize the satisfaction of the end user with a good precision. In particular, a difference of s of a few dozens of units is expected to have a marginal impact on the performance of the Monte Carlo evaluator, both in terms of precision and speed.

The average satisfaction score for *Method NN1* is 0.944, and 0.952 for *Method NN2*. In general, the *Method NN2* is able to reduce the average error, providing a slightly more precise prediction. Since the training time is carried out offline, we can conclude that the second neural network is superior to the first, considering the precision and speed.

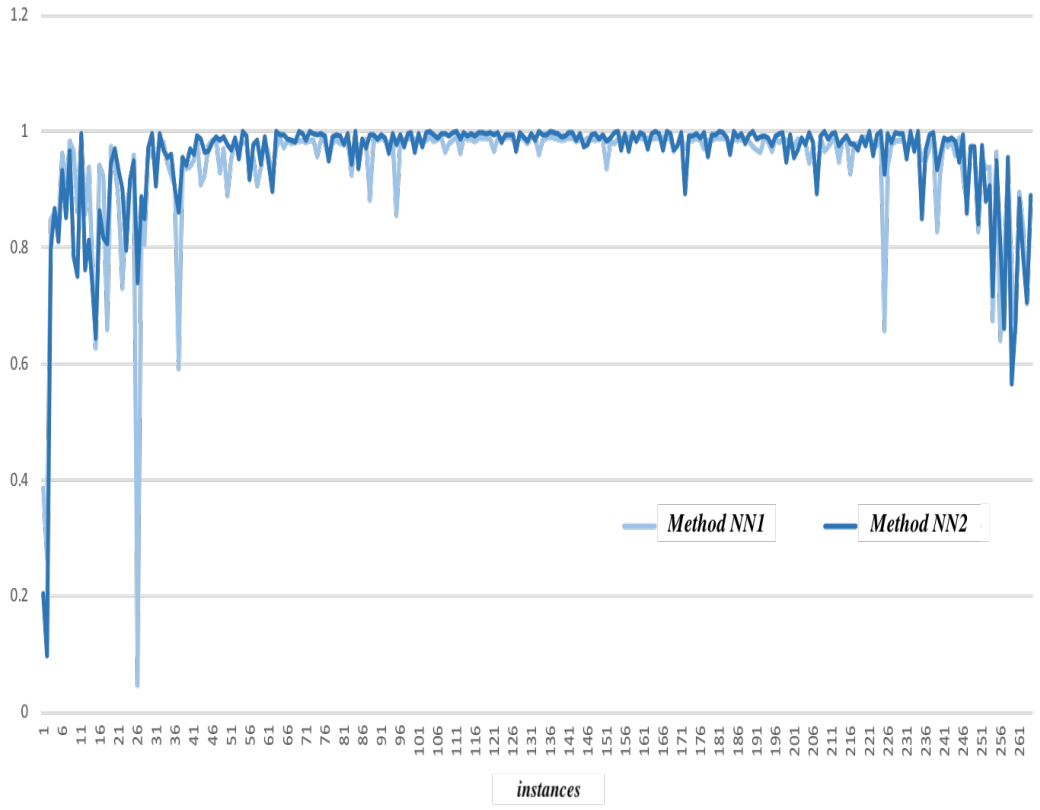

Figure 6.8. Satisfaction level of the prediction

In general we can conclude that, the predicted values of the best number of samples to use is instance independent and the results from the neural networks we propose show a comprehensive consideration of the interactions between multiple features. The values are more precise comparing to those suggested in Chapter 3 since it depends on the different metric used. By considering both precision and speed, the best number of samples to use are predicted better, but this study was carried out later in time.

6.5 Conclusions

In this chapter we have shown that it is possible, by adopting Machine Learning tools, to shift the problem of selecting the number of samples to that of selecting a trade off between speed and precision, which is intuitively a much simpler concept to model.

In particular, two neural network-based approaches have been proposed, showing (experimentally and with logical motivations) that the second one, although less intuitive, is able to provide a more precise prediction for the best value of the number of samples required to maximize the satisfaction function selected by the user. The overall prediction is extremely precise, with a negligible prediction error, especially while taking into account the nature of the

Table 6.2. Prediction results for the two Neural Network-based methods. Average difference between the predicted value of s and the optimal one

Method	Average Error in the prediction of s	Satisfaction score
NN1	36.5 units	0.944
NN2	29.3 units	0.952

predicted parameter.

It is also worth mentioning that the approach we proposed for predicting a parameter of a given problem, can be easily generalized and applied other optimization problems for the prediction of any algorithmic parameter.

Chapter 7

The Probabilistic Tourist Trip Design Problem

In this chapter we present an application of the POP on a probabilistic version of the Tourist Trip Design Problem. The work has appeared in [18].

7.1 Introduction

The Tourist Trip Design Problem (TTDP) ([61]) is a variant of a route-planning problem for tourists interested in visiting multiple points of interest (POI). Suppose that a tourist has her/his own rank of POIs that she/he wants to visit most, that each of the places has different availability, and a certain satisfaction score can be achieved when visited. The objective is to select a subset of POIs to visit within the length of the stay, in such a way that the satisfaction score of the tourist is maximized, while the total time spent between attractions and the total travel time is minimized.

For POIs with deterministic availability, a simple formulation of the TTDP is proven to be identical with the OP ([63]), where a route with maximum score is determined for a subset of locations with fixed depot and destination, limited by the time budget. However, in practice, popular POIs may probabilistically require long waiting time and open-air POIs may not be visited in case of unseasonable weather. A variety of uncertainties could then affect the availability of the POIs, and modelling the availability of POIs with probabilities can therefore lead to more realistic models. In this chapter we demonstrate that the existing model of the POP fits what we define as the Probabilistic Tourist Trip Design Problem (PTTDP). A simple heuristic solver such as the RRLS solver (see Chapter 4) can efficiently provide noble solutions for this application, which is intrinsically characterized by relatively small POP instances.

7.2 Problem Definition

A PTTDP instance contains the information (V, t, π, T_{max}, p) where:

- $V = \{0, 1...n, n+1\}$ is a set of n points of interests (POI) with the starting location being node 0 and the final destination being node $n+1$.

- t_{ij} is the expected travel time from POI i to POI j defined, for all $i, j \in V$.
- T_{max} is the length of the stay.
- π_i is the probability of a POI $i \in V$ to be visitable (according to weather forecast and/or waiting time information) and is modeled by a Bernoulli variable $b_i = \{0, 1\}$. b_i takes value 1 with an independent probability π_i for each POI.
- p_i is the satisfaction score representing how valuable is the attraction.

In the reality, the starting location 0 and the final destination $n+1$ can coincide (e.g. hotel) or not (e.g. hotel + airport). The probabilities π_i might not be independent (e.g. weather in a same city), but the Bernoulli distribution gives a good approximation. The role of the length of the stay T_{max} is equivalent to the deadline D in the POP (see Chapter 2).

In the PTTDP, a point of interest may be valued differently by different people, therefore the satisfaction score p_i is a set of integers in $\{1, 2, ..n\}$ given representing by the personal interests of the person planning the tour, and has to be provided directly by the user. The more you she/he wants to visit a POI, the higher the score is. This personal selection may be based on information from websites or the weather conditions. For example, p_i of outdoor POIs decreases when it rains. Besides, if the tourist had already visited a POI before, the corresponding p_i will decrease as well, going down to 0 possibly.

For a given tour σ, the total satisfaction score is $P(\sigma)$, and the travel time is $T(\sigma)$. We aim at simultaneously maximizing the expected satisfaction score $E(P(\sigma))$ and minimizing the expected travel time $E(T(\sigma))$, since a tourist does not like to spend too much time travelling. Thus, the objective function of the TTDP is the same as the POP equation (2.2) defined in Chapter 2:

$$u(\sigma) = E[P(\sigma)] - CE[T(\sigma)] \tag{7.1}$$

where the coefficient C represents a balance between the distance travelled and the satisfaction score in the PTTDP, that is normally set to a standard value and eventually changed by the user based on her/his attitude to move. When C is small it means the satisfaction score is more important. In this case, if there is one point of interest that the tourist really want to visit, even though it is far away or it has high probability of being crowded, we still tend to search for a tour that includes this POI. When C takes large values, it is the opposite and the trip will tend to cover attractions grouped together.

The PTTDP model allows the tourist to do both day-ahead scheduling and real-time decisions. Note that in a real-time decision making scenario, the depot and destination can also be changed according to the actual situation.

7.3 Experimental Results

In this section we create a test instance based on real data information and show how the application works.

Table 7.1. List of the 15 top attractions in Paris

Number	Name
1	Pyramides (Hotel)
2	Louvre Museum
3	Eiffel tower
4	Cathédrale Notre-Dame de Paris
5	La Seine
6	Orsay Museum
7	Arc de triomphe
8	Basilique du Sacré-Coeur
9	Palais Garnier
10	Montmatre
11	Champs-Élysées
12	Concorde
13	Le centre Georges Pompidou
14	Panthéon
15	Jardin des Tuileries

7.3.1 Touristic application in Paris, France

Since no specific instance is available in the literature for the PTTDP, we generated a test instance by the 15 top attractions in Paris, France. The detailed list of POIs is shown in Table 7.1. We collected online information for the travelling times[1] between POIs, the average staying time at each POI, and the average queues situation of each POI in order to generate the test instance. The average staying time at each POI varies from 15 minutes to 180 minutes, and it is inserted directly into travel times t_{ij}s that vary from 2 minutes to 40 minutes. The length of the stay T_{max} is set to 480 minutes. We set the visitable probability from 0.6 to 0.9 based on the historical statistics of the average queuing time in inverse proportion. The ideal calculation of the probability should also take into account an additional small parameter that measures the probabilities of meeting bad weather for outdoor POIs. We did not add this parameter in this test, but this would not have an impact on the performance of the solver. Figure 7.1 shows the locations of the 15 top attractions[2]. The red pinpoint is the location of the hotel from which the tour starts and ends. In the test instance we pick a hotel very close to one of the attractions, and we solve the instance with the RRLS heuristic solver proposed in Chapter 4 with a number of samples $s = 100$ for the Monte Carlo evaluator (see Chapter 3) and a tolerance value $y = n$. Since the solving for this size of instance is already fast enough, the heuristic speed-up criterion is not activated here to ensure accuracy. With this setting, the test instance can be solved within one second on a normal personal computer, making the approach suitable also for mobile devices and applications, eventually.

Figure 7.2 and Figure 7.3 show the optimal solution for POIs with same and different satisfaction scores. Among the 15 top attractions, the Louvre Museum is a POI that is always crowded and requires long visiting time. When we set equal satisfaction score for all POIs, it

[1] https://maps.google.com

[2] Note that the work reported in this chapter was carried out before Notre-Dame Cathedeal was severely damaged by a fire.

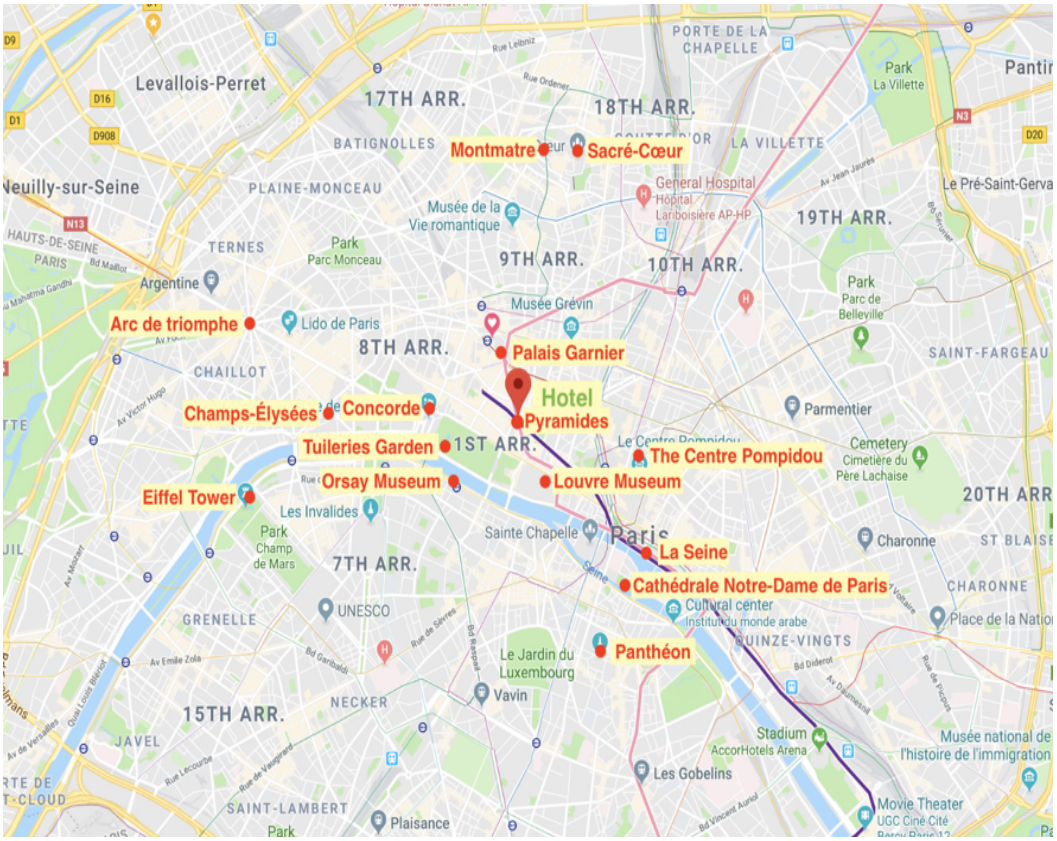

Figure 7.1. Locations of 15 Top Attractions in Paris, France

will never appear on the optimal route, for the reason that we care more about saving time and visiting more places during the length of the stay (Figure 7.2). The solution provided clearly fulfills this demand. However, the Louvre Museum is a very worthwhile place to visit. Therefore, we give Louvre Museum a relatively high score when setting different satisfaction score for each POI, and we obtain the personalized optimal solution that covers less POIs but includes places that the average tourist really wants to see (Figure 7.3).

The optimal tour found for POIs with equal satisfaction score (shown in Figure 7.2) is *Pyramides (Hotel) - La Seine - Cathédrale Notre-Dame de Paris - Palais Garnier - Arc de triomphe - Champs-Élysées - Montmatre - Concorde - Jardin des Tuileries - Eiffel tower - Pyramides (Hotel)*. The optimal tour found for POIs with different satisfaction score (shown in Figure 7.3) is *Pyramides (Hotel) - Palais Garnier - Arc de triomphe - Champs-Élysées - Eiffel tower - Cathédrale Notre-Dame de Paris - La Seine - Louvre Museum - Pyramides (Hotel)*. It is worth noting that some of the attractions are adjacent, therefore the optimal solution might not be unique, being different solutions virtually equivalent. But in general, from this experiment we prove that the PTTDP can be solve efficiently by using a simple solving method proposed for the POP such as the RRLS algorithm.

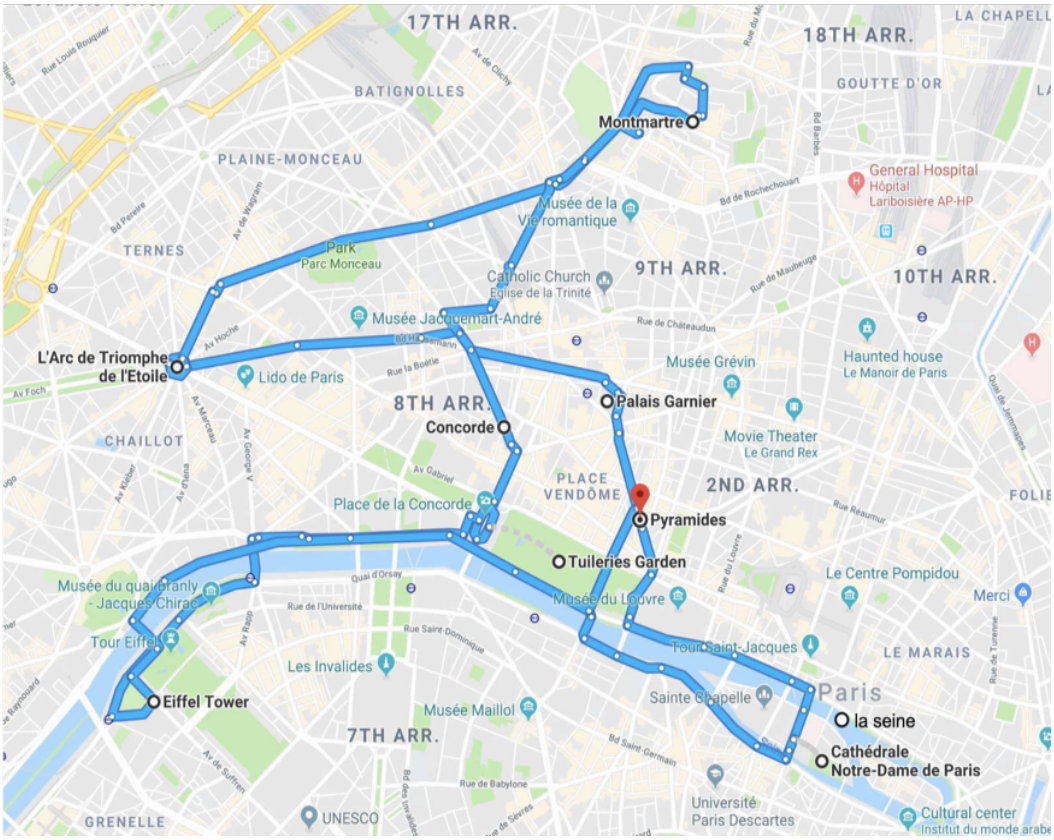

Figure 7.2. Optimal solution for the case with equal satisfaction score for all POIs

7.3.2 A comparison between the POP solvers for small instances

In Section 7.3.1 we show that the PTTDP can be solved easily with a simple POP solver like the RRLS solver. Here in this section, we give a comparison between the performances of all the currently known POP solvers for small instances with dimension up to 30 from [1], that are considered to be relevant to our application.

As reported in [1], a basic branch-and-cut algorithm requires approximately 2000 seconds on average to solve a small POP instance. The fastest variant requires 1500 seconds. Considering this exact approach as benchmark, we compare the performance of all the POP heuristic solvers mentioned in the previous chapters: Smart-Chain (**sChain**)[1], Smart-Path (**sPath**)[1], Smart-Two-ways-path (**s2Path**)[1], the **RRLS** solver (described in Chapter 4), and the **TS** solver (described in Chapter 5). For each solver we report the average gap from the best-known solution, and the average time to incumbent (normalized to the same machine) over a total of 84 small POP instances (described in Section 3.1.2) with dimension $n < 30$ that are relevant for PTTDP applications. The average time to incumbent shows the average time at which the final solution has been found. The computing times are normalized to our machine mentioned in Section 5.6.

Table 7.2 shows that the two solvers we propose (RRLS and TS) are both extremely competitive with state-of-the-art heuristics on small POP instances with dimension $n < 30$. As proven in

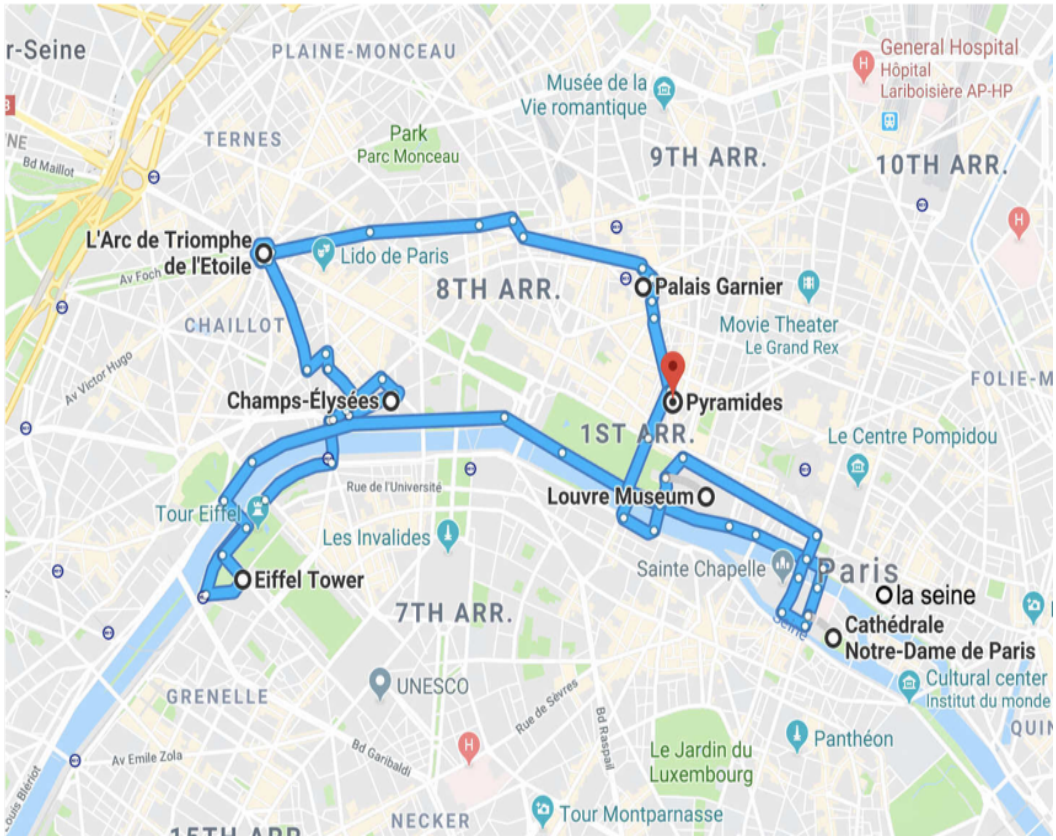

Figure 7.3. Optimal solution for the case with different satisfaction scores for the POIs

Table 7.2. Experimental comparison of heuristic methods on small instances

Methods	**sChain[1]**	**sPath[1]**	**s2Path[1]**	**RRLS**	**TS**
gap(%)	2.52	3.93	2.72	0.75	0.02
time(s)	21	120	168	3	0.09

Section 7.2, the POP model is adaptive to the PTTDP. By taking personal interests, time limits, the POIs' locations and real-time availabilities as input, they will be able to return the selection and the routing of the POIs in a very short time, which allows an application of an online decision support system.

7.4 Models with extra features

There is also some additional flexibility in the PTTDP model: in the example instance of Section 7.3.1 we set the depot and destination at the hotel. As we mentioned in Section 7.2, the starting and ending points do not have to coincide. For example, the tour can start at the hotel and finish at a bus or train station or at an airport. With up-to-date POI information and changeable personal preference, a feasible selection of POIs and a tour can always be suggested in a reasonable time.

On the other hand, a drawback of the POP model we propose is that it can only solve the problem for one day, or a fraction of it (e.g. the afternoon of a business trip when all the work is carried out in the morning). In case the tour covers multiple days, since hotel is a special location with unpredictable staying time, it could be natural to extend the approach we propose by solving several sub-problems for each day and each POI appears only once in one day. Unfortunately, the pre-allocation of POIs to days of such an approach might compromise the optimality of the solution. In order to overcome this issue, a possible extension of this model would be to consider a Probabilistic Team Orienteering Problem, where the POIs are split to the different days of the stay automatically by the solver, then organise predefined number of non-overlapping tours, one for each day of the staying.

Besides, in our current model the calculation of the availability of a POI is based on waiting time and/or weather forecast only. However, investigation shows that most tourists prefer to move within a limited area with very attractive POIs for safety reasons and because they feel more in control [36]. When dealing with similar situation with our model, it is high probable that our solver suggests a less attractive POI with less people, but this does not meet tourists' needs. Therefore, putting more specified preference constraints apart from satisfaction scores into the model could lead to another improvement. Furthermore, the probabilities are modeled by a sequence of independent Bernoulli trials in this work. It is also possible that we treat mutual dependent π_is by modifying the MC evaluator in order to model the reality even better.

7.5 Conclusions

In this chapter we demonstrate that the existing model of the POP fits a stochastic variation of the Tourist Trip Design Problem. We show that the problem can be efficiently and effectively solved by the heuristic solvers we proposed. We finally discuss several possible improvements on the model. Extensions of the model to make it even closer to tourist needs are finally identified.

Chapter 8

Conclusions

In this thesis, we have studied and developed metaheuristic methods for the Probabilistic Orienteering Problem. A detailed computational study of the new approaches is presented, with the aim of studying the performance of the evaluators in terms of precision and speed, while positioning the new metaheuristic methods within the existing literature.

At the beginning of the study, we have investigated the use of Monte Carlo sampling approach for the evaluation of the objective function of the POP. As a state-of-the-art approach for several stochastic vehicle routing problems, we extended its use to the POP. We also explored the use of the same Monte Carlo sampling procedure to decide how many customers of a given complete tour should be visited in order to maximize the profit, and this is an innovation. Such a characteristic has been proved to be very useful once the evaluator is embedded into metaheuristic algorithms. This innovation is not limited to POP, it can be applied to similar problems where a subset of customers need to be chosen. More broadly, when certain decisions need to be made originally in the searching procedure, we have opened a way to shift in part of the decision-making in the evaluation phase, which will improve the overall efficiency. Another outcome of the research was a Monte Carlo evaluator with a heuristic speed-up criterion. By tuning the number of samples, we found the considerable choice to be used which gives a good balance between the quality of the approximation and the computation speed.

We then solved the POP by using a Random Restart Local Search method. The aim of our approaches is to find high quality solutions in a short time. With this goal, the same speed-up criterion in the Monte Carlo evaluator are heavily used. We performed computational studies on precision and speed with different values of a tolerance parameter defining the speed-up criterion. From the experimental results we gave suggestions for the choice of parameters based on different characteristics of the problem instances. This analysis allowed us to understand the problem more thoroughly, and made it easier to extend the use of the algorithm to new instances with similar characteristics. After tuning, the average error with respect to best known solutions dropped below 6% in 250s with RRLS method. For dimension $n < 30$, the average error dropped below 1% in 10 seconds. This showed that a combination of the Monte Carlo evaluator and a simple RRLS method can solve the POP, and it performed particularly well on small instances. However, an insufficient performance on large instances showed that the 2-opt local search we use within the RRLS might not be sufficient for large instances, stronger local search methods and further improvements were needed for more accurate results. We then proposed to use the *k-Nearest Random Insertion* method to generate initial solutions. In the new method, while

generating an initial solution, instead of picking the nearest feasible neighbour at each step or a random one, we carried out a Random Insertion where the candidate nodes are the k feasible ones closest to the last node added in the previous iteration. A detailed computational study has also revealed the influence of the parameter k in the performance of the overall method. It was shown that by adopting this re-initialising method, the performance of the RRLS algorithm for POP improves. We expect that it is also applicable to similar use of the random restart mechanism or other heuristic methods.

We then applied this initialising method on the Tabu Search algorithm, another method that we proposed for solving the POP. In the TS algorithm, local search methods are again used to provide heuristic solutions. As a consequence of the flexibility design of our algorithm, the operators in the local search procedure can be easily replaced. To verify our assumption that stronger local search methods may improve the results, we made a comparison between the 2-opt and 3-opt operators to evaluate which is more suitable for the POP. Experimental results showed that the 2-opt heuristic works better on 84% of the instances (for the POP benchmarks studied in this work), therefore it is more suitable for the solving method we propose for the POP. The majority of the instances for which the 3-opt heuristic works better are of medium to long deadline types. This indicates that the 2-opt heuristic and the 3-opt heuristic have different adaptability on the POP instances. After tuning, the average error with respect to best known solutions dropped below 1.88% in 6s with TS algorithm. For small instances with dimension $n < 30$, the TS algorithm took only 0.09 seconds on average to solve the problem with an average error of 0.02%. It also retrieved more best known solutions with much shorter times when compared to the other state-of-the-art heuristic methods. The TS algorithm remains competitive for medium size instances on accuracy and speed, but loses advantage on accuracy for large instances. In general, we can conclude that the TS algorithm is extremely fast, and with small errors comparable with those of considerably more complex heuristics from the literature. In practical terms, it could be a precious approach for application where quick decisions are required, such as an online decision support system. Studies on stronger local search methods are expected for further improvements. Meanwhile, the outcome of our research is designed in a flexible way to have a general approach that can be easily adapted to other stochastic combinatorial optimization problem that have similar structure. A very close target can be a probabilistic variation of the Team Orienteering Problem (PTOP), we expect the TS algorithm to achieve promising results on this problem too.

In this thesis we also showed that it is possible to use Machine Learning tools to help to solve the POP. An example of parameter tuning with neural network-based approaches was presented. With experimental results and logical motivations, we showed that both models we propose are able to predict the best number of samples to use in the Monte Carlo evaluator for an instance by maximizing the satisfaction function selected by the user. The overall prediction is extremely precise, with a negligible prediction error, especially while taking into account the nature of the predicted parameter. The approach can be easily generalized and applied other optimization problems for the prediction of any algorithmic parameter. With more solving methods in the literature for the POP or related problems, the potential of learning-based approaches are also expected to help with selecting the appropriate solving methods to be used. The integration of machine learning and classical optimization techniques is nowadays of a big interest in the field.

From the application point of view, we demonstrated that the existing model of the POP fits a stochastic variation of the Tourist Trip Design Problem we propose. This application transfers

the queuing time at a tourist attraction into a probability representing the willingness of visiting the place. This allows research in different fields to work together, for example using crowd psychology to build different probabilistic models to achieve more realistic simulations. The problem itself can be efficiently and effectively solved by the POP heuristic solvers we proposed. Considering that the Monte Carlo sampling method is used for evaluation, different probabilistic models can be easily employed. A study on PTOP can be generalized to other stochastic Tourist Trip Design - related applications, such as planning a different tour for each day during the length of the variability of the stay, or determine same or separate tours for a tour group, while taking into account availability of the points of interests on the wish list of attractions to visit. A much further probabilistic extension of the problem can also be based on other variations of the OP, such as the Orienteering Problem with Time Windows or Team Orienteering Problem with Time Windows. Future work on building applications that are close to tourist needs and take into account more realistic factors is not far away.

To conclude, we strongly believe that the work presented in this thesis is able to contribute to the state-of-the-art literature for solving methods for the POP and indicated new directions for future research. We are grateful to leave footprints in the sand of time, and guide the way forward.

Appendix A

2-opt and 3-opt operators inside the Tabu Search algorithm: Extended Results

These are the complete results of the comparison between 2-opt and 3-opt operators inside the Tabu Search algorithm from Section 5.6.3.

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
att48FSTCII_q1_g1_p1	0.00	0.63	0.00	18.94
att48FSTCII_q1_g1_p2	0.00	0.51	0.00	41.53
att48FSTCII_q1_g2_p1	0.00	0.82	0.00	1.26
att48FSTCII_q1_g2_p2	0.00	0.25	0.00	1.28
att48FSTCII_q2_g1_p1	0.00	3.33	0.00	16.25
att48FSTCII_q2_g1_p2	0.26	1.46	0.63	54.07
att48FSTCII_q2_g2_p1	0.24	2.63	2.92	14.41
att48FSTCII_q2_g2_p2	0.39	8.17	1.76	23.39
att48FSTCII_q3_g1_p1	0.06	27.9	2.79	74.20
att48FSTCII_q3_g1_p2	1.44	22.58	2.00	65.51
att48FSTCII_q3_g2_p1	4.49	27.37	9.57	44.41
att48FSTCII_q3_g2_p2	4.64	12.85	1.55	5.62
bayg29FSTCII_q1_g1_p1	0.00	0.04	0.35	0.06
bayg29FSTCII_q1_g1_p2	0.00	0.00	0.09	0.14
bayg29FSTCII_q1_g2_p1	0.00	0.00	0.00	0.02
bayg29FSTCII_q1_g2_p2	0.00	0.00	0.00	0.06
bayg29FSTCII_q2_g1_p1	0.00	0.26	0.24	2.31
bayg29FSTCII_q2_g1_p2	0.00	0.96	0.00	6.99
bayg29FSTCII_q2_g2_p1	0.00	4.56	0.00	13.65
bayg29FSTCII_q2_g2_p2	0.00	0.26	0.00	4.39

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
bayg29FSTCII_q3_g1_p1	0.06	2.4	0.15	6.96
bayg29FSTCII_q3_g1_p2	0.41	4.59	0.00	3.32
bayg29FSTCII_q3_g2_p1	0.47	7.53	1.44	0.21
bayg29FSTCII_q3_g2_p2	0.99	11.14	2.03	7.91
bays29FSTCII_q1_g1_p1	0.00	0.04	0.03	1.47
bays29FSTCII_q1_g1_p2	0.00	0.00	0.00	1.49
bays29FSTCII_q1_g2_p1	0.00	0.16	0.00	4.07
bays29FSTCII_q1_g2_p2	0.00	0.90	0.50	0.04
bays29FSTCII_q2_g1_p1	0.00	0.86	0.00	20.36
bays29FSTCII_q2_g1_p2	0.00	0.50	0.07	2.92
bays29FSTCII_q2_g2_p1	0.94	4.21	0.94	7.97
bays29FSTCII_q2_g2_p2	0.00	1.05	0.00	0.40
bays29FSTCII_q3_g1_p1	0.02	3.37	0.13	3.60
bays29FSTCII_q3_g1_p2	0.00	0.17	0.00	6.77
bays29FSTCII_q3_g2_p1	0.47	0.02	0.47	26.24
bays29FSTCII_q3_g2_p2	0.52	3.25	0.97	6.18
berlin52FSTCII_q1_g1_p1	0.00	0.36	0.00	9.32
berlin52FSTCII_q1_g1_p2	0.00	0.43	0.00	4.94
berlin52FSTCII_q1_g2_p1	0.48	4.77	1.11	4.71
berlin52FSTCII_q1_g2_p2	0.00	0.26	0.00	20.6
berlin52FSTCII_q2_g1_p1	0.98	8.86	1.65	79.73
berlin52FSTCII_q2_g1_p2	0.57	15.30	0.19	79.05
berlin52FSTCII_q2_g2_p1	0.82	30.1	1.73	21.12
berlin52FSTCII_q2_g2_p2	3.11	14.6	4.73	23.74
berlin52FSTCII_q3_g1_p1	2.24	10.84	3.62	73.81
berlin52FSTCII_q3_g1_p2	1.70	15.89	3.41	66.57
berlin52FSTCII_q3_g2_p1	2.63	8.74	2.68	43.17
berlin52FSTCII_q3_g2_p2	2.90	9.97	2.75	32.72
brazil58FSTCII_q1_g1_p1	0.00	2.38	1.28	22.93
brazil58FSTCII_q1_g1_p2	0.00	0.33	0.00	16.85
brazil58FSTCII_q1_g2_p1	0.00	1.70	2.56	49.61
brazil58FSTCII_q1_g2_p2	2.73	0.38	1.36	10.51
brazil58FSTCII_q2_g1_p1	3.81	20.03	6.75	71.79
brazil58FSTCII_q2_g1_p2	1.07	9.52	2.31	94.21
brazil58FSTCII_q2_g2_p1	2.45	14.09	3.53	51.10
brazil58FSTCII_q2_g2_p2	2.31	12.56	3.53	40.11
brazil58FSTCII_q3_g1_p1	0.02	10.04	12.11	180.86
brazil58FSTCII_q3_g1_p2	0.24	35.71	0.62	182.10
brazil58FSTCII_q3_g2_p1	0.003	17.14	0.00	101.70
brazil58FSTCII_q3_g2_p2	0.01	15.34	0.00	32.23
burma14FSTCII_q1_g1_p1	0.00	0.00	0.00	0.00
burma14FSTCII_q1_g1_p2	0.00	0.00	0.00	0.00

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
burma14FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
burma14FSTCII_q1_g2_p2	0.00	0.00	0.00	0.00
burma14FSTCII_q2_g1_p1	0.00	0.00	0.00	0.02
burma14FSTCII_q2_g1_p2	0.00	0.01	0.00	0.02
burma14FSTCII_q2_g2_p1	0.00	0.02	0.00	0.01
burma14FSTCII_q2_g2_p2	0.00	0.00	0.00	0.01
burma14FSTCII_q3_g1_p1	0.00	0.00	0.00	0.03
burma14FSTCII_q3_g1_p2	0.00	0.00	0.00	0.07
burma14FSTCII_q3_g2_p1	0.00	0.00	0.00	0.00
burma14FSTCII_q3_g2_p2	0.00	0.00	0.00	0.00
dantzig42FSTCII_q1_g1_p1	0.00	0.01	0.01	0.13
dantzig42FSTCII_q1_g1_p2	0.00	0.04	0.00	8.43
dantzig42FSTCII_q1_g2_p1	0.00	0.04	0.00	0.11
dantzig42FSTCII_q1_g2_p2	0.00	0.01	0.00	0.14
dantzig42FSTCII_q2_g1_p1	0.00	1.14	0.00	10.16
dantzig42FSTCII_q2_g1_p2	0.00	0.51	0.00	18.33
dantzig42FSTCII_q2_g2_p1	1.49	2.62	1.49	7.38
dantzig42FSTCII_q2_g2_p2	1.60	6.72	3.42	12.72
dantzig42FSTCII_q3_g1_p1	0.002	13.65	0.00	3.01
dantzig42FSTCII_q3_g1_p2	1.76	2.52	0.59	4.72
dantzig42FSTCII_q3_g2_p1	5.18	10.2	3.51	34.45
dantzig42FSTCII_q3_g2_p2	4.49	2.48	4.56	10.88
eil51FSTCII_q1_g1_p1	0.00	0.81	7.23	5.21
eil51FSTCII_q1_g1_p2	0.00	19.27	2.80	87.57
eil51FSTCII_q1_g2_p1	0.00	6.98	0.00	50.60
eil51FSTCII_q1_g2_p2	0.00	0.11	2.58	4.44
eil51FSTCII_q2_g1_p1	3.53	0.79	3.56	36.29
eil51FSTCII_q2_g1_p2	1.52	7.41	0.00	5.13
eil51FSTCII_q2_g2_p1	0.00	16.43	2.99	5.22
eil51FSTCII_q2_g2_p2	2.44	3.16	6.17	21.84
eil51FSTCII_q3_g1_p1	2.48	23.84	2.48	47.20
eil51FSTCII_q3_g1_p2	0.99	8.54	2.02	88.36
eil51FSTCII_q3_g2_p1	1.86	2.86	1.99	27.56
eil51FSTCII_q3_g2_p2	0.80	5.82	2.79	25.56
eil76FSTCII_q1_g1_p1	0.00	12.69	4.64	25.15
eil76FSTCII_q1_g1_p2	2.49	13.78	6.27	12.26
eil76FSTCII_q1_g2_p1	2.89	5.88	16.47	16.86
eil76FSTCII_q1_g2_p2	2.84	8.05	9.82	37.91
eil76FSTCII_q2_g1_p1	6.69	21.14	4.42	27.44
eil76FSTCII_q2_g1_p2	5.99	1.77	10.60	28.65
eil76FSTCII_q2_g2_p1	8.12	7.37	17.89	21.15
eil76FSTCII_q2_g2_p2	8.10	13.97	16.66	26.69

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
eil76FSTCII_q3_g1_p1	4.81	8.91	1.58	75.16
eil76FSTCII_q3_g1_p2	4.81	17.10	5.85	94.69
eil76FSTCII_q3_g2_p1	6.94	16.20	4.40	112.88
eil76FSTCII_q3_g2_p2	5.36	7.82	3.44	27.96
fri26FSTCII_q1_g1_p1	0.00	0.00	0.00	0.00
fri26FSTCII_q1_g1_p2	0.00	0.00	0.00	0.00
fri26FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
fri26FSTCII_q1_g2_p2	0.00	0.00	0.00	0.00
fri26FSTCII_q2_g1_p1	0.00	0.00	0.00	0.34
fri26FSTCII_q2_g1_p2	0.00	0.47	0.44	1.76
fri26FSTCII_q2_g2_p1	0.00	0.04	0.00	24.56
fri26FSTCII_q2_g2_p2	0.00	0.15	0.00	2.29
fri26FSTCII_q3_g1_p1	0.00	0.05	0.00	0.89
fri26FSTCII_q3_g1_p2	0.00	0.03	0.00	3.39
fri26FSTCII_q3_g2_p1	0.00	0.02	0.00	2.09
fri26FSTCII_q3_g2_p2	0.00	0.09	0.00	0.17
gr17FSTCII_q1_g1_p1	0.00	0.00	0.00	0.00
gr17FSTCII_q1_g1_p2	0.00	0.00	0.00	0.00
gr17FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
gr17FSTCII_q1_g2_p2	0.00	0.00	0.00	0.09
gr17FSTCII_q2_g1_p1	0.00	0.00	0.00	0.03
gr17FSTCII_q2_g1_p2	0.00	0.00	0.00	0.07
gr17FSTCII_q2_g2_p1	0.00	0.00	0.00	0.03
gr17FSTCII_q2_g2_p2	0.00	0.00	0.00	0.01
gr17FSTCII_q3_g1_p1	0.00	0.00	0.00	0.17
gr17FSTCII_q3_g1_p2	0.00	0.02	0.00	0.48
gr17FSTCII_q3_g2_p1	0.00	0.01	0.00	0.08
gr17FSTCII_q3_g2_p2	0.00	0.00	0.00	0.08
gr21FSTCII_q1_g1_p1	0.00	0.00	0.00	0.00
gr21FSTCII_q1_g1_p2	0.00	0.00	0.00	0.00
gr21FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
gr21FSTCII_q1_g2_p2	0.00	0.00	0.00	0.00
gr21FSTCII_q2_g1_p1	0.00	0.00	0.00	0.10
gr21FSTCII_q2_g1_p2	0.00	0.02	0.00	4.41
gr21FSTCII_q2_g2_p1	0.00	0.00	0.00	0.70
gr21FSTCII_q2_g2_p2	0.00	0.00	0.00	1.25
gr21FSTCII_q3_g1_p1	0.00	0.08	0.00	0.26
gr21FSTCII_q3_g1_p2	0.00	0.07	0.00	20.15
gr21FSTCII_q3_g2_p1	0.00	0.05	0.92	3.87
gr21FSTCII_q3_g2_p2	0.00	0.02	0.60	7.99
gr24FSTCII_q1_g1_p1	0.00	0.00	0.00	0.00
gr24FSTCII_q1_g1_p2	0.00	0.00	0.00	0.32

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
gr24FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
gr24FSTCII_q1_g2_p2	0.00	0.00	0.00	0.00
gr24FSTCII_q2_g1_p1	0.00	0.22	0.28	6.18
gr24FSTCII_q2_g1_p2	0.00	0.11	0.62	= 6.38
gr24FSTCII_q2_g2_p1	0.00	0.04	0.00	6.19
gr24FSTCII_q2_g2_p2	0.00	0.08	2.25	0.18
gr24FSTCII_q3_g1_p1	0.00	0.06	0.00	3.17
gr24FSTCII_q3_g1_p2	0.00	3.14	2.89	7.55
gr24FSTCII_q3_g2_p1	0.75	0.5	0.32	0.4
gr24FSTCII_q3_g2_p2	1.28	0.36	0.00	12.92
gr48FSTCII_q1_g1_p1	0.00	0.4	0.15	20.13
gr48FSTCII_q1_g1_p2	0.00	0.05	4.44	5.35
gr48FSTCII_q1_g2_p1	3.23	7.03	4.68	45.59
gr48FSTCII_q1_g2_p2	3.67	3.78	1.80	14.71
gr48FSTCII_q2_g1_p1	3.32	10.03	4.09	29.79
gr48FSTCII_q2_g1_p2	3.55	6.91	3.64	29.75
gr48FSTCII_q2_g2_p1	1.49	33.5	3.03	29.72
gr48FSTCII_q2_g2_p2	3.47	18.61	3.78	39.47
gr48FSTCII_q3_g1_p1	2.26	14.06	2.35	73.71
gr48FSTCII_q3_g1_p2	3.85	27.99	2.95	61.23
gr48FSTCII_q3_g2_p1	2.30	8.38	2.53	47.89
gr48FSTCII_q3_g2_p2	3.18	9.35	2.07	61.46
gr96FSTCII_q1_g1_p1	0.00	8.71	8.53	112.31
gr96FSTCII_q1_g1_p2	2.28	20.87	4.48	81.27
gr96FSTCII_q1_g2_p1	1.10	14.65	9.67	50.69
gr96FSTCII_q1_g2_p2	4.00	7.38	8.13	29.9
gr96FSTCII_q2_g1_p1	19.02	45.63	91.67	81.03
gr96FSTCII_q2_g1_p2	14.92	58.11	93.83	127.85
gr96FSTCII_q2_g2_p1	8.50	26.52	8.72	100.80
gr96FSTCII_q2_g2_p2	12.41	12.47	18.66	80.70
gr96FSTCII_q3_g1_p1	31.94	49.06	78.90	192.64
gr96FSTCII_q3_g1_p2	22.01	33.18	88.01	183.45
gr96FSTCII_q3_g2_p1	4.16	10.90	2.90	147.97
gr96FSTCII_q3_g2_p2	5.15	8.04	4.66	174.93
hk48FSTCII_q1_g1_p1	0.00	4.08	10.36	3.97
hk48FSTCII_q1_g1_p2	4.86	10.98	12.36	40.12
hk48FSTCII_q1_g2_p1	0.00	0.93	3.60	11.47
hk48FSTCII_q1_g2_p2	0.00	0.03	2.84	12.46
hk48FSTCII_q2_g1_p1	3.37	12.09	7.48	42.08
hk48FSTCII_q2_g1_p2	4.14	6.45	5.44	107.45
hk48FSTCII_q2_g2_p1	0.00	6.07	3.00	26.57
hk48FSTCII_q2_g2_p2	4.03	6.96	4.70	20.68

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
hk48FSTCII_q3_g1_p1	2.91	12.47	0.86	50.06
hk48FSTCII_q3_g1_p2	1.35	11.29	1.27	27.27
hk48FSTCII_q3_g2_p1	1.26	18.41	1.48	30.09
hk48FSTCII_q3_g2_p2	0.16	15.41	1.98	3.97
pr76FSTCII_q1_g1_p1	0.00	0.05	0.00	0.00
pr76FSTCII_q1_g1_p2	0.00	0.04	0.00	0.00
pr76FSTCII_q1_g2_p1	0.70	8.07	4.50	22.45
pr76FSTCII_q1_g2_p2	3.75	13.05	4.11	40.12
pr76FSTCII_q2_g1_p1	0.00	0.08	0.00	0.00
pr76FSTCII_q2_g1_p2	0.00	0.07	0.00	0.00
pr76FSTCII_q2_g2_p1	8.10	25.72	7.87	56.70
pr76FSTCII_q2_g2_p2	8.26	13.17	11.95	52.98
pr76FSTCII_q3_g1_p1	0.00	0.08	0.00	0.00
pr76FSTCII_q3_g1_p2	0.00	0.08	0.00	0.00
pr76FSTCII_q3_g2_p1	3.42	4.97	8.07	103.86
pr76FSTCII_q3_g2_p2	3.61	5.39	5.70	114.41
rat99FSTCII_q1_g1_p1	0.01	5.51	4.25	59.63
rat99FSTCII_q1_g1_p2	5.22	21.32	9.32	19.19
rat99FSTCII_q1_g2_p1	4.84	4.56	5.91	23.05
rat99FSTCII_q1_g2_p2	6.67	4.5	0.00	21.57
rat99FSTCII_q2_g1_p1	9.84	4.53	9.87	115.02
rat99FSTCII_q2_g1_p2	10.90	6.09	15.97	100.44
rat99FSTCII_q2_g2_p1	11.51	16.54	11.93	78.95
rat99FSTCII_q2_g2_p2	18.27	5.03	16.68	79.44
rat99FSTCII_q3_g1_p1	11.69	24.19	10.43	140.12
rat99FSTCII_q3_g1_p2	12.03	12.27	9.13	175.53
rat99FSTCII_q3_g2_p1	11.47	5.11	8.43	51.75
rat99FSTCII_q3_g2_p2	12.41	11.15	9.44	54.31
st70FSTCII_q1_g1_p1	0.00	0.22	0.00	73.01
st70FSTCII_q1_g1_p2	0.62	5.00	7.09	15.44
st70FSTCII_q1_g2_p1	3.19	1.14	11.03	11.02
st70FSTCII_q1_g2_p2	0.00	11.13	12.47	10.32
st70FSTCII_q2_g1_p1	2.49	16.30	4.92	21.29
st70FSTCII_q2_g1_p2	5.62	10.34	1.19	70.98
st70FSTCII_q2_g2_p1	4.19	17.48	1.37	17.27
st70FSTCII_q2_g2_p2	5.66	13.86	4.24	24.01
st70FSTCII_q3_g1_p1	3.57	6.63	1.76	74.92
st70FSTCII_q3_g1_p2	3.73	3.94	2.63	43.08
st70FSTCII_q3_g2_p1	5.30	23.04	4.00	14.59
st70FSTCII_q3_g2_p2	5.23	33.05	3.78	51.02
swiss42FSTCII_q1_g1_p1	0.00	0.73	0.00	1.40
swiss42FSTCII_q1_g1_p2	0.00	0.07	0.00	1.25

Continued on next page

Table A.1. TS - 2-opt vs 3-opt for 264 POP instances (cont'd)

operators / instances	**2-opt**		**3-opt**	
	error(%)	time(s)	error(%)	time(s)
swiss42FSTCII_q1_g2_p1	0.00	0.56	1.97	1.71
swiss42FSTCII_q1_g2_p2	0.00	1.45	1.35	1.7
swiss42FSTCII_q2_g1_p1	0.00	2.79	0.00	2.08
swiss42FSTCII_q2_g1_p2	0.00	3.68	1.19	1.82
swiss42FSTCII_q2_g2_p1	0.07	2.56	1.46	33.08
swiss42FSTCII_q2_g2_p2	0.19	19.59	0.17	103.92
swiss42FSTCII_q3_g1_p1	0.00	20.61	0.25	19.19
swiss42FSTCII_q3_g1_p2	1.99	6.15	0.73	43.47
swiss42FSTCII_q3_g2_p1	1.45	3.68	0.27	7.02
swiss42FSTCII_q3_g2_p2	0.63	26.65	0.41	42.71
ulysses16FSTCII_q1_g1_p1	0.00	0.00	0.00	0.03
ulysses16FSTCII_q1_g1_p2	0.00	0.00	0.00	0.21
ulysses16FSTCII_q1_g2_p1	0.00	0.00	0.00	0.00
ulysses16FSTCII_q1_g2_p2	0.00	0.00	0.00	0.00
ulysses16FSTCII_q2_g1_p1	0.00	0.00	0.00	0.09
ulysses16FSTCII_q2_g1_p2	0.00	0.00	0.00	0.07
ulysses16FSTCII_q2_g2_p1	0.00	0.06	0.00	22.04
ulysses16FSTCII_q2_g2_p2	0.00	0.01	0.00	0.72
ulysses16FSTCII_q3_g1_p1	0.00	0.03	0.00	0.18
ulysses16FSTCII_q3_g1_p2	0.00	0.00	0.00	0.12
ulysses16FSTCII_q3_g2_p1	0.00	0.01	0.00	0.06
ulysses16FSTCII_q3_g2_p2	0.00	0.00	0.00	0.05
ulysses22FSTCII_q1_g1_p1	0.00	0.00	0.00	0.04
ulysses22FSTCII_q1_g1_p2	0.00	0.00	0.00	0.12
ulysses22FSTCII_q1_g2_p1	0.00	0.00	0.00	0.02
ulysses22FSTCII_q1_g2_p2	0.00	0.00	0.00	0.03
ulysses22FSTCII_q2_g1_p1	0.00	0.01	0.00	0.53
ulysses22FSTCII_q2_g1_p2	0.00	0.01	0.00	0.50
ulysses22FSTCII_q2_g2_p1	0.00	0.00	0.00	0.05
ulysses22FSTCII_q2_g2_p2	0.00	0.62	1.04	0.11
ulysses22FSTCII_q3_g1_p1	0.00	0.05	0.08	1.32
ulysses22FSTCII_q3_g1_p2	0.00	0.66	0.00	1.17
ulysses22FSTCII_q3_g2_p1	0.00	0.01	0.00	0.34
ulysses22FSTCII_q3_g2_p2	0.00	0.05	0.00	0.25

Bibliography

[1] E. Angelelli, C. Archetti, C. Filippi, and M. Vindigni. The probabilistic orienteering problem. *Computers and Operations Research*, 81:269–281, 2017.

[2] E. Angelelli, C. Archetti, and M. Vindigni. The clustered orienteering problem. *European Journal of Operational Research*, 238(2):404–414, 2014.

[3] C. Archetti, A. Hertz, and M. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2007.

[4] S. Basu. Tabu search implementation on traveling salesman problem and its variations: A literature survey. *American Journal of Operations Research*, 2:163–173, 2012.

[5] J.F. Bérubé, M. Gendreau, and J.Y. Potvin. An exact ϵ-constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. *European Journal of Operational Research*, 194:39–50, 2009.

[6] A. Blazinskas and A. Misevicius. Combining 2-opt, 3-opt and 4-opt with k-swap-kick perturbations for the traveling salesman problem. In *Proceedings of the 17th International Conference on Information and Software Technologies*, pages 50–401, April 2011.

[7] K.P. Bogart and P.G. Doyle. Non-sexist solution of the ménage problem. *The American Mathematical Monthly*, 93(7):514–518, 1986.

[8] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2015.

[9] L. Calvet, J. de Armas, D. Masip, and A.A. Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15:261–280, 2017.

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

[11] A.M. Campbell and B.W. Thomas. Probabilistic traveling salesman problem with deadlines. *Transportation Science*, 42(1):1–21, 2008.

[12] I. Chao, B. Golden, and E. Wasil. Theory and methodology - the team orienteering problem. *European Journal of Operational Research*, 88:475–489, 1996.

[13] X. Chou, L.M. Gambardella, P. Luangpaiboon, P. Aungkulanon, and R. Montemanni. 3-opt metaheuristics for the probabilistic orienteering problem. In *Proceedings of the 8th International Conference on Industrial Engineering and Applications (ICIEA 2021-Europe)*. Association for Computing Machinery, to appear.

[14] X. Chou, L.M. Gambardella, and R. Montemanni. Monte Carlo sampling for the probabilistic orienteering problem. In *Daniele P., Scrimali L. (eds) New Trends in Emerging Complex Real Life Problems*, volume 1 of *AIRO Springer Series*, pages 169–177. Springer, Cham, December 2018.

[15] X. Chou, L.M. Gambardella, and R. Montemanni. A metaheuristic algorithm for the probabilistic orienteering problem. In *MLMI 2019: Proceedings of the 2019 2nd International Conference on Machine Learning and Machine Intelligence*, pages 30–34. Association for Computing Machinery, 2019.

[16] X. Chou, L.M. Gambardella, and R. Montemanni. A tabu search algorithm for the probabilistic orienteering problem. *Computers and Operations Research*, 126:105107, 2021.

[17] X. Chou, U.J. Mele, L.M. Gambardella, and R. Montemanni. Re-initialising solutions in a random restart local search for the probabilistic orienteering problem. In *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 159–163. IEEE, 2020.

[18] X. Chou, R. Montemanni, and L.M. Gambardella. Monte Carlo sampling for the tourist trip design problem. *Millenium*, 10(2):83–90, 2019.

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

[20] M. Dorigo and L.M. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43(2):73–81, 1997.

[21] D. Feillet, P. Dejax, and M. Gengreau. Travelling salesman problems with profits. *Transportation Science*, 39:188–205, 2005.

[22] M. Fischetti, J. Salazar, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10:133–148, 1998.

[23] M.M. Flood. The traveling-salesman problem. *Operations Research*, 4:61–75, 1956.

[24] M. Gengreau, G. Laporte, and F. Semet. A branch-and-cut algorithm for the undirected selective travelling salesman problem. *Networks*, 32:263–273, 1998.

[25] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[26] B. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[27] A. Gunawana, H.C. Laua, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[28] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[29] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press Cambridge, MA, USA, 1992.

[30] H. H. Hoos and T. StÃ¼tzle. Generalised local search machines. In *Stochastic Local Search*, The Morgan Kaufmann Series in Artificial Intelligence, pages 113–147. Elsevier, 2005.

[31] F. Hutter, H.H. Hoos, and K. Leyton-Brown. Automated con- figuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202, 2010.

[32] F. Hutter, L. Xu, H.H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[33] T. Ilhan, S.M.R. Iravani, and M.S. Daskin. The orienteering problem with stochastic profits. *IIE Transactions*, 40(4):406–421, 2008.

[34] D. Johnson and L. McGeoch. The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, New York, 1997.

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

[36] R. Kramer, M. Modsching, and K. ten Hagen. A city guide agent creating and adapting individual sightseeing tours based on field trial results. *International Journal of Computational Intelligence Research*, 2(2):191–206, 2006.

[37] S. Kulturel-Konak, B.A. Norman, D.W. Coit, and A.E. Smith. Exploiting tabu-search memory in constrained problems. *INFORMS Journal on Computing*, 16(3):241–254, 2004.

[38] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete Applied Mathematics*, 26:193–207, 1990.

[39] Y.C. Liang, S. Kulturel-Konak, and M.H. Lo. A multiple-level variable neighborhood search approach to the orienteering problem. *Journal of Industrial and Production Engineering*, 30(4):238–247, 2013.

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

[41] A. Lodi and G. Zarpellon. On learning and branching: a survey. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 25(2):207–236, 2017.

[42] A. Lodi, G. Zarpellon, and P. Bonami. Learning a classification of mixed-integer quadratic programming problems. *Technical Report G-2017-106, Departement de Mathematiques et de Genie Industriel, Polytechnique Montreal, Canada*, 2016.

[43] E. Lucas. *Théorie des nombre*. Gauthier-Villars, Paris, 1891.

[44] U.J. Mele, X. Chou, L.M. Gambardella, and R. Montemanni. Reinforcement learning and additional rewards for the traveling salesman problem. In *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 170–176. IEEE, 2020.

[45] R. Montemanni, F. D'ignazio, X. Chou, and L.M. Gambardella. Machine learning and Monte Carlo sampling for the probabilistic orienteering problem. In *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 14–18. IEEE, 2018.

[46] R. Montemanni and L.M. Gambardella. Ant colony system for team orienteering problems with time windows. *Foundations of computing and Decision Sciences*, 34(4):287–306, 2009.

[47] R. Montemanni, D. Weyland, and L. M. Gambardella. An enhanced ant colony system for the team orienteering problem with time windows. In *2011 International Symposium on Computer Science and Society*, pages 381–384, 2011.

[48] V. Papapanagiotou, R. Montemanni, and L.M. Gambardella. Hybrid sampling-based evaluators for the orienteering problem with stochastic travel and service times. *Journal of Traffic and Logistics Engineering*, 3(2):108–114, 2015.

[49] G. Parascandolo, L. Buesing, J. Merel, L. Hasenclever, J. Aslanides, J.B. Hamrick, N. Heess, A. Neitz, and T. Weber. Divide-and-conquer Monte Carlo tree search for goal-directed planning. *arXiv:2004.11410*, 2020.

[50] R. Ramesh, Y. Yoon, and M. Karwan. An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4:155–165, 1992.

[51] G. Reinelt. TSPLIB - A Traveling Salesman Problem library. *ORSA Journal on Computing*, 3(4):267–384, 1991.

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

[53] R.G. Mbiadou Saleu, L. Deroussi, D. Feillet, N. Grangeon, and A. Quilliot. An iterative two-step heuristic for the parallel drone scheduling traveling salesman problem. *Networks*, 72(4):459–474, 2018.

[54] P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical report*, 1997.

[55] D. Silver, A. Huang, C. Maddison, and al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[56] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. Vanden Berghe, and D. Van Oudheusden. A personalised tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence*, 22(10):964–985, 2008.

[57] D.F. Specht. A general regression neural network. *IEEE Transactions on Neural Netoworks*, 2(6):568–576, 1991.

[58] H. Tang and E. Miller-Hooks. Algorithms for a stochastic selective travelling salesperson problem. *Journal of the Operational Research Society*, 56(4):439–452, 2005.

[59] H. Tang and E. Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computer and Operations Research*, 32:1379–1407, 2005.

[60] P. Vansteenwegen and D. Van Oudheusden. The mobile tourist guide: An or opportunity. *OR Insights*, 20(3):21–27, 2007.

[61] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. Metaheuristics for tourist trip planning. *Lecture Notes in Economics and Mathematical Systems*, pages 15–31, 2009.

[62] P. Vansteenwegen, W. Souffriau, Vanden Berghe G, and D. Van Oudheusden. Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research*, 36(12):3281–3290, 2009.

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

[64] J. Verzani. *Using R for Introductory Statistics, Second Edition*. CRC Press, CUNY/College of Staten Island, New York, USA, 2014.

[65] G. Villarrubia, J. F. De Paz, P. Chamoso, and F. De la Prieta. Artificial neural networks used in optimization problems. *Neurocomputing*, 272:10–26, 2018.

[66] Q. Wang, B. Golden, and E. Wasil. Using a genetic algorithm to solve the generalized orienteering problem. In *B. Golden, S. Raghavan, E. Wasil (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 263–274, 2008.

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