Erasmus School of Economics

Bachelor Thesis Double Degree Programme
Econometrics & Economics

July 7th 2018

Extending the sample average approximation method to the traveling salesman problem with random travel times and soft time windows

Author:	Reinier van Uden (415366)
Supervisor:	T.R. Visser MSc
Second Supervisor:	Dr. R. Spliet

Abstract

In this thesis, we apply the sample average approximation method to stochastic routing problems. We replicate the results by Verweij et al. (2003) with respect to the shortest path problem with random travel times and the traveling salesman problem with random travel times. In addition, we propose a branch-and-cut framework to apply the sample average approximation method to the traveling salesman problem with random travel times and soft time windows. Different approaches for eliminating subtours are evaluated. We find that we can solve the problems using less memory by adding subtour elimination constraints in fractional branch-and-bound nodes.

1 Introduction

The subjects of this thesis are the shortest path problem, traveling salesman problem and traveling salesman problem with soft time windows, all with uncertain travel times. In each problem, the objective is to minimize the sum of the deterministic routing cost and the expected value of possible delays. Each problem is formulated as a two-stage stochastic problem. We adopt the Sample Average Approximation method, as described in Verweij et al. (2003). We replicate the results of Verweij et al. (2003) and extend their methodology to the traveling salesman problem with random travel times and soft time windows.

The Traveling Salesman Problem (TSP) is one of the most famous problems in combinatorial optimization. This problem is part of a more general class of problems, called Vehicle Routing Problems (VRPs). It consists of finding a Hamiltonian cycle that visits each of a specified set of destinations and minimizes travel cost. Early formulations and solution approaches were presented by Dantzig et al. (1954) and Flood (1956). Since then, the TSP has been studied extensively and it has been expanded to include additional constraints such as limited vehicle capacity or customer time windows. The academic interest is mainly due to the fact that this problem arises often in practice. The practical relevance of the traveling salesman problem with time windows has increased over the past few decades. The growth of the internet has given rise to many online sales of products which increases the need for efficient deliveries of these orders. A delivery vehicle often starts and ends at a depot and an efficient route is desired, visiting all demand locations. Usually, customers can choose a time window in which they would like the product to be delivered. The delivery company will try to satisfy these customer preferences as much as possible. This type of problem also arises in supermarkets, delivering groceries at its customers' homes. In this thesis, we contribute to the academic literature on this subject.

In combinatorial optimization, many other routing problems have been considered, for instance the Shortest Path Problem (SPP) and Vehicle Routing Problem. Most research has focused on solving deterministic problem instances. However, uncertainty often plays a role in practice. For example, traffic congestion caused by road works or accidents appears randomly, meaning that the traveling time on certain roads is variable. Hence, it is difficult to determine a priori the shortest path from one destination to another.

In this thesis, we address two types of Stochastic Routing Problems (SRPs), namely the shortest path problem with random travel times (SPPRT) and the traveling salesman problem with random travel times (TSPRT). The traveling times follow a discrete probability distribution. These SRPs are formulated using two stages. This means that two decisions have to be made at different points in time. The first decision is made when the traveling times are uncertain and only their distributions are known. This decision involves selecting a route (either a path or tour) and this is called the first-stage problem. The second decision is made after the traveling times have been realized. For the given traveling times, the route determined in the first stage may not be optimal or feasible anymore. Therefore, an adjustment, called a recourse, can be made to the path or tour. The objective is to minimize the total of the first-stage routing costs and the expected recourse cost. In the formulation of our problems, the recourse cost is defined as a penalty if the traveling time exceeds a given deadline, like in Verweij et al. (2003) and Laporte et al. (1992).

A realization of the stochastic traveling time on each arc combined is called a scenario. If for each arc there is a number of possible traveling times and if the number of arcs grows large, a huge number of possible scenarios exist. Each scenario has a certain probability of occurring, meaning that the expected recourse cost can be expressed as a weighted sum of the recourse cost in each scenario. Thus, the objective function becomes a complicated minimization problem, where exact algorithms are difficult to apply. To solve the problem, we apply an exterior sampling approximation, called the Sample Average Approximation (SAA). First, we generate multiple samples of scenarios using the Monte Carlo method. In each sample, the expected recourse function is approximated by the sample average. Hence, for each sample a deterministic problem instance can be solved. Kleywegt et al. (2002) show that the optimal solution of the approximating problem converges to the optimal solution of the original stochastic problem.

In this thesis, we first replicate most of the results derived by Verweij et al. (2003). To be more precise, we apply the SAA method to the SPPRT and TSPRT, where the arcs have deterministic costs and random travel times. In the SAA method, deterministic problem instances are solved for each generated sample. For solving these non-stochastic problem instances, a branch-and-cut framework is used, based on decomposition. We see a decrease in computation time compared to

the results in Verweij et al. (2003). Computing power has increased strongly over the last 15 years and commercial MIP solvers have made several advancements.

In addition, we apply the SAA methodology to a more involved version of the TSPRT, namely with time windows included. Each destination should be visited within a pre-specified time interval. We deal with soft time windows, meaning that a penalty is incurred if a destination is visited too late. If the destination is visited too early, the vehicle has to wait. Such constraints arise often in practice but the literature on this problem with random travel times is limited. Moreover, as far as we know, the SAA method has not been applied to such problems yet. We apply the SAA and evaluate its performance. We find that it is difficult to solve the deterministic problems for each sample in the SAA to optimality. The algorithm is applied to a routing problem with one vehicle without capacity constraints. In addition, it could be utilized to solve VRPs with stochastic travel times with m vehicles within heuristic procedures, for instance to find efficient routes per vehicle after customers have been clustered.

The structure of this thesis is as follows. Section 2 gives an overview of the literature on SRPs and the solution approaches that have been applied. In Section 3 the problems addressed in this thesis are defined and Section 4 describes the solution methods. Section 5 gives a description of how the problem instances are generated. Next, Section 6 reports the results for our computations. Section 7 concludes.

2 Literature review

Stochastic Routing Problems (SRPs) have received considerably less attention in the literature than their deterministic counterparts. For most deterministic routing problems, many different solution algorithms exist. For instance, the SPP can be solved in polynomial time using Dijkstra's algorithm, given there are no negative-cost cycles in the graph (Dijkstra, 1959). For the TSP, several solution approaches have been introduced. Because the TSP is well known to be an $\mathcal{NP}$-hard problem (Laporte, 1992), both exact methods and heuristic procedures have been proposed. Most exact procedures use branch-and-bound algorithms, which begin with a linear programming relaxation. Such algorithms represent the solution space by a tree, in which each branch represents a subspace of the solution space and each node is a candidate solution. The algorithm explores the branches to find the optimal integer solution and an upper and lower bound are computed at each node. Branches can be pruned if the lower and upper bound of that branch make it impossible that a better solution will be found than the current best solution. Sometimes, difficult constraints are left out at the beginning of the branch-and-bound but they are added iteratively during the solution process (Laporte, 1992). This if often referred to as branch-and-cut. Notable heuristics for the TSP are the one proposed by Christofides (1976) and the algorithm of Lin and Kernighan (1973).

Solution methods for routing problems are not only helpful in logistics, but can be applied more generally. Some scheduling problems can be formulated as a TSP. For example, if n jobs need to be performed on a single machine and change-over times exist between different jobs, then the most efficient sequence of jobs can be found using TSP solution methods (Laporte, 1992).

In real-world applications of routing problems, more problem aspects have to be taken into account. For example, at the time of planning a route, some characteristics might not be known with certainty yet. In the literature, different cases of uncertainty have been investigated. The most studied variant of SRP is probably the vehicle routing problem with stochastic demands (VRPSD). In this problem, a set of least-cost routes should be determined using a number of vehicles with limited capacity while the demand of customers is uncertain. The first heuristic solution method was introduced by Tillman (1969). Stewart and Golden (1983) give another heuristic solution method. They also provide different formulations for the problem, with either random variables in the objective function or in the constraints.

These two types of formulations are very common in stochastic programming problems. When random variables are present in the constraints only, we use the term Chance Constrained Programming (CCP). When random variables occur in the objective function, the problem is often modeled using two stages. In the first stage, a decision should be made when only the distributions of the random variables are known and in the second stage a so-called recourse action must be taken after the random variables are realized. For example, in the VRPSD, a route determined in the first stage may turn out to be infeasible if the realized demand on that route is higher than the vehicle capacity. If the original route is executed, a recourse action is required, namely to return to

the depot somewhere on the route. This is necessary in order to feasibly serve the next customers on the route by the capacity limited vehicle. In this case, the recourse cost is the extra distance that has to be traveled due to the recourse action.

A different approach for the recourse function is to define it as the penalty for exceeding a given deadline on the route duration. For example, a penalty for late route completions arises naturally in the paper by Lambert et al. (1993) from a banking context. Money collection routes are designed in the presence of stochastic travel times, while late arrivals at the depot imply that all money contained in the vehicle loses one day's interest.

Another common SRP is the vehicle routing problem with stochastic customers (VRPSC), in which each location has a certain probability of requiring a visit, but the demands are deterministic. The traveling salesman problem with stochastic customers, sometimes referred to as the probabilistic traveling salesman problem (PTSP), was introduced by Jaillet (1985). This study presents some interesting properties of the problem, particularly that an optimal solution to this problem defined in a plane, may cross itself, contrary to optimal solutions to deterministic TSPs. An exact solution approach, proposed by Laporte et al. (1994), uses the integer L-shaped method. The main idea of this method is that the two stages of the problem are separated. When solving the second-stage problem, the first-stage solution is taken as given and a lower bound for the objective of the second-stage problem is computed. Each solution of the second stage adds a constraint, called an optimality cut, to the first-stage problem, improving its solution. When solving the first-stage problem, the second-stage objective is approximated by a constant. This constant should satisfy a number of constraints, namely the optimality cuts. We iteratively solve the first- and second-stage problems until a stopping criterion is met. This procedure can be applied to mixed integer problems in general. Then, it is often called the Benders decomposition method (Benders, 1962). It is explained in more detail in Section 4.2. With their approach, Laporte et al. (1994) can solve instances of up to 50 locations to optimality.

A heuristic method for the PTSP was introduced by Bianchi et al. (2005). It uses local search, which means small adjustments are made to feasible tours to see if improvements are possible. A combination of stochastic customers and stochastic demands has been investigated for the vehicle routing problem by Gendreau et al. (1995). For a survey of SRPs, we refer to Gendreau et al. (1996).

An SRP that has received considerably less attention in the literature is the TSP with random travel times and time windows. Jula et al. (2006) study a TSP with time windows in which the travel times and service times are stochastic processes. The time windows do not have to be satisfied always, but a route is considered feasible if the probability of arriving at each destination within its time window is greater than a certain value. The solution method is based on dynamic programming and finds an approximate solution because the distribution of the arrival time at each node is approximated using the first and second moment of these random variables. Tas et al. (2014) examine a more general problem, namely the VRP with soft time windows and stochastic travel times. In their problem, the distribution of the travel times is time-dependent. The objective is to minimize a convex combination of the time window violations and the vehicle costs. The problem is solved using different types of local search. The VRP with deterministic time-dependent travel times is more often studied, see Hill & Benton (1992) and Donati et al. (2008).

Several solution approaches have been applied to SRPs, most of which can be applied to stochastic programming problems in general. We focus on those that can be applied to two-stage stochastic programs, where in the second stage a recourse decision should be taken. For cases with a small number of scenarios, exact mathematical programming techniques have been applied in which the expected recourse cost is evaluated exactly. The most notable solution approach is the integer L-shaped method, which has been applied to solve all kinds of stochastic VRPs, see Laporte et al. (1992) and Gendreau et al. (1995).

For problem instances with a large number of scenarios, exact algorithms are intractable and hence most approaches for these cases are based on sampling. One of the first approximations is the Stochastic Approximation (SA), which was introduced by Robbins and Monro (1951). The classical SA algorithm mimicks a simple subgradient descent method by iteratively calculating subgradients of the expected recourse function and calculating the projection unto the set of feasible solutions X, while sampling scenarios in each iteration (Nemirovski et al., 2009). It performs well for a certain class of stochastic problems with convex objective functions. This method can be classified as an interior sampling approach, since the samples are modified during the optimization process.

Other interior sampling methods modify samples within the integer L-shaped method for stochastic linear programming problems (Slyke & Wets, 1969).

Opposed to interior sampling methods, exterior sampling procedures first generate multiple samples of scenarios according to the (known) probability distributions of the random variables. Each sample consists of a number of realizations of the random variables and therefore, a deterministic optimization problem can be solved for each sample. We refer to this deterministic problem as the SAA MIP. An advantage of these methods is that solution algorithms are known for the deterministic problems. A very common exterior sampling method is the Sample Average Approximation (SAA) method. This approach is often applied to two-stage stochastic problems. The SAA method approximates the expected value of the recourse function for each generated sample by the sample average function. The resulting deterministic problem is then solved in each sample. The optimal solution for each sample is an approximate solution to the original problem. The statistical properties and convergence rate for this method, when applied to discrete optimization problems, are discussed in Kleywegt et al. (2002). Because of its intuitive concept and wide applicability, it has been used in various disciplines. For example, it has been employed in finance for solving asset investment problems (Blomvall & Shapiro, 2006) and in economics to solve stochastic Nash Equilibrium problems (Li, 2014). Its use is not restricted to two-stage stochastic problems, but can also be applied to expected value or chance constrained programs (Wang & Ahmed, 2008; Pagnoncelli et al., 2009). SAA methods for SRPs are investigated by Kenyon and Morton (2003), Verweij et al. (2003) and Ahmed and Shapiro (2002). For a recent survey on the applications of the SAA method, we refer to Emelogu et al. (2016).

One drawback of the SAA method is the difficulty in choosing the appropriate sample size. This is important, because a size that is too small will lead to low-quality solutions whereas a sample size that is too large will lead to a high computational burden (Emelogu et al., 2016). While some authors keep the sample size constant throughout the optimization process (Verweij et al., 2003; Nemirovski et al., 2009), others dynamically increase the number of scenarios in a sample (Balaprakash et al., 2009). More recently, it was suggested to use clustering techniques to dynamically update the sample size, leading to significant savings in computation time (Emelogu et al., 2016). The basic idea of clustering is to generate a very large sample, divide it into clusters and use the averages of the clusters as the sample. In this way, some information of the large sample is included, but a smaller SAA MIP has to be solved. Other authors improve the performance of the SAA by using more efficient sampling procedures, for example using Latin hypercube designs (Tang & Qian, 2010). Latin hypercube sampling divides the sample space in disjoint regions and the number of random variables generated from each region is proportional to the probability of that region (Homem-de Mello, 2008). This ensures that the realizations in the sample are spread more evenly across all possible values.

3 Problem description

In this section, we formulate the problems that we investigate in this thesis. We first give a general formulation of two-stage stochastic integer programs. Next, we formulate the shortest path problem with random travel times (SPPRT) and the traveling salesman problem with random travel times (TSPRT). We also explain why the recourse function we use there is relevant and interesting. Finally, we extend the TSPRT to include soft time windows.

3.1 Two-stage SRPs

The two-stage formulation is an important class of stochastic programs. In such problems, two decisions have to be made at different points in time. The first decision is made when only the distributions of the random variables are known. This decision involves selecting a route (either a path or tour) and this is called the first-stage problem. The second decision is made after the random variables have been realized. For the given realizations, the route determined in the first stage may not be optimal or feasible. Therefore, an adjustment, called a recourse, can be made to the path or tour. The objective is to minimize the total of the costs of the first-stage decision and the expected recourse cost in the last stage.

Let Ω be the set of scenarios ω. We assume the random variables $\omega \in \Omega$ have a discrete probability distribution $\mathcal{P}$ resulting in a finite set of possible scenarios Ω. The decision vector x denotes the first-stage decision, which has to be determined a priori. When the scenario ω is

known, the recourse decision vector y is determined. Let the set X contain the set of feasible solutions in the first stage, involving several constraints. A general formulation of a two-stage stochastic integer programming problem is

$$z^* = \min_{x \in X} c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))], \tag{1}$$

where c denotes the cost and the recourse function $Q(x, \xi(\omega))$ is given by

$$Q(x, \xi(\omega)) = \min_{y \geq 0} \{q(\omega)^T y | Dy \geq h(\omega) - T(\omega)x\}. \tag{2}$$

The recourse function $Q(x, \xi(\omega))$ gives the optimal objective value of the second-stage recourse problem for a given first-stage solution x and the parameters $\xi(\omega) = (q(\omega), h(\omega), T(\omega))$.

3.2 Shortest path problem with random travel times

The SPPRT is defined on a directed graph $G = (V, A)$, with node set V and arc set A. The goal is to find a least-cost path from a source node $s \in V$ to a sink node $t \in V$. The cost consists of two parts. The first part are arc costs, determined by a cost vector $c \in \mathbb{R}^{|A|}_{\geq 0}$. The second part is the recourse cost, which measures the extent to which the actual total traveling time exceeds the deadline $\kappa \in \mathbb{R}$. The vector of random travel times is given by $\xi(\omega) \in \mathbb{R}^{|A|}_{\geq 0}$ for a scenario ω from the set of scenarios Ω with associated probability distribution $\mathcal{P}$. A path is defined as a selection of arcs, meaning that for each arc $(i, j) \in A$ there is decision variable x_{ij} with value 1 if the arc is included in the path and value 0 otherwise. Hence, the path is represented by the decision vector $x \in \{0, 1\}^{|A|}$. Using this notation, the SPPRT can be formulated as follows:

$$\min_x c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))] \tag{3}$$

$$\text{subject to} \sum_{j \in V \setminus \{i\}} x_{ij} - \sum_{j \in V \setminus \{i\}} x_{ji} = \begin{cases} 1, & \text{for } i = s \\ -1, & \text{for } i = t \\ 0, & \text{for } i \in V \setminus \{s, t\} \end{cases} \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A, \tag{5}$$

where the recourse function $Q(x, \xi(\omega))$ for a given first-stage solution x and scenario $\omega \in \Omega$ is given by

$$Q(x, \xi(\omega)) = \max\{\xi(\omega)^T x - \kappa, 0\}. \tag{6}$$

Constraints (4) ensure that the solution x is indeed an $s - t$ path. Note that the arc costs are defined to be nonnegative ($c \in \mathbb{R}^{|A|}_{\geq 0}$). This implies that an optimal solution will not contain any cycles and therefore, we do not need to formulate any constraints to eliminate them. The recourse function (6) can be interpreted as a penalty for exceeding the deadline κ on the route duration.

The recourse function we use is the same as in Verweij et al. (2003). This recourse function has an intuitive interpretation and does not seem to involve an optimization problem as shown in (2). We will now show how this interpretation arises from an optimization problem of the form (2). Laporte et al. (1992) show how to use the general formulation for this type of recourse for a problem with multiple vehicles. Since we use only one vehicle, we let y be a variable with one element. For our specific problems, $q(\omega)$, D and $h(\omega)$ are non-random. More specifically, we set $q(\omega) = 1$, $D = 1$ and $h(\omega) = -\kappa$ for some $\kappa \in \mathbb{R}$. Let $T(\omega)$ be a vector with the negative of the traveling time for each arc, so we can write $T(\omega) = -\xi(\omega)$. Then the recourse function can be written as

$$Q(x, \xi(\omega)) = \min_{y \geq 0} \{y | y \geq -\kappa - T(\omega)^T x\} = \min_{y \geq 0} \{y | y \geq \xi(\omega)^T x - \kappa\} \tag{7}$$

for which the solution for a given scenario ω and first-stage solution x, is given by $y = \max\{\xi(\omega)^T x - \kappa, 0\}$. Hence, the recourse function can be seen as the penalty for exceeding a certain deadline κ on the route duration.

From a mathematical point of view, it is interesting to formulate the recourse function with a deadline. If instead, we would minimize the total expected travel time in the second stage, the objective function would become

$$c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))] = c^T x + \mathbb{E}_{\mathcal{P}}[\xi(\omega)^T x]. \tag{8}$$

However, the expectation now becomes linear in ξ because we can write

$$\mathbb{E}_{\mathcal{P}}[\xi(\omega)^T x] = \mathbb{E}_{\mathcal{P}}[\sum_{(i,j)} \xi_{ij}(\omega) \cdot x_{ij}] = \sum_{(i,j)} \mathbb{E}_{\mathcal{P}}[\xi_{ij}(\omega)] x_{ij}.$$

As shown above, we do not have to deal with stochasticity in this case because the random travel times can be replaced by their means. This is often called the Mean Value Problem (MVP). The MVP also leads to an optimal solution when the deadline κ is either very small or very large, because then the expected violation of the deadline becomes linear in $\xi(\omega)$ as well. On the other hand, for other values of κ, Verweij et al. (2003) show that optimal solutions to the MVP can be arbitrarily bad compared to the optimal solution of the stochastic problem. For these cases, we require mathematical algorithms, such as the SAA, that can cope with the stochasticity of the variables.

3.3 Traveling salesman problem with random travel times

The TSPRT is defined on an undirected graph $G = (V, E)$ with node set V and edge set E. Because the graph is undirected, $\{i, j\} \in E$ is only defined for $i < j$. The goal is to find a least-cost tour which visits all nodes in V. The costs are made up of edge costs, determined by a cost vector $c \in \mathbb{R}_{\geq 0}^{|E|}$, and the recourse cost, determined by the vector of random travel times $\xi(\omega) \in \mathbb{R}_{\geq 0}^{|E|}$ with associated probability distribution $\mathcal{P}$ and deadline $\kappa \in \mathbb{R}$. A tour is defined as a selection of edges, meaning that to each edge $\{i, j\} \in E$ there corresponds a decision variable x_{ij} with value 1 if it is included in the tour and value 0 otherwise. Hence, the tour is represented by a decision vector $x \in \{0, 1\}^{|E|}$. Using this notation, the TSPRT can be formulated in the following way:

$$\min_x c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))] \tag{9}$$

$$\text{subject to} \sum_{j \in V : \{i,j\} \in E} x_{ij} + \sum_{j \in V : \{j,i\} \in E} x_{ji} = 2, \quad \forall i \in V, \tag{10}$$

$$\sum_{\{i,j\} \in E : i \in S, j \notin S} x_{ij} + \sum_{\{j,i\} \in E : i \in S, j \notin S} x_{ji} \geq 2, \quad \forall S \subset V \text{ with } |S| \geq 2, \tag{11}$$

$$x_{ij} \in \{0, 1\}, \quad \forall \{i, j\} \in E, \tag{12}$$

where the recourse function $Q(x, \xi(\omega))$ for a given first-stage solution x and scenario $\omega \in \Omega$ is given by

$$Q(x, \xi(\omega)) = \max\{\xi(\omega)^T x - \kappa, 0\}.$$

Constraints (10) ensure that each node in V is connected to two edges and (11) make sure that subtours are eliminated. The recourse function is identical to the one used for the SPPRT.

3.4 TSPRT with soft time windows

We consider an extension to the TSPRT, namely the traveling salesman problem with random travel times and soft time windows (TSPRTTW). In contrast to the TSPRT, the problem is defined on a directed graph $G = (V, A)$. Also, we define a starting point for the tour, the so-called depot. This starting point is labeled node 0, which is included in V. For the formulation, we will also use the set $V' = V \cup \{n + 1\}$ where the node $n + 1$ is an extra node for the depot. Furthermore, the arc set A' is the same as the set A, except for the arcs from and to the depot. The outarcs of the depot leave from node 0 whereas the inarcs of the depot go to node $n + 1$. The TSPRTTW is defined as finding a least-cost tour, visiting each node in the graph, while each node should be visited within a pre-specified time window or otherwise a penalty is incurred. Therefore, we need to know in which direction an arc (i, j) is traveled which is the reason for using the directed graph. Let $[a_i, b_i]$ denote the time interval during which node $i \in V'$ should be visited, with $a_i \in \mathbb{R}_{\geq 0}$ and $b_i \in \mathbb{R}_{\geq 0}$. In addition, the decision variable $y_i \in \mathbb{R}$ represents the time at which node i is visited and the decision variable z_i is defined as the violation of the latest arrival time b_i at node $i \in V'$. Furthermore, $t_{ij}(\omega)$ denotes the travel time on arc $(i, j) \in A'$ for the scenario $\omega \in \Omega$.

The formulation is given below. The timing constraints in the second-stage problem are based on the formulation given by Desrosiers et al. (1988) for the VRP with time windows.

$$\min_x c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))] \tag{13}$$

$$\text{subject to} \sum_{j \in V' \setminus \{i\}} x_{ij} = 1, \quad \forall i \in V' \setminus \{n+1\}, \tag{14}$$

$$\sum_{j \in V' \setminus \{i\}} x_{ji} = 1, \quad \forall i \in V' \setminus \{0\}, \tag{15}$$

$$\sum_{(i,j) \in A' : i \in S, j \notin S} x_{ij} \geq 1, \quad \forall S \subset V \text{ with } |S| \geq 2, \tag{16}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in A, \tag{17}$$

where the recourse function $Q(x, \xi(\omega))$ is defined as

$$Q(x, \xi(\omega)) = \min_z \sum_{i \in V'} z_i \tag{18}$$

$$y_j - y_i \geq t_{ij}(\omega) + M(x_{ij} - 1), \quad \forall i, j \in V', \text{ s.t. } (i,j) \in A', \tag{19}$$

$$y_i \geq a_i, \quad \forall i \in V', \tag{20}$$

$$z_i - y_i \geq -b_i, \quad \forall i \in V', \tag{21}$$

$$z_i \geq 0, \quad \forall i \in V'. \tag{22}$$

The objective is to minimize the cost of the tour and the sum of the expected penalties for arriving too late at nodes. Arriving too early does not incur a penalty, but the vehicle must wait a location i until a_i due to constraints (20). Constraints (14) ensure that each node is visited and (19) make sure that, if we travel from i to j, j is visited after i. To be more precise, if $x_{ij} = 1$ the constraint requires that $y_j \geq y_i + t_{ij}$. So j can be visited no earlier than the time we arrived at i plus the traveling time from i to j. Additionally, this set of constraints ensure increasing arrival times at each node along a tour, making subtours infeasible (Desrosiers et al., 1988). The constant M is taken large enough to make sure that the constraints are always satisfied if $x_{ij} = 0$. For the depot node, we define an extra variable y_{n+1}, besides y_0, to make sure that we can leave from the depot at a certain time and arrive at a much later time. Using only one variable corresponding to the depot, constraints (19) would make any tour of positive length infeasible. Therefore, we use $y_0 = 0$ to denote the time we leave the depot and y_{n+1} for the arrival time. Additionally, we set $a_0 = b_0 = 0$ and $a_{n+1} = 0$.

Although constraints (19) do not allow for any subtours in the second stage, constraints (16) are necessary in the solution approach we apply. We need to make sure that the first-stage solution does not contain any subtours because otherwise, no feasible solution to the second-stage problems exists. This will be explained in more detail in Section 4.2.

4 Solution methods

In this section, we explain how we apply the Sample Average Approximation (SAA) method to the SPPRT, TSPRT and TSPRTTW by describing how it can be applied to a general form of two-stage SRPs. When using the SAA, many deterministic mixed integer programs need to be solved. In order to find solutions efficiently, Benders decomposition is applied. We will explain this method in general and show how it can be applied to the SPPRT, TSPRT and TSPRTTW. In the final part of this section, we summarize the branch-and-cut framework, which we use to implement Benders decomposition.

4.1 Sample Average Approximation method

We apply the SAA to a general two-stage SRP described by the first-stage problem (1) and the second-stage problem (2) in Section 3.1. The scenarios $\omega \in \Omega$ follow a discrete probability

distribution $\mathcal{P}$, given by $\{p_1, p_2, \ldots, p_{|\Omega|}\} \in [0,1]^{|\Omega|}$. Consequently, the expected recourse cost $\mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))]$ can be written as follows:

$$\mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))] = \sum_{k=1}^{|\Omega|} p_k Q(x, \xi(\omega_k)). \tag{23}$$

Solving SRPs with such a second-stage objective function becomes very difficult if the number of scenarios grows large, because for a given x, the solution of numerous second-stage optimization problems is required. The following example illustrates that the number of scenarios can grow exponentially in the dimension of x: given a graph of 100 arcs, where the parameter for each arc has 3 potential values. This gives $|\Omega| = 3^{100}$ possible scenarios, which means the computational effort of evaluating function (23) is immense (Verweij et al., 2003). In addition, since $\mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))]$ is a function of $x \in \mathbb{R}^d$ while X involves integrality constraints on x, Verweij et al. (2003) show that the SRP involves minimizing a piecewise linear objective function over an integer feasible region which can be very challenging in itself.

Therefore, the SAA simplifies the problem by using Monte Carlo sampling. M independent samples of a certain size N are generated using the probability distribution $\mathcal{P}$. A generated scenario n is denoted as ω^n so a sample can be represented by $\{\omega^1, \omega^2, \ldots, \omega^N\}$. For each sample, the expected recourse function in (23) is approximated by the sample average function $\frac{1}{N}\sum_{n=1}^{N} Q(x, \xi(\omega^n))$. The following problem, called the SAA MIP, is then solved for each sample:

$$z_N = \min_{x \in X} c^T x + \frac{1}{N}\sum_{n=1}^{N} Q(x, \xi(\omega^n)). \tag{24}$$

The solution $\hat{x}$ to the SAA MIP (24) and the corresponding objective value z_N for each sample are an approximation of the true solution x^* to equation (1) with objective value z^* respectively. This gives M candidate solutions $\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^M$ and objective values $z_N^1, z_N^2, \ldots, z_N^M$. It is proven in Mak et al. (1999) that, in expectation, the average of these objective values, $\bar{z}_N$, is a lower bound for the true objective value z^*. In other words, it holds that

$$\mathbb{E}[\bar{z}_N] = \mathbb{E}[\frac{1}{M}\sum_{m=1}^{M} z_N^m] \leq z^*. \tag{25}$$

As the samples are all generated from the same distribution $\mathcal{P}$,

$$\mathbb{E}[\frac{1}{M}\sum_{m=1}^{M} z_N^m] = \mathbb{E}[z_N^m], \tag{26}$$

so it suffices to prove that $\mathbb{E}[z_N^m] \leq z^*$ for $m = 1, \ldots, M$. For a given sample, we denoted the objective value of the SAA MIP by z_N in (24). Therefore, proving that $\mathbb{E}[z_N] \leq z^*$ gives the desired result. We highlight the proof given by Mak et al. (1999) below and we add some intermediate steps.

Theorem 1. *Given a two-stage stochastic program $z^* = \min_{x \in X} c^T x + \mathbb{E}_{\mathcal{P}}[Q(x, \xi(\omega))]$ and an i.i.d. sample $\{\omega^1, \omega^2, \ldots, \omega^N\}$ from the discrete probability distribution $\mathcal{P}$, then*

$$\mathbb{E}[z_N] = \mathbb{E}[\min_{x \in X} c^T x + \frac{1}{N}\sum_{n=1}^{N} Q(x, \xi(\omega^n))] \leq z^*.$$

Proof. First we rewrite

$$z_N = \mathbb{E}[\min_{x \in X} c^T x + \frac{1}{N}\sum_{n=1}^{N} Q(x, \xi(\omega^n))] = \mathbb{E}[\min_{x \in X} \frac{1}{N}\sum_{n=1}^{N} \big(c^T x + Q(x, \xi(\omega^n))\big)].$$

Because the random variables $\omega^1, \omega^2, \ldots, \omega^N$ are identically distributed and the probability distribution $\mathcal{P}$ is discrete, we can write

$$\mathbb{E}[\min_{x \in X} \frac{1}{N}\sum_{n=1}^{N} \big(c^T x + Q(x, \xi(\omega^n))\big)] = \sum_{k=1}^{|\Omega|} p_k [\min_{x \in X} \frac{1}{N}\sum_{n=1}^{N} \big(c^T x + Q(x, \xi(\omega_k))\big)]$$

If each term in this weighted sum is minimized separately, the objective value can be made at least as low compared to the case where we minimize the total of the weighted sum, meaning that

$$\sum_{k=1}^{|\Omega|} p_k[\min_{x\in X} \frac{1}{N}\sum_{n=1}^{N}\left(c^T x + Q(x,\xi(\omega_k))\right)] \leq \min_{x\in X}\sum_{k=1}^{|\Omega|} p_k[\frac{1}{N}\sum_{n=1}^{N}\left(c^T x + Q(x,\xi(\omega_k))\right)]$$

Finally, we rewrite

$$\min_{x\in X}\sum_{k=1}^{|\Omega|} p_k[\frac{1}{N}\sum_{n=1}^{N}\left(c^T x+Q(x,\xi(\omega_k))\right)] = \min_{x\in X}\mathbb{E}[\frac{1}{N}\sum_{n=1}^{N}\left(c^T x+Q(x,\xi(\omega_k))\right)] = \min_{x\in X}\mathbb{E}[c^T x+Q(x,\xi(\omega))]$$

$$= \min_{x\in X} c^T x + \mathbb{E}[Q(x,\xi(\omega))] = z^*.$$

It follows that $\mathbb{E}[z_N] \leq z^*$ for a given two-stage stochastic program $z^* = \min_{x\in X} c^T x + \mathbb{E}_{\mathcal{P}}[Q(x,\xi(\omega))]$ and a given i.i.d. sample $\{\omega^1, \omega^2, \ldots, \omega^N\}$ from the discrete probability distribution $\mathcal{P}$. □

Besides a lower bound, the candidate solutions obtained from the sample can also be used to obtain an upper bound on z^* by evaluating the objective function. Specifically, we know that for the solution $\hat{x}^m$ of each sample m it holds that $c^T\hat{x}^m + \mathbb{E}[Q(\hat{x}^m,\xi(\omega))] \geq z^*$. Evaluating the expected recourse function over all possible scenarios in Ω can require a large computational effort. Therefore, we generate another sample of size N', independent of the M other samples, in order to derive an unbiased estimate of z^*. This estimate is given by

$$z_{N'}(\hat{x}^m) = c^T\hat{x}^m + \frac{1}{N'}\sum_{n=1}^{N'}[Q(\hat{x}^m,\xi(\omega^n))]. \tag{27}$$

N' can be taken much larger than N because evaluating the recourse function for a given first-stage solution generally requires much less computational effort than optimizing it. Because $z_{N'}(\hat{x}^m)$ is an unbiased estimator of the true objective value corresponding to $\hat{x}^m$ given by $c^T\hat{x}^m + \mathbb{E}[Q(\hat{x}^m,\xi(\omega))]$, we know that, for each candidate solution $\hat{x}^m$, it holds that

$$\mathbb{E}[z_{N'}(\hat{x}^m)] \geq z^* \tag{28}$$

Since we want the upper bound to be as tight as possible, we choose the candidate solution from the sample m for which the unbiased estimator $z_{N'}(\hat{x}^m)$ is lowest. Let this solution be denoted by $\hat{x}^*$ As for a given random variable, the sample variance of an i.i.d. sample is an unbiased estimator of the true variance, we can estimate the variance of the lower bound $\bar{z}_N$ by

$$\hat{\sigma}^2_{\bar{z}_N} = \frac{1}{(M-1)M}\sum_{m=1}^{M}(z_N^m - \bar{z}_N)^2 \tag{29}$$

and the variance of the upper bound by

$$\hat{\sigma}^2_{\hat{z}_{N'}(\hat{x}^*)} = \frac{1}{(N'-1)N'}\sum_{n=1}^{N'}[c^T\hat{x}^* + Q(\hat{x}^*,\xi(\omega^n)) - \hat{z}_{N'}(\hat{x}^*)]^2. \tag{30}$$

The optimality gap of the solution $\hat{x}^*$ can be computed as

$$z_{N'}(\hat{x}^*) - \bar{z}_N, \tag{31}$$

with an estimated variance of

$$\hat{\sigma}^2_{z_{N'}(\hat{x}^*)-\bar{z}_N} = \hat{\sigma}^2_{\hat{z}_{N'}(\hat{x}^*)} + \hat{\sigma}^2_{\bar{z}_N} \tag{32}$$

because the upper and lower bound are independent (Verweij et al., 2003). The SAA procedure of generating M samples and calculating the upper and lower bound is repeated until the optimality gap is smaller than a certain stopping value ϵ. At iteration l, the lower bound $\bar{z}^l_N$ is given by the average objective value of all SAA MIPs solved so far. More precisely,

$$\bar{z}^l_N = \frac{1}{lM}\sum_{m=1}^{lM} z_N^m. \tag{33}$$

4.2 Benders decomposition

When applying the SAA method, we need to solve the SAA MIP in (24) for each generated sample. Therefore, it is crucial to employ an efficient solution algorithm. In this thesis, we will use a solution method similar to the one used in Verweij et al. (2003), which is often referred to as the Benders decomposition method. When this method is applied to stochastic programming problems where the first-stage solution should be an integer variable, the term integer L-shaped method is often used. In what follows, we use the term Benders decomposition. It was introduced as a solution approach for difficult mixed integer problems, not necessarily for stochastic programming problems (Benders, 1962).

The basic idea is to partition the problem into two separate problems. Then, the first problem is solved, while the second problem is being approximated. This approximation consists of constraints that are added to the first problem. This process is iterated multiple times, where in each iteration, a constraint is added to the first problem, improving the approximation of the second problem. In two-stage stochastic programming, this partition arises naturally, since a distinction can be made between the first- and second-stage problem.

In this section, we explain how the procedure can be applied to an SRP for one scenario n of the form

$$z = \min_{x \in X} c^T x + q(\omega^n)^T y \tag{34}$$

$$\text{subject to} \quad Dy \geq h(\omega^n) - T(\omega^n)x, \tag{35}$$

$$y \geq 0, \tag{36}$$

where the notation has the same meaning as before. After that, the explanation is extended to the SAA MIP in (24). The two problems that can be distinguished are the first-stage problem

$$z = \min_{x \in X} c^T x \tag{37}$$

and the second-stage problem

$$Q(x, \xi(\omega^n)) = \min_{y \geq 0} q(\omega^n)^T y \tag{38}$$

$$\text{subject to} \quad Dy \geq h(\omega^n) - T(\omega^n)x. \tag{39}$$

In Benders decomposition, these problems are solved iteratively, where in each iteration, the first stage is solved before the second stage. However, notice that the second stage is not present when solving (37). In order to account for that, a decision variable θ is added to the objective of the first stage, such that it approximates the second-stage objective. We make sure θ approximates the second-stage objective by adding constraints. We will now explain how this can be achieved.

In the second stage, the solution to the first-stage problem is taken as given and we optimize over the variable y. The way we defined θ means that it serves as a lower bound for the objective value $Q(x, \xi(\omega^n))$ of the second-stage problem. Therefore, we aim to find a lower bound for $Q(x, \xi(\omega^n))$ in terms of the first-stage variable x. Notice that the second-stage problem described in (38) and (39) is an LP. Hence, we can find a lower bound by examining the dual of the second-stage problem. It can be denoted as

$$\max_{\pi \geq 0} \pi^T [h(\omega^n) - T(\omega^n)x] \tag{40}$$

$$\text{subject to} \quad D^T \pi \leq q(\omega^n), \tag{41}$$

where π is the vector of dual decision variables. The objective value of the dual problem is lower than or equal to the objective of the primal problem. Furthermore, for non-optimal x-vectors, the objective value of the dual will always be lower than the optimal primal objective value corresponding to the optimal solution x^*. Hence, for every $x \in X$ it holds that $Q(x, \xi(\omega^n)) \geq \pi^T [h(\omega^n) - T(\omega)x]$. We define $e = \pi^T h(\omega^n)$ and $E = \pi^T T(\omega^n)$ so

$$Q(x, \xi(\omega^n)) \geq e - Ex. \tag{42}$$

In each iteration p, we solve the first-stage problem to get a solution x. Then, we solve the dual problem, resulting in $e_p = \pi_p^{*T} h(\omega^n)$ and $E_p = \pi_p^{*T} T(\omega^n)$, and the lower bound on θ, which we call an optimality cut, is added to the first-stage problem. The first-stage problem is then denoted as

$$z = \min_{x \in X} c^T x + \theta \tag{43}$$

$$\text{subject to} \quad \theta \geq e_p - E_p x, \quad p = 1, \ldots, P \tag{44}$$

where constraints (44) are the optimality cuts. An optimality cut θ is added in each iteration, such that the approximation of the second-stage objective improves.

Now, we explain how we apply Benders decomposition to the SAA MIP in (24). For general two-stage SRPs, it can be formulated as

$$z_N = \min_{x \in X} c^T x + \frac{1}{N} \sum_{n=1}^{N} Q(x, \xi(\omega^n)). \tag{45}$$

We approximate the second part of the objective value by introducing a decision variable θ, resulting in

$$z_N = \min_{x \in X} c^T x + \frac{\theta}{N}, \tag{46}$$

where optimality cuts need to be added to improve the approximation θ. To generate the optimality cuts, the second-stage problem for each scenario $n = 1, \ldots, N$ is solved and the optimal solution to the dual problem for a certain scenario n is denoted by π_n^*. The dual objective value for each scenario provides a lower bound on $Q(x, \xi(\omega^n))$ in (45). Therefore, θ should be greater than or equal to the sum of the dual objectives, which gives the optimality cut

$$\theta \geq \sum_{n=1}^{N} \pi_n^{*T} [h(\omega^n) - T(\omega^n) x]. \tag{47}$$

4.3 The branch-and-cut framework

Now, we will give an overview of how we solve the SAA MIP in Equation (24). The pseudocode is provided in Algorithm 1 . We solve the problem with a branch-and-cut algorithm. This means we start a branch-and-bound search with a relaxed problem and add constraints iteratively during the solution process. For the SPPRT, we only relax the integrality constraints (5) and for the TSPRT, we relax both the subtour elimination constraints (11) and the integrality constraints (12). For the TSPRTTW, we also relax both the subtour elimination constraints (16) and the integrality requirements (17). So, the branch-and-bound search is initialized with a linear programming problem. The solution space is represented by a tree, in which each branch represents a subspace of the solution space and each node is a candidate solution. The algorithm explores the branches to find the optimal integer solution. At each node, an upper and lower bound are computed. Branches can be pruned if the lower and upper bound of that branch make it impossible that a better solution will be found than the current best solution. Because we use branch-and-cut, we also add constraints at certain nodes in the tree. If a constraint is added at a given node, the problem is solved again at this node and we check once more whether we should add a constraint.

As described in the previous section, some of the constraints we add iteratively are the optimality cuts. For the TSPRT and TSPRTTW, we also add the subtour elimination constraints iteratively. The number of subtour elimination constraints (11) and (16) is exponential in the number of edges or arcs. For most instances, it would require too much memory to enumerate all of them. Hence, we add them iteratively to the solver during the branch-and-cut algorithm. This can be done in two ways, namely at integer branch-and-bound nodes and at fractional and integer branch-and-bound nodes. In this thesis, we adopt both approaches and compare the results.

The first approach is that we add a subtour elimination constraint at integer solutions. An integer solution has an intuitive interpretation. It is a tour, but it may consist of subtours. In this case, determining whether such subtours are present is straightforward. We can simply start at any solution node, and follow the arcs or edges until we arrive back at the same node. We will return to the same node because of the other constraints in the model. If we visited all nodes, the tour contains no subtours and we do not have to add a constraint. We can add an optimality cut, because we found a feasible solution in the first stage. However, if we only visited a set of nodes $S \subset V$, a subtour is present and we eliminate this subtour by adding the appropriate constraint.

The second approach, which is used by Verweij et al. (2003), is to check for each solution in the branch-and-bound tree, fractional and integer, whether a subtour elimination constraint is being violated. This requires solving several minimum cut problems, for which we use the algorithm of Hao and Olrin (1992). This algorithm is also used by Verweij et al. (2003) and it is explained in Section 4.5.

Algorithm 1: BranchAndCut(`addSubtourConstr`,`addFracSubtourConstr`)

Input: `addSubtourConstr` is a boolean indicating whether we add subtour constraints or not. It has value true for the TSPRT and TSPRTTW and false for the SPPRT. `addFracSubtourConstr` is a boolean indicating whether we are adding subtour constraints at fractional solutions.

```
Initialize the branch-and-bound tree.
Start with relaxed MIP model in current node;
while an optimal solution is not found do
    Solve the current node;
    if the current solution is fractional and addFracSubtourConstr = true then
        Determine whether a subtour constraint is violated;
        if a subtour constraint is violated then
            Add the violated constraint;
        else Continue branching and select next node to explore;
        end if
    else if the current solution is integer then
        if addSubtourConstr, = true then
            Determine whether a subtour constraint is violated;
            if a subtour constraint is violated then
                Add the violated constraint;
            else
                Solve the second-stage problem and add an optimality cut;
            end if
        else
            Solve the second-stage problem and add an optimality cut;
        end if
    else Continue branching and select next node to explore;
    end if
end while
Return the optimal solution.
```

4.4 The optimality cuts for the SPPRT, TSPRT and TSPRTTW

For the SPPRT and TSPRT, the primal second-stage problem is described in Equation (7). The dual of this problem for a given scenario n is given by

$$\max_{\pi \geq 0} \pi[\xi(\omega^n)^T x - \kappa] \tag{48}$$

$$\text{subject to} \quad \pi \leq 1. \tag{49}$$

It requires no derivation to see that the optimal solution is given by $\pi^* = 0$ if $\xi(\omega^n)^T x - \kappa \leq 0$ and $\pi^* = 1$ otherwise. Therefore, the problem does not actually need to be solved, since the optimality cut is easily derived for a given first-stage solution x. Taking this into account and defining $S(x)$ as the subset of scenarios in the sample for a given solution x for which $\xi(\omega^n)^T x - \kappa > 0$, the optimality cut can be written as

$$\theta \geq \sum_{n \in S(x)} [\xi(\omega^n)^T x - \kappa]. \tag{50}$$

On the other hand, for the TSPRTTW, the second-stage problem is not that straightforward. There, we need to solve an LP problem for every scenario $n = 1, \ldots, N$ to determine an optimality cut for a given first-stage solution x. We take the objective of the dual problem of each scenario which gives the following optimality cut:

$$\theta \geq \sum_{n=1}^{N} \Big[\sum_{(i,j) \in A} \pi_{ij}^n [t_{ij}(\omega^n) + M(1 - x_{ij})] + \sum_{i \in V} \pi_i^n [a_i] + \sum_{j \in V} \pi_j^n [-b_j] \Big]. \tag{51}$$

In the dual objective, we sum over the arcs for constraints (19) and we sum over the nodes for constraints (20) and (21).

4.5 Algorithm for finding subtours in fractional solutions

Determining whether a subtour constraint is being violated in a fractional solution is more involved, since a tour is not defined for a non-integer solution. In this case, we have to determine whether the nodes can be divided in two sets, say S and $V \setminus S$, such that

$$\sum_{\{i,j\} \in E : i \in S, j \in V \setminus S} x_{ij} + \sum_{\{j,i\} \in E : i \in S, j \in V \setminus S} x_{ji} < 2. \tag{52}$$

Or, on a directed graph, we want to determine whether there exists a subset of nodes $S \subset V$ such that

$$\sum_{(i,j) \in A : i \in S, j \notin S} x_{ij} < 1. \tag{53}$$

If we interpret the values x_{ij} as capacities for a minimum cut problem, we can define this problem as finding the minimum cut in a network and checking whether its value is smaller than two or smaller than one (Hao & Orlin, 1992). By applying this methodology, we hope to find tours without subtours using less branch-and-bound nodes.

For finding the minimum cut in a network, several algorithms exist. In this thesis, we use the algorithm of Hao and Orlin (1992), which is also used by Verweij et al. (2003). This algorithm finds the minimum cut in a directed graph. Because the TSPRT is defined on an undirected graph, we create two arcs (i, j) and (j, i) for each edge $\{i, j\} \in E$ for the minimum cut problem. The TSPRTTW is already defined on a directed graph. Let s be any node from V and let $n = |V|$. The basic idea of the algorithm is to find the minimum $s - j$ cut and the minimum $j - s$ cut for each $j \neq s$ and to select the minimum of these $2n - 2$ cuts. The Hao and Orlin (1992) algorithm is formulated in such a way that the total running time for solving the $n - 1$ minimum $s - j$ cut problems is comparable to the time to solve one minimum $s - j$ cut problem. The running time is $O(nm \log(\frac{n^2}{m}))$, where m is the number of arcs.

We define $u(S^*, V \setminus S^*)$ as the value of the cut $(S^*, V \setminus S^*)$, defined as the sum of the capacities of all arcs from S to $V \setminus S$. A high-level description is given in Algorithm 2. Note that the sink node t changes in each minimum cut problem.

Algorithm 2: FindMinCut(s)

$S \leftarrow \{s\}$;
BestValue $\leftarrow \infty$;
Cut $\leftarrow \emptyset$;
while $S \neq V$ **do**
  Select a sink $t \in V \setminus S$;
  Determine a minimum $S - t$ cut $(S^*, V \setminus S^*)$;
  $z \leftarrow u(S^*, V \setminus S^*)$;
  if $z <$ BestValue **then**
    Cut $\leftarrow (S^*, V \setminus S^*)$;
    BestValue $\leftarrow z$;
  end if
  Add t to S;
end while

To solve minimum cut problems, the preflow-push algorithm is applied (Hao & Orlin, 1992). We will now provide a summary of this algorithm. The interested reader is referred to the Appendix which contains a more detailed explanation and all the pseudocode. Recall that the minimum cut problem is solved on a directed graph $G = (V, A)$, where the capacity for each arc arc $(i, j) \in A$ is given by x_{ij}. A preflow is defined as a vector $y \in \mathbb{R}^{|A|}$ which satisfies a number of conditions. For each node $i \in V$ we define the excess of node i as

$$e(i) = \sum_{j \in V:(j,i) \in A} y_{ji} - \sum_{j \in V:(i,j) \in A} y_{ij} \tag{54}$$

Then, y is a preflow if the flow y_{ij} on each arc satisfies the capacity limit and if the flow into every node i exceeds the outflow (except for the source node). In other words, y is a preflow if $0 \leq y_{ij} \leq x_{ij}$ for each $(i, j) \in A$ and $e(i) \geq 0$ for each $i \in V$. For a given preflow y, we define the residual capacity r_{ij} on each arc $(i, j) \in A$ as the maximum additional flow that can be sent from node i to node j via the arc (i, j). Obviously, the flow from i to j can be increased if $y_{ij} < x_{ij}$. In addition, if $y_{ji} > 0$, we can cancel the flow from j to i to enlarge the net flow from i to j. Therefore, we define $r_{ij} = x_{ij} - y_{ij} + y_{ji}$.

The algorithm starts by sending as much flow as possible away from the source node. In other words, all arcs leaving the source are saturated. This means the excess of other nodes becomes strictly positive and we can send flow from these nodes. The flow will be sent from nodes with positive excess via arcs with residual capacity. To make sure the flow goes to the sink, so-called distance labels are used. For a detailed explanation, we refer to Hao and Orlin (1992). We continue this procedure until no more flow can be pushed towards the sink. At this point, we find a maximum flow, and thus a minimum cut. After a problem is solved, a new sink node should be selected. By choosing the next sink node carefully and implementing the methods of the preflow-push algorithm appropriately, the computation time of solving $n - 1$ minimum cut problems is reduced to the computation time of solving one problem.

5 Problem instances

In this section, we give a description of how we generate the problem instances. Unfortunately, the exact problem instances used by Verweij et al. (2003) are unavailable. Therefore, we tried to generate them ourselves, using the description provided in their paper. We use the TSP library (Reinelt, 1991) which contains TSP instances. Each instance consists of a set of cities and a distance matrix, specifying the distance between any pair of cities. Based on the TSP instance, we generate a graph G accompanied by a cost vector c and a probability distribution over the travel times of the arcs or edges in G. For the TSPRT and TSPRTTW, we also generate deadlines.

For the SPPRT, we generate a directed graph $G = (V, A)$, where each node $v_i \in V$ corresponds to a certain city i from the TSP library instance. The distances between any pair of cities (i, j) is defined as d_{ij}. We say that nodes v_i and v_j have a direct connection if there exist arcs (i, j) and (j, i). In order to generate the arcs, we iterate over the nodes in V. In iteration i, we make a

Table 1: Dimensions of the graphs generated from the TSP library instances

		$\|A\|$		$\|E\|$				$\|A\|$		$\|E\|$	
Name	$\|V\|$	Our	Verweij	Our	Verweij	Name	$\|V\|$	Our	Verweij	Our	Verweij
burma14	14	182	172	91	86	rat99	99	1980	1872	990	936
ulysses16	16	230	212	115	106	kroA100	100	2000	1892	1000	946
ulysses22	22	400	332	200	166	kroB100	100	2000	1892	1000	946
eil51	51	1020	912	510	456	kroC100	100	2000	1892	1000	946
berlin52	52	1040	932	520	466	kroD100	100	2000	1892	1000	946
st70	70	1400	1292	700	646	kroE100	100	2000	1892	1000	946
eil76	76	1520	1412	760	706	rd100	100	2000	1892	1000	946
pr76	76	1520	1412	760	706	eil101	101	2020	1912	1010	956
gr96	96	1920	1812	960	906						

This table shows the dimensions of the graphs that are generated using the TSP library instances. For each instance, the table displays the number of nodes, arcs and edges. In addition, the graphs are compared with those in Verweij et al. (2003).

direct connection between v_i and the δ closest nodes with which it has no direct connection yet. Therefore, we will add 2δ arcs for each node, unless it already has a direct connection with more than $|V| - 1 - \delta$ nodes. In that case, we connect this node to the remaining nodes with which it has no direct connection yet, meaning we will add less arcs. The cost of each arc is added to c with value $c_{(v_i,v_j)} = d_{ij}$. To select a source and a sink, we find the pair of nodes (v_s, v_t) for which the minimum number of arcs in a $v_s - v_t$ path is maximal. If multiple such pairs exist, we choose a pair randomly, with each pair having equal probability of being chosen. Next, we determine the distribution of travel times over the arcs in such a way that the use of short arcs is made unattractive. To be more precise, the traveling time on short arcs can become very large with a certain probability p while the traveling time on long arcs is kept deterministic. If we define $\bar{c}$ as the median of the arc lengths and $\xi_a(\omega)$ as the random travel time corresponding to arc $a \in A$, we set

$$P\{\xi_a(\omega) = Fc_a\} = \begin{cases} p, & \text{if } c_a \leq \bar{c} \\ 0, & \text{if } c_a > \bar{c} \end{cases} \quad \text{and} \quad P\{\xi_a(\omega) = c_a\} = \begin{cases} 1-p, & \text{if } c_a \leq \bar{c} \\ 1, & \text{if } c_a > \bar{c} \end{cases} \tag{55}$$

for some parameter $F > 1$. Finally, we solve a SPP for the source, sink and cost vector c defined above and we take the expected travel time on this path. That is the value for κ, the deadline.

For the TSPRT, we create a graph $G = (V, E)$ in the same way as for the SPPRT, except that a direct connection between two nodes now exists if an undirected edge connects them. The distribution over the travel times is determined in the same way. The deadline is determined by solving a TSP with respect to the cost vector c. The expected length of an optimal tour is the value for κ.

The following parameter values were used for generating the instances of both the SPPRT and the TSPRT: $\delta = 10$ and $p = 0.1$. For the travel time distribution, $F = 10$ is used for the SPPRT and $F = 20$ for the TSPRT. The generated instances are summarized in Table 1. The number of arcs and edges in the graphs generated by Verweij et al. (2003) are denoted by $|A|_{\text{V}}$ and $|E|_{\text{V}}$, respectively.

Notice that our resulting graphs differ from the ones generated in Verweij et al. (2003). We generate more edges and arcs, but it is unclear to us where this comes from as the methodology of Verweij et al. (2003) has been replicated. For the larger instances, the difference in the number of edges is large, with a size of 54. Since this increases the number of possible routes and tours and thus the sample space, it may have a downward effect on the objective value.

For generating the TSPRTTW instances, we use the same approach as for the TSPRT, with some minor dissimilarities. The first important difference is that we need a directed graph $G = (V, A)$ instead of an undirected graph. Therefore, we connect nodes v_i and v_j by adding the arcs (i, j) and (j, i) like for the SPPRT instances. Secondly, we need to select a depot, from which the tour starts. For every instance, we take the first node as mentioned in the TSP library instance, as the depot. Additionally, we create an extra node for the depot, which we label v_{n+1} Also, we need to generate a time window for each node. For the last node in the route, the depot v_{n+1}, the latest arrival time b_{n+1} is taken as the value of the deadline κ derived for the TSPRT. a_{n+1}, a_0 and b_0 are set equal to zero. For the remaining a_i and b_i, we first compute the average arrival times at each node from the best solution we found for the TSPRT. Let the average arrival time at node i be denoted by t_i. The time window for this node is then set to $[t_i - \frac{\kappa}{2n}\alpha, t_i + \frac{\kappa}{2n}\alpha]$, where

n is the number of nodes and α is a factor which determines the width of the intervals. Hence, all windows have equal length. The TSPRTTW is evaluated for $\alpha \in \{1, 3, 1000\}$. The value $\alpha = 1000$ corresponds to the case where the TSPRTTW has no deadlines at intermediate nodes, meaning it becomes identical to the TSPRT.

6 Computational results

In this section, we report and discuss the computational results. First, we discuss the optimization software. Next, we show the results for the SPPRT and TSPRT, where we do not add subtour constraints at non-integer solutions of the TSPRT yet. Then, we report the results for the TSPRTTW and we compare the solution approach for the TSPRTTW with the approach for the TSPRT. Next, we display the results when the subtour constraints are also added at non-integer solutions. Finally, we run the SAA for multiple iterations and investigate the convergence of the lower and upper bound. All computations are performed on a computer with 4GB RAM and an Intel(R) Core(TM) i7-4700MQ CPU with 2.40 GHz. Unless stated otherwise, the number of nodes in the branch-and-bound tree is limited at 10000, for each SAA MIP.

6.1 CPLEX implementation

The solution methods of Section 4 have been implemented in Java using the commerical MIP solver CPLEX, version 12.6.3. For adding cuts in the branch-and-bound tree at nodes where an integer solution is found, we use the `LazyConstraintCallback` provided by CPLEX. We use the `UserCutCallback` for adding the subtour constraints when a non-integer solution is found. Note that CPLEX defines user cuts as constraints that do not cut off feasible integer solutions, but only strengthen the formulation. The reason is that CPLEX does a variety of problem reductions, both before and during the branch-and-bound search, to tighten the feasible region of the problem. Sometimes, solutions to dual problems are involved. If some constraints that restrict the integer solution space are left out, the dual problem lacks the variables corresponding to these constraints. This can lead to incorrect results. However, in our case the subtour constraints do cut off (otherwise feasible) integer solutions. Therefore, if the `UserCutCallback` is enabled, we also use the `LazyConstraintCallback` described previously, because when this callback is called, the reductions are disabled by the solver. Furthermore, because CPLEX does not expect the user cut to make an incumbent solution infeasible, the `UserCutCallback` will not be called at nodes where an integer solution is found. Because the integer solutions may be infeasible as well, the `LazyConstraintCallback` is needed too.

6.2 Computational results for the SPPRT and TSPRT

Table 2 displays the results for the SPPRT and TSPRT for a sample size of $N = 1,000$. Furthermore, the Mean Value Problem (MVP) is solved and the results are shown. In this approach, each random variable is replaced by its mean and the resulting deterministic problem is solved. The gap of the MVP objective value is obtained by taking the difference of the MVP objective and the objective resulting from the SAA. In most cases, the Mean Value approach clearly underperforms when compared against the SAA. The objective value from the MVP is never lower and the gap is often large. For the instances burma14 and ulysses22 of the SPPRT, the SAA and MVP produce the same solution. In both cases, the solution of the SAA only contains arcs with one possible travel time, because the standard deviation of the gap between the SAA upper and lower bound is equal to zero. Therefore, it is not unreasonable that this path is also the shortest path with respect to expected travel times. On the other hand, there are also instances for which the SAA solution does not consist of random arcs, but the MVP gap is non-zero. This is not a surprising observation if we recall how the randomness on the arcs was generated. For the values of $F = 10$ and $p = 0.1$, the expected travel time on arcs, defined in Equation (55), is not much higher than the deterministic travel time. Hence, if the optimal path found by the SAA is non-random, there may well exist a shorter path with respect to the expected travel time, found by the MVP.

Another observation from the table is that the objective values for the TSPRT are much larger than those obtained for the SPPRT. The solution to the TSPRT is a cycle, which contains two different paths between any pair of vertices (v_i, v_j). Therefore, the length of the path obtained in the SPPRT can never exceed the length of a cycle.

Table 2: Solution values and absolute gaps with lower bound for the SPPRT and TSPRT

	SPPRT						TSPRT					
	SAA			MVP			SAA			MVP		
Instance	$\hat{z}_{10^5}(\hat{x}^*)$	Gap	$\hat{\sigma}_{\text{Gap}}$	$\hat{z}_{10^5}(\bar{x})$	Gap	$\hat{\sigma}_{\text{Gap}}$	$\hat{z}_{10^5}(\hat{x}^*)$	Gap	$\hat{\sigma}_{\text{Gap}}$	$\hat{z}_{10^5}(\bar{x})$	Gap	$\hat{\sigma}_{\text{Gap}}$
burma14	218.0	0.0	0.00	218.0	0.0	0.69	1688.7	1.9	4.69	2103.3	414.6	7.92
ulysses16	606.0	4.5	3.66	876.8	270.9	3.74	3455.2	-2.3	9.63	6752.6	297.4	13.16
ulysses22	551.0	0.0	0.00	551.0	0.0	1.74	3522.7	2.2	7.63	3745.5	222.8	12.89
eil51	61.6	0.1	0.33	73.5	12.0	0.47	553.8	5.0	1.94	570.9	17.1	2.28
berlin52	1670.1	0.0	0.00	1835.2	165.1	6.08	9049.4	40.7	17.92	9368.6	319.4	31.06
st70	94.6	0.1	0.31	124.3	29.7	0.54	808.1	7.1	4.28	884.7	76.7	2.98
eil76	72.9	0.0	0.00	82.8	9.9	0.28	649.4	-1.8	2.66	685.9	36.5	2.28
pr76	22433.0	0.0	0.00	25798.7	3365.7	85.99	125796.7	1057.8	468.26	132752.2	6955.5	237.77
gr96	3073	0	0.00	3879.0	806.0	13.38	23317.3	-472.5	206.99	23885.7	568.4	108.64
rat99	211.8	0.0	0.00	234.4	22.6	0.77	1486.7	0.4	5.98	1511.7	25.0	6.96
kroA100	3648.9	0.0	0.00	4317.9	669.0	14.79	25419.1	-205.2	183.7	26565.3	1146.2	122.42
kroB100	3982.4	3.2	12.70	4440.3	458.0	27.36	25923.6	-11780.0	7144.3	27718.8	1795.2	124.86
kroC100	4022.3	0.0	0.00	4833.4	811.1	16.15	24259.3	-283.1	153.79	26201.5	1942.2	117.28
kroD100	3486.0	0.2	11.03	4766.0	1280.0	27.60	25596.3	50.5	113.86	26758.4	1162.1	121.18
kroE100	3929.6	27.3	12.36	4337.1	407.5	26.65	24963.6	-1471.4	303.20	27763.4	2799.7	122.21
rd100	1245.0	0.0	0.00	1404.2	159.2	4.69	9686.3	-498.3	223.76	9825.5	139.2	45.19
eil101	72.0	-0.0	0.23	91.3	19.3	0.54	757.0	-4.8	3.68	786.8	29.8	3.56

This table shows the solution values, estimated optimality gaps and the corresponding standard deviation for a sample size of $N = 1000$. The Mean Value Problem (MVP) is the deterministic optimization problem where all random variables are replaced by their means. The gap of the MVP is calculated with respect to the optimal solution of the SAA approach.

More interestingly, we observe that the objective values for the SPPRT differ strongly from the results in Verweij et al. (2003) for some problem instances. For example, the objective value for the berlin52 instance is 839.7 in Verweij et al. (2003) whereas we obtain a value of 1,670.1. One possible explanation is that our graphs differ from the ones generated in Verweij et al. (2003). Another explanation for the difference is the manner in which the source and the sink are chosen. We first determine the pairs of vertices for which the shortest path contains the maximum number of arcs. Next, we choose a pair randomly. Thus, the length of the resulting path can vary strongly, because the length of the arcs is not taken into account when setting the source and sink. In addition, the number of random arcs that have to be used can differ strongly between different pairs of vertices. This also has an effect on the objective value, through the recourse function. For other instances, the SPPRT objective values are of the same order of magnitude across the instances as in Verweij et al. (2003), which gives some confidence that the results are correct.

Table 3: Detailed results for the SPPRT for eil51

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
eil51									
$\bar{z}_N$	61.1	61.5	61.4	61.3	61.4	61.4	61.6	61.3	61.5
$\hat{\sigma}_{\bar{z}_N}$	0.17	0.21	0.18	0.13	0.12	0.11	0.15	0.09	0.11
$\hat{z}_{10^5}(\hat{x}^*)$	61.5	61.6	61.6	61.6	61.6	61.5	61.6	61.6	61.6
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
Nodes	12.9	18.5	16.6	14.6	16.8	16.3	18.0	17.2	19.2
$\hat{\sigma}_{\text{Nodes}}$	5.79	8.15	4.61	4.20	6.14	3.29	4.80	4.24	3.89
CPU	0.8	1.3	1.7	2.2	3.0	3.8	4.8	6.0	7.0
$\hat{\sigma}_{\text{CPU}}$	0.08	0.16	0.25	0.17	0.12	0.21	0.44	0.75	0.69
Opt. Cuts	16.4	18.0	17.3	17.0	17.8	17.6	18.4	18.1	18.7
$\hat{\sigma}_{\text{Opt. Cuts}}$	2.11	2.14	2.41	1.41	1.08	1.11	1.11	0.70	0.90
Total CPU	8s	13s	17s	22s	30s	38s	48s	1m	1m11s

This table gives a detailed overview of the obtained results for the instance eil51. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average number of nodes used, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the final row, the total computation time is shown.

Table 4: Detailed results for the TSPRT for eil51

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
eil51									
$\bar{z}_N$	538.1	546.0	543.6	541.5	545.7	547.1	547.6	547.7	548.8
$\bar{\sigma}_{\bar{z}_N}$	4.58	2.80	1.93	1.77	1.97	1.56	2.26	2.46	1.53
$\hat{z}_{10^5}(\hat{x}^*)$	555.9	550.1	558.5	550.0	553.3	548.9	554.4	549.6	553.8
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	1.03	1.16	1.17	1.25	1.22	1.29	1.31	1.21	1.20
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	39.3	59.1	74.9	92.0	137.4	166.4	217.5	249.7	294.1
$\hat{\sigma}_{\text{CPU}}$	7.43	12.51	14.36	10.06	26.04	19.46	37.63	24.11	23.58
Opt. Cuts	320.6	342.0	358.4	347.1	375.4	362.4	359.7	366.7	334.8
$\hat{\sigma}_{\text{Opt. Cuts}}$	46.84	69.77	57.14	42.94	64.61	41.34	51.31	32.12	25.58
Total CPU	6m17s	9m55s	12m33s	15m23s	22m57s	27m47s	36m18s	41m39s	49m4s

This table gives a detailed overview of the obtained results for the instance eil51. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average number of nodes used, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the final row, the total computation time is shown.

Table 5: Detailed results for the SPPRT for st70

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
st70									
$\bar{z}_N$	94.7	94.5	94.9	94.4	94.6	94.6	94.8	94.7	94.6
$\bar{\sigma}_{\bar{z}_N}$	0.27	0.18	0.21	0.09	0.15	0.10	0.12	0.19	0.09
$\hat{z}_{10^5}(\hat{x}^*)$	94.6	94.6	94.6	94.6	94.6	94.6	94.6	94.6	94.6
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30	0.30
Nodes	12.0	7.9	7.7	6.6	14.0	7.4	11.6	8.5	8.9
$\hat{\sigma}_{\text{Nodes}}$	6.96	2.98	4.54	3.98	7.51	5.04	8.38	6.22	4.61
CPU	0.7	1.4	2.4	3.2	4.6	5.3	6.1	7.8	8.4
$\hat{\sigma}_{\text{CPU}}$	0.16	0.29	0.62	0.56	1.15	0.61	1.46	1.07	1.50
Opt. Cuts	9.8	10.9	12.1	11.3	12.1	11.6	11.1	11.5	10.5
$\hat{\sigma}_{\text{Opt. Cuts}}$	2.44	2.30	3.18	2.05	3.36	1.69	2.43	1.36	1.75
Total CPU	8s	14s	24s	33s	46s	54s	1m1s	1m18s	1m24s

This table gives a detailed overview of the obtained results for the instance st70. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average number of nodes used, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the final row, the total computation time is shown.

Concerning the TSPRT, the objective values are fairly similar to those reported in Verweij et al. (2003). This is expected, since all nodes have to be visited. Because the scenarios are generated randomly, the objective values can differ slightly. For the sample size of 1,000 used for these results, this effect should be small. More importantly, the graphs we generated contain more edges than those in Verweij et al. (2003) although we adopted their methodology in this thesis. This means that more tours are possible and hence we expect the objective value to be lower here. This is the case for many instances. However, if there are more edges, this also means that more edges are made random because all edges shorter than the median length are made stochastic. Additionally, the different approach for adding subtour cuts can cause the results to be dissimilar. We will discuss the effect of the different strategy for adding subtour cuts in more detail later this section, when discussing the negative gaps. Moreover, we also implement the approach of Verweij et al. (2003) for adding the subtour cuts and compare the results in Section 6.5.

For the instances burma14, ulysses16, ulysses22 and gr96, the objective values for the TSPRT diverge most clearly. It should be mentioned that these problem instances have an atypical distance function. Distances are measured as geographical distances, which we calculated using the distance function described by Reinelt (1991). However, we suspect that we used a different distance function than Verweij et al. (2003) and hence we will not discuss the results regarding those instances in more detail.

Surprisingly, we find a negative gap between the upper and lower bound for many instances of the TSPRT. Both the upper and lower bound are random variables, implying that negative gaps may occur. Verweij et al. (2003) argue that the size of the negative gap should not exceed two times the standard deviation. The absolute value of most negative gaps is smaller than twice the standard deviation, which shows that these results are not unexpected. However, for kroE100 and rd100, the negative gap exceeds the allowed size. An explanation for this anomaly can be found in the different approach for adding the subtour cuts. We only add subtour cuts in a branch-and-bound node where an integer solution is found, whereas in Verweij et al. (2003) such a cut is added at every node in the branch-and-bound process. Applying our methodology, the solver may need more nodes to find a feasible tour. Since the number of nodes is limited at 10,000, less tours will be found and we will often find a bad tour. In some of the $M = 10$ samples, we may find a good tour. In other words, we expect that the optimality gap of some of the M SAA MIPs has not become small yet when the node limit is reached.

This difference in the quality of the solutions to the SAA MIPs is seen clearly for the instance kroB100, for which the detailed results are shown in Table 7. Especially for $N = 1,000$ we see that the number of optimality cuts is relatively low (for example compared with Tables 4 and 6) and varies strongly. Recall that a cut is added every time a feasible tour is found. This means

Table 6: Detailed results for the TSPRT for st70

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
st70									
$\bar{z}_N$	784.6	790.1	790.0	799.9	797.1	801.5	800.4	799.5	801.0
$\bar{\sigma}_{\bar{z}_N}$	4.34	4.22	3.98	6.65	2.77	4.49	4.11	2.27	3.38
$\hat{z}_{10^5}(\hat{x}^*)$	788.3	795.6	827.1	813.5	814.1	795.6	828.7	796.6	808.1
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	1.54	1.79	1.65	1.62	1.82	2.01	1.97	2.11	2.63
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	70.8	85.8	108.9	138.1	197.6	201.5	249.1	324.0	380.2
$\hat{\sigma}_{\text{CPU}}$	8.20	12.68	13.58	22.51	32.01	34.16	39.16	23.61	44.61
Opt. Cuts	442.5	439.7	432.7	437.5	478.5	463.3	477.4	480.8	459.7
$\hat{\sigma}_{\text{Opt. Cuts}}$	62.31	57.57	48.36	73.15	46.19	71.15	89.71	40.75	50.36
Total CPU	11m52s	14m22s	18m13s	23m5s	32m59s	33m37s	41m35s	54m4s	1h3m26s

This table gives a detailed overview of the obtained results for the instance st70. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average number of nodes used, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the final row, the total computation time is shown.

Table 7: Detailed results for the TSPRT for the instance kroB100

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
kroB100									
$\bar{z}_N$	30959.4	31200.9	39710.3	32133.3	36841.9	50467.4	28141.4	27318.6	37703.6
$\bar{\sigma}_{\bar{z}_N}$	5075.47	4298.27	7329.03	4824.85	6662.31	8470.65	648.57	453.53	7143.78
$\hat{z}_{10^5}(\hat{x}^*)$	25928.3	26687.9	25846.7	26863.7	25922.7	51126.6	27268.7	26778.6	25923.6
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	84.10	86.43	83.62	83.66	83.75	83.36	84.09	84.83	84.54
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	36s	42s	50s	1m16s	1m31s	1m18s	2m1s	2m49s	3m32s
$\hat{\sigma}_{\text{CPU}}$	5.96	4.48	14.60	13.67	28.58	31.09	41.47	38.44	61.41
Opt. Cuts	100.8	88.0	70.4	97.6	93.9	51.7	78.3	91.0	103.7
$\hat{\sigma}_{\text{Opt. Cuts}}$	35.23	42.18	47.84	43.56	56.43	46.90	45.41	40.34	58.46
Total CPU	6m1s	6m57s	8m15s	12m39s	15m9s	13m2s	20m7s	28m13s	35m33s

This table shows a detailed overview of the results obtained for the kroB100 instance for the TSPRT. Lower bounds, upper bounds, number of nodes, computation time and computation time are shown with their standard deviation. In addition, the total CPU time is shown. A node limit of 10,000 is used for the SAA MIP in each sample.

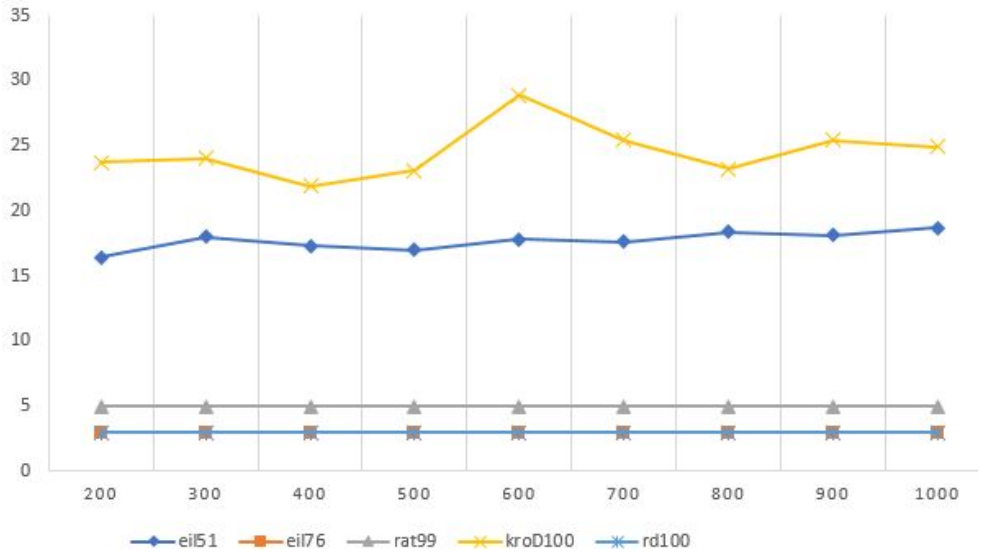
that in some cases, we find many tours whereas in other cases, we find only a few. A few very bad tours are given as a solution, giving a huge variance in the lower and upper bound. For the lower bound, we do take these bad solutions into account because we take the average of the M lower bounds. However, for the upper bound we take the best one from the M solutions, leaving the bad tours out of consideration. This is how the negative gap arises. For example, for $N = 1000$, we find a very poor solution for two SAA MIPs. For these two cases, the objective values are 78431 and 82513, where 28 and 8 optimality cuts were added respectively. These values are far from the average and thus, they cause the lower bound to be very high, whereas for the upper bound, these two solutions are ignored.

From Table 7, it can also be seen that the number of optimality cuts does not increase with the sample size. This is an important result in the paper by Verweij et al. (2003). It extends to our analysis, which is shown more clearly in Figures 1 and 2. For both the SPPRT and TSPRT, the average number of optimality cuts is fairly constant across the sample size. Note that the constant number of cuts for the SPPRT arises when the shortest path only includes deterministic travel times. At each integer solution, an optimality cut is added. The solver finds the same integer solutions in the branch-and-bound process for differing sample size because the problem remains approximately the same.

We also replicate another result from Verweij et al. (2003), namely that the computation time increases linearly with the sample size N. Figure 3 shows the average computation time for the TSPRT for instances of varying size. The slope of the CPU time is fairly constant for all instances. Note that the computation times for the larger instances are lower because less optimality cuts are added there.

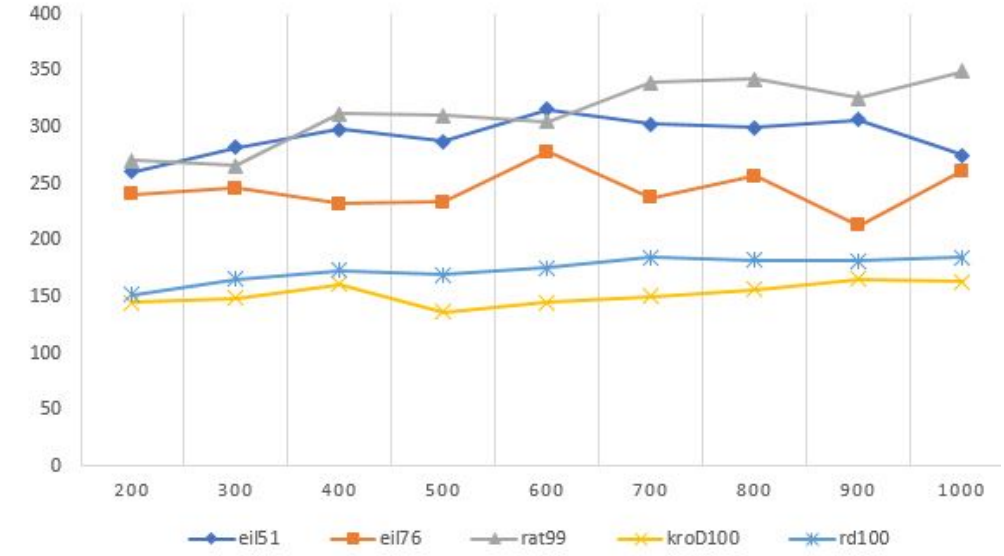
Finally, Tables 3, 4, 5 and 6 give a detailed description of the results for the SPPRT and TSPRT corresponding to the instances eil51 and st70. The results are very similar to those in Verweij et al. (2003). However, the computation times are much lower here. The decrease in CPU time is around 70 percent. This is due to the increased computing power and advancements in the MIP solver. For a more detailed overview of the results for other instances, we refer to the Appendix.

Figure 1: Average number of optimality cuts for the SPPRT



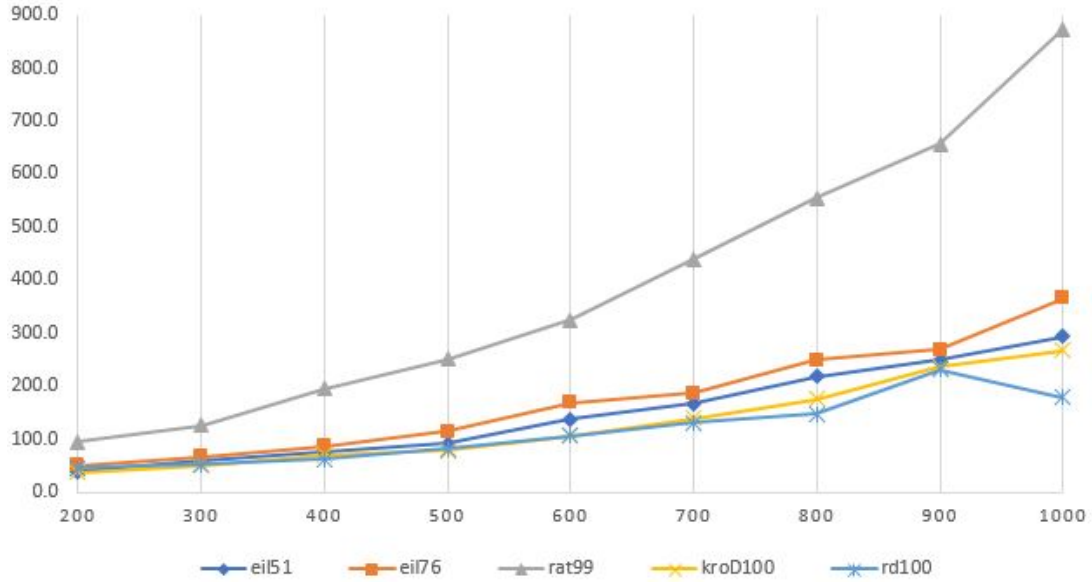
The average number of optimality cuts added in the SPPRT for instances of varying size.

Figure 2: Average number of optimality cuts for the TSPRT



The average number of optimality cuts added in the TSPRT for instances of varying size.

Figure 3: Average computation time for the TSPRT



The average computation time for the TSPRT for instances of varying size.

6.3 Computational results for the TSPRTTW

The results for the TSPRTTW for instances eil51, st70, eil76, rat99, kroC100, kroD100 and rd100 are shown in Table 8. The problem instances have been solved for a sample size of $N = 200$ and for each sample, the number of nodes in the branch-and-bound tree for the SAA MIP is limited at 10,000. The objective value, defined in Equations (13) and (18), can be split up in three parts: the deterministic routing cost, the violation of the final deadline and the violation of deadlines at intermediate nodes. The first two values and the total are shown in the table. Furthermore, recall from Section 5 that the time windows were derived using the average arrival times of a solution to the TSPRT. We also report the objective value in the TSPRTTW setting using this solution ($\hat{x}_T^*$). We expect it to perform well, because the time windows are set around the average arrival time of this route at the nodes. Therefore, we use it as a benchmark. This objective is also broken down into the three components mentioned before.

Not surprisingly, we observe from the table that the objective values are much higher than the results for the TSPRT shown in Table 2. The deterministic routing cost is of the same order of magnitude as the objective of the TSPRT. However, when we include the violation of the final deadline, the costs are always higher. This can be explained by the fact that avoiding the violation of the final deadline is not the only objective when time windows are included. Several other deadlines are at hand, meaning that it is sometimes more optimal to violate the final deadline more, while incurring less violations at intermediate nodes. Another important reason for the high violation at the depot, is that the vehicle has to wait till a_i if it arrives at node i before that time. This occurs several times in the solutions, which causes the delay at later nodes to increase. The remaining costs are due to violating the deadlines of intermediate nodes on the tour. These costs are very high, which means that the solutions we found may well be suboptimal due to the node limit of 10,000. This is confirmed when we take into account the objective value corresponding to the solution of the TSPRT, shown in column seven of Table 8. These objective values are much lower for all instances, meaning that the solution we found is far from optimal. Hence, the solution method is not capable of finding a good solution to the SAA MIPs in 10,000 nodes. This was also discussed in Section 6.2, where we argued that the approach for adding subtour cuts causes the solver to run out of nodes quickly, without finding a good solution.

Note that we could have found the solution $\hat{x}_T^*$ by visiting the time windows in the chronological order. However, this is not a good solution in general. Because of the way we generated the time windows, the chronological order of the time windows corresponds with a short route in terms of traveling time since it is based on a good tour for the TSPRT. In general, visiting the nodes in the chronological order of the time windows may result in a very high objective value because it may have a very high traveling time. In such cases, the solution approach we propose may well perform

better.

Table 8: Solution values and gaps with the lower bound for the TSPRTTW

Instance	$\hat{z}_{10^5}(\hat{x}^*)$	$\hat{z}_{10^5}(\hat{x}^*)_{\text{Det}}$	$\hat{z}_{10^5}(\hat{x}^*)_{\text{Viol}}$	Gap	$\hat{\sigma}_{\text{Gap}}$	$\hat{z}_{10^5}(\hat{x}^*_T)$	$\hat{z}_{10^5}(\hat{x}^*_T)_{\text{Det}}$	$\hat{z}_{10^5}(\hat{x}^*_T)_{\text{Viol}}$
eil51	20630.2	496.1	458.3	-3826.1	1241.2	7528.7	460.5	154.8
st70	27124.0	853.1	401.1	-39333.6	9030.1	13736.8	706.2	147.5
eil76	35506.4	571.2	522.3	-17176.9	3146.2	11718.0	562.6	166.0
rat99	188551.5	1618.1	2604.2	-18295.8	5908.4	29942.3	1276.4	387.7
kroC100	2166552.7	36498.2	29188.6	-190833.5	29791.2	538808.2	21574.4	5598.4
kroD100	3773921.5	31773.8	43744.8	-1460006.0	334070.8	491709.5	22636.8	4800.3
rd100	1016216.6	12915.4	11862.9	-706775.4	139440.7	168344.3	8974.9	1752.9

This table shows the objective values of the solutions obtained for the TSPRTTW for problem instances of varying size. The number of nodes in the branch-and-bound tree was set at 10,000 and a sample size of $N = 200$ was used. The third and fourth column show the part of the costs that are due to the deterministic routing costs and the violation of the final deadline, respectively. The same is shown in the last two columns for the solution $\hat{x}^*_T$, which denotes the solution obtained for the TSPRT on which the time windows are based.

The reason we discussed the node limit issue in Section 6.2 was that some large negative gaps were found between the upper and lower bounds. Although it was only present for the larger instances in that section, we find a large negative gap for every instance of the TSPRTTW. When we measure the gap in terms of its standard deviation, the size seems to increase when the problem size increases. For example, for the kroC100 instance, the size of the gap is approximately six standard deviations. As explained previously, the upper and lower bound are random variables and a negative gap up to two standard deviations would not be unexpected. However, the absolute value of the gap is always larger here. Because the problem seems to be more severe for the large instances, we again argue that the node limit is to blame. The SAA MIPs, arising in each sample, are difficult to solve to optimality. A relatively good solution is found only rarely, and therefore the variation in the lower bound is not that high whereas the best of the $M = 10$ solutions is much lower than the average solution. So only in one or two out of the $M = 10$ samples, a good route is found. For the other samples, we do not find a good tour yet, but we expect to find it when more nodes can be used in the branch-and-bound process.

Table 9: Detailed solutions for the TSPRTTW for eil51 and st70

Instance	$\bar{z}_N$	$\hat{\sigma}_{\bar{z}_N}$	$\hat{z}_{10^5}(\hat{x}^*)$	$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	CPU	$\hat{\sigma}_{\text{CPU}}$	Opt. Cuts	$\hat{\sigma}_{\text{Opt. Cuts}}$	Total CPU
eil51	24456.3	1239.25	20630.2	70.03	25m27s	248.47	553.5	82.90	4h14m41s
eil51 ($N = 500$)	21344.9	457.61	17804.0	59.65	1h10m43s	666.09	567.5	65.42	11h48m9s
eil51 ($\alpha = 3$)	20544.6	1302.87	13538.5	46.04	23m25s	259.43	576.3	99.68	3h54m34s
st70	66457.6	9029.53	27124.0	100.45	19m35s	148.19	293.7	39.25	3h16m7s

This table gives a detailed overview of the obtained results for two instances: eil51 and st70. The sample size we used is $N = 200$, except for the results in the third row. In the fourth row, $\alpha = 3$ indicates that the width of the intervals is three times higher for this instance (see Section 5). If not indicated explicitly, $\alpha = 1$. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the last column, the total computation time is shown.

Table 10: Detailed solutions for the TSPRTTW for eil51 with a node limit of 30,000

Instance	$\bar{z}_N$	$\hat{\sigma}_{\bar{z}_N}$	MIP Gap	$\hat{\sigma}_{\text{MIP Gap}}$	$\hat{z}_{10^5}(\hat{x}^*)$	$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	CPU	$\hat{\sigma}_{\text{CPU}}$	Opt. Cuts	$\hat{\sigma}_{\text{Opt. Cuts}}$	Total CPU
eil51	20256.0	937.09	98%	0.00	14914.2	50.57	7285.4	744.42	2450.3	234.46	20h9m59s

This table gives a detailed overview of the obtained results for the instance eil51 where the node limit is increased to 30,000. The sample size we used is $N = 200$. The table shows the average lower bound, the upper bound, the average optimality gap of the SAA MIPs, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the last column, the total computation time is shown.

For eil51 and st70, a more detailed description of the results is given in Table 9. The variation in the number of optimality cuts is high. For example, for the eil51 instance, the minimum number of cuts added is 415 and the maximum is 687. This supports our argument that for some samples, the solution found is much better than other solutions. If a lot of cuts are added, it means that many feasible tours were found in the branch-and-bound process. Most likely, a better tour is found than when the number of feasible tours found is rather small. This implies that the SAA MIP in some samples is solved closer to optimality than in most other samples. To examine this in more detail, we solved the same problem with a node limit of 30,000 instead of 10,000 of eil51. The results in Table 10 indicate that the limit is still far too low. It seems that the TSPRTTW

instances are very difficult to solve to optimality with the applied methodology. The average SAA MIP gap (98%) is high, although the objective value is lower than the one for the lower node limit in Table 9. Furthermore, we find a strong correlation (0.98) between the objective of the SAA MIP (i.e. the lower bound) and the gap of the SAA MIP. So the variation in the lower bounds is not only due to the difference in the samples, but also due to the difference in solution quality.

We expect the variation in the samples to reduce if the sample size is increased. This seems to hold, since we can see in Table 9 that for a sample size of $N = 500$, the variation in the lower bound is smaller than the one for $N = 200$. We also observe that the objective value decreases if the width of the time windows increases. In the fourth row of Table 9, we find that the costs decrease by approximately 35%. Therefore, increasing the window width has a large effect on the costs.

Another important result in the table is the strong increase in computation time compared to the SPPRT and TSPRT. For example, for eil51, the total computation time is approximately 35 times as large as for the TSPRT, if we use a sample size of $N = 200$. This suggests that it requires much more computational effort to generate the optimality cuts for the TSPRTTW than those for the TSPRT. We investigate this in more detail in the next section.

6.4 Comparison of the methodology for TSPRT and TSPRTTW

The TSPRTTW is a more general problem than the TSPRT. If we set the time windows $[a_i, b_i]$ very wide ($\alpha = 1000$) for all nodes except for the depot, we are left with the TSPRT. Therefore, we can compare the solution approach for the TSPRT with the solution approach for the TSPRTTW. The first-stage problem is similar, but the second-stage problem is different, leading to different optimality cuts. By applying the solution approaches to identical problem instances, we can compare the performance.

Table 11 reports the results for both methodologies on the instances eil51 and st70 for a sample size of $N = 200$. In both cases, the node limit is set at 10,000 and this limit is reached always. Clearly, the solutions for the TSPRT are better than those for the TSPRTTW. The reason is that the TSPRT approach is able to solve the problems closer to optimality, as we observe that the average SAA MIP gap for the TSPRT is lower. Because the number of optimality cuts is higher for the TSPRTTW, it may well be that these cuts are weaker than the cuts for the TSPRT. Another factor that plays a role is the graphs on which the problems are defined. The TSPRTTW is defined with directed arcs, so it contains twice as much variables as the TSPRT. This will also make it more difficult to solve the SAA MIPs. In addition, the variation in the SAA MIP gap is smaller for the TSPRT. Earlier this section, we mentioned that the reason for the negative gaps may be that some of the M problems are solved close the optimality, while for others the SAA MIP gap is still large. The table seems to confirm this explanation as the optimality gap seems to become lower (i.e. tending to $-\infty$) when the variation in the SAA MIP gap is higher.

Another clear difference between the two approaches is the computation time. The TSPRT is solved much faster. First of all, less optimality cuts are added in the TSPRT. Deriving those cuts is the most time-consuming part of the solution process. As we discussed in Section 4, the second-stage problem for the TSPRT is very straightforward, whereas for the TSPRTTW, N LP problems need to be solved. For illustration, the average time to derive an optimality cut for eil51 is 0.03 seconds for the TSPRT, but 2.75 seconds for the TSPRTTW. This is the most important factor for the difference in computation times. In addition, the graph for the TSPRTTW contains two directed arcs for each edge in the TSPRT. Hence, the number of variables is twice as high in the TSPRTTW, which also makes the problem more complex.

Table 11: Comparison of the methodology for the TSPRT and TSPRTTW for eil51 and st70

Instance	Meth.	$\bar{z}_N$	$\hat{\sigma}_{\bar{z}_N}$	MIP Gap	$\hat{\sigma}_{\text{MIP Gap}}$	$\hat{z}_{10^5}(\hat{x}^*)$	$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	Opt. Cuts	$\hat{\sigma}_{\text{Opt. Cuts}}$	Total CPU
eil51	TSPRT	542.7	5.21	8%	0.01	556.5	1.85	177.6	20.68	3m36s
	TSPRTTW	570.7	5.82	28%	0.02	575.3	1.95	189.1	22.67	1h32m28s
st70	TSPRT	783.0	3.25	11%	0.01	793.7	2.58	116.2	19.03	4m28s
	TSPRTTW	967.5	33.81	38%	0.06	852.4	2.83	159.1	19.91	1h49m52s

This table gives a detailed comparison of the methodology for the TSPRT and the TSPRTTW for the instances eil51 and st70. For the TSPRTTW, the time windows at the intermediate nodes are set very wide, such that they are never violated. The second column shows which type of methodology is adopted. The sample size we used is $N = 200$ and a node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the average remaining optimality gap for the SAA MIPs at the node limit and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the last column, the total computation time is shown.

6.5 Computational results with user cuts

Recall from Section 4 that there are two approaches to adding the subtour cuts. We can either add them when an integer solution is found or also at fractional nodes. For the second approach, we refer to the constraints added in fractional nodes as user cuts. The results for the first approach are shown in Sections 6.1, 6.2 and 6.3. Now, we report the results for the TSPRT and TSPRTTW with user cuts.

6.5.1 TSPRT

The motivation for adding subtour cuts in each node in branch-and-bound, is that we expect to find a better route using less nodes in the search tree. In Table 2, we found some negative gaps for the TSPRT which were larger than expected. We suspect that the optimality gap of the SAA MIP in each sample differs across the M samples, giving rise to the large negative gaps. In Table 12, we compare the results for running the algorithm with and without user cuts. We use identical samples (of size $N = 1000$) such that all the differences are due to the presence or absence of user cuts.

As expected, the presence of user cuts means that we are able to add more optimality cuts. The average number of optimality cuts is higher if user cuts are present for all instances. This also means that the SAA MIP in each sample can be solved further, since the average SAA MIP gap is always smaller with user cuts. For the larger instances, this difference is pronounced. For example, for rd100 the average SAA MIP gap is 5% with user cuts and 20% without them. Like we suspected, this leads to a reduction of the size of the negative gap. For kroE100, the negative gap is much larger than twice the standard deviation without user cuts. When user cuts are added, the negative gap is reduced strongly and it is slightly larger than twice the standard deviation. For kroD100, the negative gap disappears with user cuts and for rd100 the size of the negative gap is reduced. However, this comes at a cost. The computation time approximately doubles. This is not surprising, since the in each non-integer node, we have to check whether a user cut should be added. In addition, more optimality cuts are generated.

Table 12: Comparison of the results for the TSPRT with and without user cuts

Instance	eil51		kroD100		kroE100		rd100	
UserCuts	No	Yes	No	Yes	No	Yes	No	Yes
$\bar{z}_N$	544.8	543.4	25563.1	25395.3	26664.0	25275.8	9805.6	9448.5
$\hat{\sigma}_{\bar{z}_N}$	0.66	1.02	74.82	63.70	407.24	59.09	32.37	25.86
$\hat{z}_{10^5}(\hat{x}^*)$	548.8	548.8	25530.4	25487.5	25371.1	25022.7	9751.3	9424.7
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	1.81	1.83	83.02	82.15	82.15	80.71	31.69	30.46
Gap	4.1	5.4	-32.7	92.1	-1292.8	-253.1	-54.3	-23.8
$\hat{\sigma}_{\text{Gap}}$	1.93	2.09	111.76	103.95	415.44	100.03	45.30	39.96
MIP Gap	7%	6%	15%	7%	20%	6%	20%	5%
Opt. Cuts	198.3	268.6	160.5	211.7	76.3	98.1	105.1	149.4
$\hat{\sigma}_{\text{Opt. Cuts}}$	32.68	29.24	31.05	20.73	26.15	22.78	23.97	28.60
Total CPU	24m26s	47m21s	41m55s	1h36m3s	25m59s	1h21m40s	36m22s	1h42m50s

This table gives a detailed overview of the obtained results for the instances eil51, kroD100, kroE100 and rd100. For each instance, we show the results for $N = 1000$ with and without applying user cuts. Including user cuts means that we check for subtour cuts in each node of the branch-and-bound search, also in non-integer nodes. A node limit of 10,000 was used for the SAA MIP in each sample. The table shows the average lower bound, the upper bound, the optimality gap of the SAA MIPs, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. We also report the average of the remaining MIP gap of the SAA MIP in each sample. In the final row, the total computation time is shown.

6.5.2 TSPRTTW

For the TSPRTTW, we report the results for the solution approaches with and without user cuts in Table 13. Again, identical samples are used of size $N = 200$. The node limit is 10,000 and is always reached. Like for the TSPRT, better tours are found when user cuts are applied. The average lower bound is lower, because the presence of user cuts enables the solver to solve the SAA MIPs further. This also results in a lower upper bound. In addition, the size of the negative gap

between the upper and lower bound is reduced. However, the difference in the average SAA MIP gap is only slightly smaller with user cuts. Moreover, the size of the negative gap does not reduce when we measure it in terms of the standard deviation of the gap. Therefore, the advantage of including user cuts seems to be small.

In contrast to the results for the TSPRT, the computation time is lower when user cuts are present. The reduction in computation time is substantial. The main reason is that the number of optimality cuts is much higher if we do not add user cuts. This is a peculiar result, since we expect the solver to find more feasible tours in a given number of branch-and-bound nodes when user cuts are present. For each feasible tour that the solver finds, an optimality cut is added. Although more optimality cuts are added, the solutions found without user cuts are of inferior quality.

Table 13: Detailed solutions for the TSPRTTW for eil 51 with and without user cuts

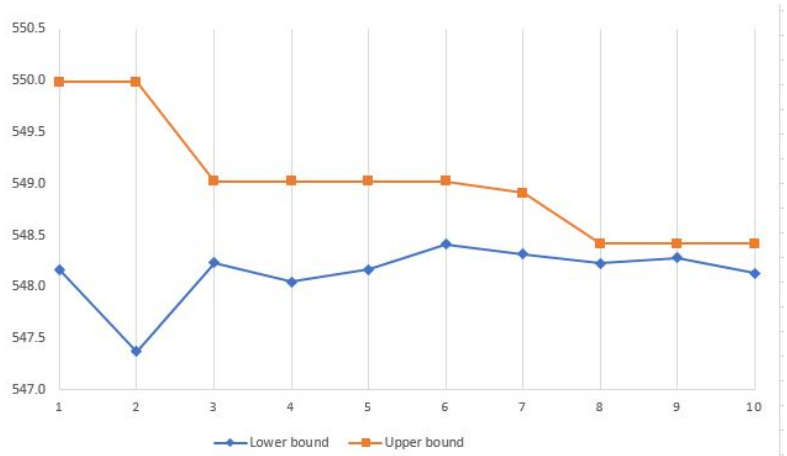
UserCuts	$\bar{z}_N$	$\hat{\sigma}_{\bar{z}_N}$	MIP Gap	$\hat{\sigma}_{\text{MIP Gap}}$	$\hat{z}_{10^5}(\hat{x}^*)$	$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	CPU	$\hat{\sigma}_{\text{CPU}}$	Opt. Cuts	$\hat{\sigma}_{\text{Opt. Cuts}}$	Total CPU
No	24456.3	1239.25	98.3%	0.00	20630.2	70.03	1526.6	248.47	553.5	82.90	4h14m41s
Yes	19615.4	705.97	97.8%	0.00	17226.7	58.79	386.3	43.04	88.4	19.18	1h4m34s

This table gives a detailed overview of the obtained results for the instance eil51 both with and without user cuts. The sample size we used is $N = 200$. The table shows the average lower bound, the average SAA MIP gap, the upper bound, the average computation time and the average number of optimality cuts. Each of these values is accompanied by the standard deviation. In the last column, the total computation time is shown.

6.6 Running the SAA multiple iterations with user cuts

As explained in Section 4.1, we can generate M samples multiple times and solve the SAA MIPs until the difference between the upper and lower bound converges to a small value ϵ. Although this is also mentioned in Verweij et al. (2003), they only report results for a single iteration. Therefore, we run the SAA for multiple iterations here. In each iteration, we update the upper bound in the usual sense. If a smaller upper bound is found, we update it, but otherwise we keep the same upper bound. For the lower bound, this process is different. In iteration l, the lower bound is given by the average of the objective values found in all lM SAA MIPs solved previously, as defined in Equation (33). Therefore, the lower bound can also decrease in value and the gap between the upper and lower bound does not always decrease monotonically. We ran 10 iterations for the TSPRT of the instance eil51. A node limit of 10,000 was used and user cuts were applied. The size of each sample is $N = 1000$.

Figure 4: Running 10 iterations of the SAA for the TSPRT of eil51



The SAA is run for 10 iterations for the TSPRT of the instance eil51. The best upper and lower bound are shown after each iteration. A sample size of $N = 1000$ is used and the node limit is 10000.

The results are shown in Figure 4. Clearly, the optimality gap does not decrease monotonically. This is due to the fluctuation in the lower bound. The upper bound is an approximation with a very small standard deviation, since it is based on a sample size of $N' = 100000$. On the other hand, the standard deviation of the lower bound can be relatively large in the first iterations. For example, in the first iteration, the lower bound is based on the average of the objective values of $M = 10$ SAA MIPs, each based on a sample of size $N = 1000$. Therefore, the lower bound can fluctuate in the first few iterations, but its standard deviation will decrease in higher iterations.

This can also be seen in the figure, because in the last few iterations, the lower bound is relatively constant. At the ninth iteration, the optimality gap is 0.14 which is 0.03% of the lower bound. This seems to indicate that the optimality gap will converge to zero if more iterations are added. This suggests that the SAA method is a suitable approach for solving two-stage SRPs.

7 Conclusion

In this thesis, we applied the sample average approximation (SAA) method to the shortest path problem, traveling salesman problem and the traveling salesman problem with soft time windows, all with random travel times. The first two problems have also been investigated by Verweij et al. (2003). We replicate their solution method, which implements the SAA in a branch-and-cut framework based on Benders decomposition. We find similar results, for example that the number of optimality cuts is constant in the sample size used in the SAA and that hence, the computation time grows linearly with the sample size. We also investigated different approaches for adding subtour elimination constraints to the traveling salesman problem with random travel times. We find that adding the constraints also at fractional solutions, instead of only at integer solutions, leads to a better solution when using a given number of nodes in the branch-and-bound tree. Therefore, it is recommended to use this approach when computer memory is limited. It remains to be seen whether it is also requires less computation time. This would be interesting to investigate in future research. We also run the SAA for multiple iterations and observe that the gap between the upper and lower bound converges to zero. However, the gap does not decrease monotonically due to the way in which the lower bound is defined in the SAA.

We extended the SAA to the traveling salesman problem with random travel times and soft time windows (TSPRTTW). We are not able to find good solutions to the deterministic SAA MIP arising for each sample, using 10000 nodes in the branch-and-bound search. For most SAA MIPs, an optimality gap of 98% remained at the node limit. Adding subtour elimination constraints also at fractional branch-and-bound nodes, slightly improves the solution quality.

For all three types of problems, we find that it is important to have a good formulation and solution algorithm for solving the SAA MIPs. If the remaining optimality gaps are large, we may find differences in the optimality gaps across the samples. This can result in a negative gap between the upper and lower bound that are computed with the SAA. We often find these negative gaps for the TSPRTTW. Therefore, an interesting road ahead for future research is to find an improved formulation or solution algorithm for the TSPRTTW. In addition, test instances are needed for this type of problem.

8 Appendix

8.1 Pseudocode for the Hao and Orlin (1992) algorithm

Besides the notation developed in Section 4.5, we define $d(i)$ as the distance label corresponding to node $i \in V$. This label is an integer in the range $\{0, 1, \ldots, |V|\}$. These labels are used to make sure that the flow is sent in the direction of the sink. Flow can only be sent to nodes with a lower distance label. Therefore, the sink will have the lowest distance label.

Furthermore, the nodes are divided in two sets, D and W. The set D denotes the set of dormant nodes and W the set of awake nodes. Nodes belong to D or W, depending on which condition they satisfy. A node i belongs to D if there is no arc (i, j) such that $r_{ij} > 0$ and $j \in W$. On the other hand, for two nodes i and j in W, it should hold that if $r_{ij} > 0$ then $d(i) \leq d(j) + 1$. Then, we can define a node i to be active if $i \in W \setminus \{t\}$ and $e(i) > 0$. An arc (i, j) is admissible if $i, j \in W, r_{ij} > 0$ and $d(i) = d(j) + 1$. The Hao and Orlin algorithm is then given by the following set of methods:

Algorithm *FindMinCut(s)*
Initialize;
while $S \neq V$ **do**
  while G contains an active node **do**
    Select an active node i;
    if G contains an admissible arc (i, j) **then**
      push $\delta \leftarrow \min\{e(i), r_{ij}\}$ units of flow from node i to node j;
    else Relabel(i);
    end if
  end while
  if $u(D, W) <$ BestValue **then**
    Cut $\leftarrow (D, W)$;
    BestValue $\leftarrow u(D, W)$;
  end if
  SelectNewSink;
end while

Algorithm *Initialize*
for each arc $(0, j)$, send r_{0j} units of flow on $(0, j)$;
DormantSet(0) $\leftarrow \{0\}$;
$D_{\max} \leftarrow 0$;
$W \leftarrow V \setminus \{0\}$;
$t \leftarrow 1$;
$d(t) \leftarrow 0$;
for each $j \in V \setminus \{t\}$ **do**
  $d(j) \leftarrow 1$;
end for

Algorithm *Relabel(i)*
if i is the only node in W with distance label $d(i)$ **then**
    $D_{\max} \leftarrow D_{\max} + 1$;
    $R \leftarrow \{j \in W : d(j) \geq d(i)\}$;
    DormantSet($D_{\max}$) $\leftarrow R$;
    $W \leftarrow W \setminus R$;
else if there is no arc (i, j) such that $r_{ij} > 0$ and $j \in W$ **then**
    $D_{\max} \leftarrow D_{\max} + 1$;
    DormantSet($D_{\max}$) $\leftarrow \{i\}$;
    $W \leftarrow W \setminus \{i\}$;
else $d(i) \leftarrow \min\{d(j) + 1 : (i, j) \in A, j \in W \text{ and } r_{ij} > 0\}$;
end if

Algorithm *SelectNewSink*
delete t from W;
add t to both set S and DormantSet(0);
if S=V **then**
    quit;
end if
for each arc $(t, k) \in A$ with $k \in V \setminus S$ **do**
    send r_{tk} units of flow on arc (t, k);
end for
if $W = \emptyset$ **then**
    $W \leftarrow$ DormantSet($D_{\max}$);
    $D_{\max} \leftarrow D_{\max} - 1$;
end if
select $j \in W$ such that $d(j)$ is minimum;
$t \leftarrow j$;

8.2 More detailed results for some instances

Here we report more detailed results for the instances kroD100, kroE100 and rd100. For some large instances, we found a large negative gap for the TSPRT with a sample size of $N = 1000$. As can be seen from the tables, we do not find a negative gap for all sample sizes.

Table 14: Detailed results for the TSPRT for the instance kroD100

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
kroD100									
$\bar{z}_N$	25427.4	25538.9	25511.7	25305.6	25653.5	25444.5	25503.5	25697.6	25545.8
$\bar{\sigma}_{\bar{z}_N}$	79.14	130.64	78.04	125.00	72.29	92.77	81.78	63.28	78.52
$\hat{z}_{10^5}(\hat{x}^*)$	25891.0	25611.3	25623.2	25504.6	25540.3	25480.8	25486.7	25714.2	25596.3
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	84.00	83.14	83.24	83.68	82.93	82.45	82.40	83.06	82.46
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	37.1	50.3	69.4	78.3	106.0	139.9	174.8	235.9	267.6
$\hat{\sigma}_{\text{CPU}}$	4.51	6.65	7.81	12.03	10.72	21.52	23.51	29.15	53.88
Opt. Cuts	145.2	148.9	161.4	136.8	145.1	150.6	156.5	165.3	163.1
$\hat{\sigma}_{\text{Opt. Cuts}}$	29.57	23.05	12.70	23.90	17.47	24.15	25.20	22.53	35.29
Total CPU	6m15s	8m28s	11m38s	13m7s	17m24s	23m23s	29m53s	39m27s	44m41s

This table shows a detailed overview of the results obtained for the kroD100 instance for the TSPRT. Lower bounds, upper bounds, number of nodes, computation time and computation time are shown with their standard deviation. In addition, the total CPU time is shown. A node limit of 10,000 is used for the SAA MIP in each sample.

Table 15: Detailed results for the TSPRT for the instance kroE100

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
kroE100									
$\bar{z}_N$	26002.6	26218.6	26350.9	26151.0	27010.6	26345.6	26629.4	26531.2	26435.1
$\bar{\sigma}_{\bar{z}_N}$	218.10	210.42	241.47	195.49	314.99	160.61	207.04	215.77	292.38
$\hat{z}_{10^5}(\hat{x}^*)$	25474.4	25709.9	25773.8	25329.5	26067.1	25833.4	25916.7	25700.3	24963.6
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	82.09	83.38	83.75	82.17	85.05	83.04	83.96	82.90	80.27
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	36.4	43.5	58.4	70.8	95.4	101.8	137.4	179.3	207.3
$\hat{\sigma}_{\text{CPU}}$	6.13	7.69	11.87	17.58	21.23	21.52	25.33	38.24	52.06
Opt. Cuts	90.2	73.4	94.5	86.1	86.3	81.0	95.3	97.7	96.7
$\hat{\sigma}_{\text{Opt. Cuts}}$	26.89	17.34	41.96	25.19	26.42	28.25	31.19	24.62	31.07
Total CPU	6m9s	7m20s	9m50s	11m54s	15m0s	17m4s	22m59s	29m58s	34m49s

This table shows a detailed overview of the results obtained for the kroE100 instance for the TSPRT. Lower bounds, upper bounds, number of nodes, computation time and computation time are shown with their standard deviation. In addition, the total CPU time is shown. A node limit of 10,000 is used for the SAA MIP in each sample.

Table 16: Detailed results for the TSPRT for the instance rd100

	Number of sample scenarios (N)								
	200	300	400	500	600	700	800	900	1000
rd100									
$\bar{z}_N$	9943.4	9984.9	9747.5	10036.7	9900.1	9801.1	9841.7	9880.2	10184.5
$\bar{\sigma}_{\bar{z}_N}$	119.76	180.39	48.43	210.69	75.14	60.71	88.77	93.77	221.54
$\hat{z}_{10^5}(\hat{x}^*)$	9938.9	9747.1	9691.1	9676.5	9785.6	9681.9	9633.4	9628.2	9686.3
$\hat{\sigma}_{\hat{z}_{10^5}}(\hat{x}^*)$	32.33	31.46	31.41	31.65	31.81	31.30	31.40	31.31	31.43
Nodes	10000	10000	10000	10000	10000	10000	10000	10000	10000
$\hat{\sigma}_{\text{Nodes}}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
CPU	44.7	52.1	61.9	81.9	106.3	130.9	147.0	230.9	179.1
$\hat{\sigma}_{\text{CPU}}$	6.83	5.31	8.49	14.90	20.70	27.34	15.34	46.15	42.00
Opt. Cuts	95.9	76.3	73.7	89.1	91.9	100.7	95.8	122.3	76.9
$\hat{\sigma}_{\text{Opt. Cuts}}$	39.33	19.95	20.53	26.81	29.84	32.09	16.65	23.61	27.26
Total CPU	7m33s	8m46s	10m25s	13m45s	17m49s	21m54s	24m55s	38m33s	29m55s

This table shows a detailed overview of the results obtained for the rd100 instance for the TSPRT. Lower bounds, upper bounds, number of nodes, computation time and computation time are shown with their standard deviation. In addition, the total CPU time is shown. A node limit of 10,000 is used for the SAA MIP in each sample.

References

Ahmed, S., Shapiro, A., & Shapiro, E. (2002). *The Sample Average Approximation Method for Stochastic Programs with Integer Recourse.*

Balaprakash, P., Birattari, M., Stutzle, T., & Dorigo, M. (2009). Adaptive Sample Size and Importance Sampling in Estimation-Based Local Search for the Probabilistic Traveling Salesman Problem. *European Journal of Operational Research*, *199*(1), 98–110.

Benders, J. F. (1962). Partitioning Procedures for Solving Mixed-Variables Programming Problems. *Numerische mathematik*, *4*(1), 238–252.

Bianchi, L., Knowles, J., & Bowler, N. (2005). Local search for the probabilistic traveling salesman problem: Correction to the 2-p-opt and 1-shift algorithms. *European Journal of Operational Research*, *162*(1), 206–219.

Blomvall, J., & Shapiro, A. (2006). Solving Multistage Asset Iinvestment Problems by the Sample Average Approximation Method. *Mathematical Programming*, *108*(2-3), 571–595.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem* (Tech. Rep.). Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Dantzig, G. B., Fulkerson, R., & Jonhson, S. (1954). Solution of a Large-Scale Traveling-Salesman Problem. *Journal of the Operations Research Society of America*, *2*(4), 393–410.

Desrosiers, J., M.Sauve, & Soumis, F. (1988). Lagrangian Relaxation Methods for Solving the Minimum Fleet Size Multiple Traveling Salesman Problem with Time Windows. *Management Science*, *34*(8), 1005–1022.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*(1), 269–271.

Donati, A. V., Montemanni, R., Casagrande, N., Rizzoli, A. E., & Gambardella, L. M. (2008). Time dependent vehicle routing problem with a multi ant colony system. *European journal of operational research*, *185*(3), 1174–1191.

Emelogu, A., Chowdhury, S., Marufuzzaman, M., Bian, L., & Eksioglu, B. (2016). An Enhanced Sample Average Approximation Method for Stochastic Optimization. *International Journal of Production Economics*, *182*(1), 230–252.

Flood, M. M. (1956). The Traveling-Salesman Problem. *Operations Research*, *4*(1), 61–75.

Gendreau, M., Laporte, G., & Seguin, R. (1995). An Exact Algorithm for the Vehicle Routing Problem with Stochastic Customers and Demands. *Transportation Science*, *29*(2), 143–155.

Gendreau, M., Laporte, G., & Séguin, R. (1996). Stochastic Vehicle Routing. *European Journal of Operational Research*, *88*(1), 3–12.

Hao, J., & Orlin, J. B. (1992). A Faster Algorithm for Finding the Minimum Cut in a Graph. In *Proceedings of the Third Annual Acm-Siam Symposium on Discrete Algorithms.*

Hill, A. V., & Benton, W. (1992). Modelling intra-city time-dependent travel speeds for vehicle scheduling problems. *Journal of the Operational Research Society*, *43*(4), 343–351.

Homem-de Mello, T. (2008). On Rates of Convergence for Stochastic Optimization Problems under Non–Independent and Identically Distributed Sampling. *SIAM Journal on Optimization*, *19*(2), 524–551.

Jaillet, P. (1985). *Probabilistic Traveling Salesman Problems* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

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

Kenyon, A. S., & Morton, D. P. (2003). Stochastic Vehicle Routing with Random Travel Times. *Transportation Science*, *37*(1), 69–82.

Kleywegt, A. J., Shapiro, A., & de Mello, T. H. (2002). The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, *12*(2), 479–502.

Lambert, V., Laporte, G., & Louveaux, F. (1993). Designing Collection Routes through Bank Branches. *Computers & Operations Research*, *20*(7), 783–791.

Laporte, G. (1992). The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research*, *59*(2), 231–247.

Laporte, G., Louveaux, F., & Mercure, H. (1992). The Vehicle Routing Problem with Stochastic Travel Times. *Transportation Science*, *26*(3), 161–170.

Laporte, G., Louveaux, F. V., & Mercure, H. (1994). A Priori Optimization of the Probabilistic Traveling Salesman Problem. *Operations Research*, *42*(3), 543–549.

Li, P. Y. (2014). Sample Average Approximation Method for a Class of Stochastic Generalized Nash Equilibrium Problems. *Journal of Computational and Applied Mathematics*, *261*(1), 387–393.

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, *21*(2), 498–516.

Mak, W. K., Morton, D. P., & Wood, R. K. (1999). Monte Carlo Bounding Techniques for Determining Solution Quality in Stochastic Programs. *Operations Research Letters*, *24*(1), 47–56.

Nemirovski, A., Juditsky, A., Lan, G., & Shapiro, A. (2009). Robust Stochastic Apprixmation Approach to Stochastic Programming. *SIAM Journal on Optimization*, *19*(4), 1574–1609.

Pagnoncelli, B. K., Ahmed, S., & Shapiro, A. (2009). Sample Average Approximation Method for Chance Constrained Programming: Ttheory and Applications. *Journal of Optimization Theory and Applications*, *142*(2), 399–416.

Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *ORSA journal on computing*, *3*(4), 376–384.

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*(3), 400–407.

Slyke, R. M. V., & Wets, R. (1969). L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *Society for Industrial and Applied Mathematics*, *17*(4), 638–663.

Stewart Jr, W. R., & Golden, B. L. (1983). Stochastic Vehicle Routing: A Comprehensive Approach. *European Journal of Operational Research*, *14*(4), 371–385.

Tang, Q., & Qian, P. Z. G. (2010). Enhancing the Sample Average Approximation Method with U Designs. *Biometrika*, *97*(4), 947–960.

Taş, D., Dellaert, N., van Woensel, T., & de Kok, T. (2014). The time-dependent vehicle routing problem with soft time windows and stochastic travel times. *Transportation Research Part C: Emerging Technologies*, *48*, 66–83.

Tillman, F. A. (1969). The Multiple Terminal Delivery Problem with Probabilistic Demands. *Transportation Science*, *3*(3), 192–204.

Verweij, B., Ahmed, S., Kleywegt, A. J., Nemhauser, G., & Shapiro, A. (2003). The Sample Average Approximation Method Applied to Stochastic Routing Problems: A Computational Study. *Computational Optimization and Applications*, *24*(1), 289–333.

Wang, W., & Ahmed, S. (2008). Sample Average Approximation of Expected Value Constrained Stochastic Programs. *Operations Research Letters*, *36*(1), 515–519.