Exact, Heuristic and Machine Learning Approaches to The Probabilistic Travelling Salesman Problem with Crowdsourcing

Alberto Santini[1], Ana Viana[2], Xenia Klimentova[3], and João Pedro Pedroso[4]

[1]Universitat Pompeu Fabra, Barcelona, Spain — *alberto.santini@upf.edu*
[2]INESC TEC and Polithecnic of Porto, Porto, Portugal — *ana.viana@inesctec.pt*
[3]INESC TEC, Porto, Portugal — *xenia.klimentova@inesctec.pt*
[4]INESC TEC and Universidade do Porto, Portugal — *jpp@fc.up.pt*

March 5, 2020

Abstract

We study a variant of the Probabilistic Travelling Salesman Problem arising when retailers crowdsource last-mile deliveries to their own customers, who can refuse or accept in exchange for a reward. A planner has to identify which deliveries to offer, knowing that all deliveries need fulfilment, either via crowdsourcing or using an own vehicle. We formalise the problem and position it in both the literature about crowdsourcing and among routing problems in which not all customers need a visit. As a binary minimisation problem, we show that its objective function is highly non-linear and, to evaluate it, one needs to solve an exponential numbers of Travelling Salesman Problems. To address this complexity, we propose Machine Learning and Monte Carlo simulation methods to approximate the objective function, and heuristics to reduce the number of evaluations. We show that these approaches work well on small size instances and derive managerial insights on the benefits of crowdsourcing to customers.

1 Introduction

With the increasing growth of e-commerce worldwide, business models for last-mile delivery (LMD) of parcels need to innovate and consider fast, cheap and reliable transportation to end customers. One possibility is to crowdsource some deliveries, through digital platforms, to non-professional couriers who use their own transportation means. While reducing costs and providing a source of income for people who might otherwise be off the labour market [22], crowdsourcing also raises concerns about job quality, environmental impact [32] and trust [26].

One main motivation for this work is to investigate the practice of *crowdsourcing delivery to end customers* as a system that takes advantage of the benefits of crowdsourcing while mitigating the major associated drawbacks. We address the case of retail companies with physical shops that also sell online.

For example, consider a supermarket chain that offers home delivery of groceries. In traditional LMD, a professional fleet either owned by the chain or outsourced, would deliver the groceries. In current crowdsourced LMD models, the chain would have a platform where potential couriers enrol, get offers for some deliveries, and accept or refuse the offers. This is the model of, e.g., Amazon Flex. Because there is no consolidation of parcels and couriers make journeys that they otherwise wouldn't, this system generates both extra traffic and emissions.

Now imagine that the chain has a loyalty programme in which the clients enrolled provide their home address. We defend a slightly different model where *the crowd* is composed of clients that are already in the store and whose home address is close to one of the delivery points. Because the clients who accept were already heading home, we reduce the number of cars associated to the ecosystem supermarket-delivery and reduce the number of miles travelled. Under this model, the planner offers participating clients to deliver someone else's groceries in exchange for a discount. Since clients can accept or refuse the offer, at some point in the day a supermarket vehicle will serve all deliveries which the planner did not *crowdsource*. A similar scheme was first discussed in the work of Archetti, Savelsbergh, and Speranza [7], in which the authors

consider that either a professional fleet or occasional drivers can carry out deliveries, but assume that the occasional drivers will always accept offers.

In this work we assume that each crowdsourcing fee (i.e., the discount offered to the customer) is fixed and that the probability that some customer accepts a delivery during the day, provided the decision-maker offers it, is also fixed and known. We also assume that the supermarket's fleet comprises a single vehicle, although this assumption is easy to relax.

Under the assumptions above, our problem becomes a stochastic generalisation of the Travelling Salesman Problem which we name the **Probabilistic Travelling Salesman Problem with Crowdsourcing (PTSPC)**. Other generalisations of the TSP which do not require to visit all customers have been studied extensively. In particular, the literature considers the extreme cases in which the tour planer has either *no power* or *total power* at deciding which clients shouldn't be visited. In Appendix 2 we position the PTSPC in this spectrum, as an intermediate problem in which the planner has partial power. We also provide a formal model and discuss the intrinsic difficulty of the problem and the need to resort to advanced optimisation techniques to scale problem size. Appendix 3 describes the state-of-the-art in LMD with crowdsourcing and details some literature on related problems, including the Probabilistic Travelling Salesman Problem and the Probabilistic Travelling Salesman Problem with Profits. Appendix 4 describes exact and heuristic approaches for the PTSPC and Appendix 5 reports computational results on small and medium-size instances. Finally, in Appendix 6 we draw conclusions and identify future lines of research.

2 Problem description

In this section we place the PTSPC in a spectrum of TSP generalisations which do not require to visit all customers, along a gradient of "decision power" attributed to the tour planner. We also formalise the problem and introduce the notation used throughout the rest of the paper.

2.1 Contextualisation

As mentioned in the introduction, the Probabilistic Travelling Salesman Problem with Crowdsourcing is an intermediate problem between two extreme variants of the TSP which do not require to visit all customers. At one extreme lies the case in which the tour planner has no choice over which customer will require a delivery (we assume we visit customers to deliver some goods) and which will not, because a random variable determines customer presence. This problem is the *Probabilistic Travelling Salesman Problem* (PTSP) [38]. In the PTSP a decision-maker has to devise a tour to visit a set of delivery points, some of which might be later revealed not to be available. Because the decision-maker doesn't know in advance which customers will drop out, they face two options. The first option, called the *reoptimisation* approach, is to wait until the status of all customers is revealed and, at that point, plan a TSP tour visiting the customers requiring delivery. This strategy can be impractical if there is not enough time to compute a solution after the outcome is revealed. It can also be inconvenient for operational reasons, for example if it produces every day a radically different tour while still visiting a similar set of delivery points. The second option, called the *a priori* approach, is to first plan a tour visiting all the customers. When the stochastic outcome is revealed, the decision-maker amends the solution skipping the deliveries that are not required, and performing the remaining ones in the same order as they appear in the *a priori* tour. This is the approach most commonly used, introduced together with the definition of PTSP in [35].

At the opposite extreme there is the case in which the tour planner has total control over which deliveries to perform, giving rise to the *Profitable Tour Problem* (PTP) [25, 27]. In this case, visiting a delivery point earns a profit, while traversing an edge incurs into a cost. The objective is to select the deliveries and plan the tour that maximise the difference between collected profits and travel costs.

Intermediate cases arise when the visit requirements are stochastic, but the decision-maker has some leverage on their outcome. There are many ways in which the decision-maker can affect the outcome of the random variables associated with customer acceptance. In the PTSPC the planner has the power of forcing a delivery with the own vehicle if they never offer that delivery. One could imagine more complicated interactions in which, e.g., the decision-maker can increase (decrease) the discount offered to raise (lower) the probability of customers accepting a delivery [11].

As a motivation for such approach, in Appendix 1 we described the case of a supermarket chain crowdsourcing grocery deliveries. Another real-life scenario is that of an on-line retailer which operates lockers where customers can pick-up their orders. When customers visit the locker to receive a parcel, they see a message asking if they want to deliver another parcel in exchange for a gift card or coupon. As in the supermarket example, parcels not accepted for crowdsourcing are then delivered by the standard last-mile segment of the retailer supply chain. Finally, we remark how this problem can have applications beyond LMD, e.g., in social-care problems where pharmacies ask their customers to deliver medicines to their elderly neighbours.

2.2 Formalisation of the PTSPC

Consider a complete undirected graph $G = (V', E)$ with vertex set $V' = \{0, 1, \ldots, n\}$. Vertex 0 represents the depot, while $V = \{1, \ldots, n\}$ is the set of delivery locations. Let $c_{ij} \in \mathbb{R}^+$ be the cost of traversing an edge $\{i, j\} \in E$ and assume that, if the planner offers delivery $i \in V$ for crowdsourcing, there is a probability $p_i \in [0, 1]$ that some provider will accept the offer. In this case, the decision-maker pays a fee $m_i \in \mathbb{R}^+$ to the courier. Otherwise, if the offer is not accepted, the planner needs to visit i with the own vehicle.

We assume that probabilities p_i are independent which, under our motivating example is a reasonable approximation: we expect the number of potential couriers to be larger than the number of deliveries, and to offer at most one delivery to each customer. Different assumptions might hold if the planner outsourced deliveries to a logistic provider. For example, the provider could accept or refuse in block deliveries in a same area.

Denote with $O \subseteq V$ the subset of deliveries offered for crowdsourcing and with $A \subseteq O$ the set of accepted offers, which is only revealed at the end of the day. The decision-maker has to decide which deliveries to offer for crowdsourcing (i.e., the elements of O), assuming that they will reoptimise the end-of-day tour after set A is revealed. In other words, they look for the set O with the lowest expected cost with respect to the random variable A.

This approach can appear similar to the reoptimisation approach for the PTSP, but the two differ in an important aspect. In the PTSP the decision-maker cannot affect the outcome of the random variables: they will wait for their realisation and then solve one single TSP over the customers requiring a visit. In short, the reoptimisation PTSP becomes equivalent to a TSP on the operational level. On the tactical level, the PTSP decision-maker can calculate the expected cost of the reoptimised tour, but still (different from our problem) cannot act to decrease it.

Let $\mathrm{TSP}(V \setminus A)$ be the *reoptimised* tour of the PTSPC: the shortest simple tour starting and ending at the depot, visiting all delivery points which were either not offered, or whose offer for crowdsourcing was not accepted. Let $c_{V \setminus A}$ be the total cost of such tour, obtained by summing the edge costs. Under the reoptimisation approach, the cost associated with offering deliveries O and having deliveries A accepted, $C_{\mathrm{r}}(O, A)$, is the sum of the crowdsourcing fees for the accepted deliveries plus the cost of the tour visiting all other delivery points:

$$C_{\mathrm{r}}(O, A) = \sum_{i \in A} m_i + c_{V \setminus A}$$

The probability that a particular set $A \subseteq O$ is the set of accepted deliveries is $\prod_{i \in A} p_i \prod_{i \in O \setminus A} (1 - p_i)$. We can then calculate, for a fixed set O of deliveries offered for crowdsourcing, what is the expected cost $\mathbb{E}_A\big[C(O)\big]$ over all possible realisations of A.

$$\mathbb{E}_A\big[C(O)\big] = \sum_{A \subseteq O} \Bigg[\underbrace{\left(\prod_{i \in A} p_i \prod_{i \in O \setminus A} (1 - p_i) \right)}_{\text{Prob. that } A \text{ is the accepted set}} \cdot \underbrace{\left(\sum_{i \in A} m_i + c_{V \setminus A} \right)}_{\text{Cost when } A \text{ is the accepted set}} \Bigg] \tag{1}$$

The objective of the problem is to find the set O^{opt} which gives the lowest expected cost for the reoptimised tour:

$$O^{\mathrm{opt}} = \operatorname*{arg\,min}_{O \subseteq V} \ \mathbb{E}_A\big[C(O)\big] \tag{2}$$

Finding a solution to this problem by complete enumeration is hard because we have to iterate over all sets $O \subseteq V$ and, to compute each term $\mathbb{E}_A\big[C(O)\big]$, we have to iterate again over all $A \subseteq O$. Furthermore, at each

inner iteration we have to solve a TSP, which is $\mathcal{NP}$-complete. To avoid full enumeration (e.g., with a branch-and-bound algorithm) we would have to produce a lower bound on the objective function (2), which seems hard if not outright impossible.

Note that in our motivating examples the set of delivery points changes every day. In this case there are no operational considerations hindering the use of the reoptimisation strategy, which would only be limited by its high computational cost. If the set of delivery points tend to be stable, operational considerations make an *a priori* tour approach more attractive (see Appendix A).

Relationship with classical TSP problems The PTSPC is a generalisation of three well-studied routing problems. When the crowdsourcing costs are large, $m_i = \infty \; \forall i \in V$, there is no incentive to offer any delivery for crowdsourcing and the planner will decide to perform all deliveries with the own vehicle. In this case, the PTSPC becomes the classical **Travelling Salesman Problem**. The reduction to the TSP also happens when all probabilities of providers accepting a crowdsourcing offer are zero, $p_i = 0 \; \forall i \in V$. When there are no crowdsourcing fees, $m_i = 0 \; \forall i \in V$, the planner will try to crowdsource all deliveries. At that point, the sole factor determining which deliveries to make with the own vehicle and which not, is the stochasticity related to providers' acceptance probabilities p_i and we are back to the **Probabilistic Travelling Salesman Problem**. When the customers always accept crowdsourcing offers, $p_i = 1 \; \forall i \in V$, the problem reduces to the **Profitable Tour Problem** with prizes for performing deliveries with the own vehicle equal to the savings of the corresponding crowdsourcing fees.

3 Literature review

In this section we position the PTSPC in a broader context, highlighting the characteristics it shares with other non-deterministic routing problems. First, in Section 3.1, we mention recent studies proposing optimisation models to integrate crowdsourcing into last-mile delivery, as this is one of the main applications for the PTSPC. In Section 3.2 we analyse the Probabilistic TSP, the best-known non-deterministic routing problem. Then, in Section 3.3, we compare the PTSPC with a class of problems introduced in 2017 by Zhang, Wang, and Liu [57], known as Probabilistic TSP with Profits and Stochastic Customers.

3.1 Crowdsourcing in last-mile delivery

In a recent survey, Alnaggar, Gzara, and Bookbinder [2] review operational research literature on crowdsourced LMD and propose a classification of problems arising in the field. Under their classification scheme, the PTSPC focuses on *e-retailers* (i.e., the company offering the delivery is the same that sells the product) as opposed to couriers (i.e., a company that handles deliveries for third parties), it offers a per-delivery rate determined by the company (other possibilities are per-hour and negotiated rates), and uses pre-planned trips (drivers were already heading in the direction of the delivery points). From the point of view of the target market, the PTSPC focuses on self-scheduling individuals (the customers enter the supermarket at their own convenience, not at specified times) and short-haul deliveries (deliveries within the same city). These characteristics set the PTSPC apart from most other problems considered in the literature, which focus more on crowdsourcing to entities performing the delivery professionally, rather than truly occasional drivers.

Archetti, Savelsbergh, and Speranza [7] were among the first to address a problem arising in outsourcing in LMD: the Vehicle Routing Problem (VRP) with Occasional Drivers (ODs). In this problem, the company can decide whether to serve a delivery with a vehicle of its own fleet, or to outsource it for a fixed fee. The planner assigns deliveries to ODs which are already heading towards a destination. For the assignment to take place, the delivery point needs to be close to the driver's destination. The model works under the assumption that ODs always accept requests from the company, provided they fulfil this "closeness" condition and there is no stochasticity on acceptance. This assumption is important, as optimal solutions tend to use a high percentage of available drivers. The authors propose a Mixed-Integer Programming (MIP) formulation with arc-indexed variables and a multi-start heuristic to tackle instances with more than 25 deliveries. They identify three characteristics affecting the profitability of such a schema: the number of available drivers, their flexibility (how far the delivery point can be from the OD's original destination) and the compensation amount.

Other authors extended the VRP with ODs model to incorporate real-life features such as time windows, multiple and split deliveries [42], transshipment nodes [41], and coordinating ODs on bikes or on foot with

a delivery truck from which they relay [36, 33]. The MIP model by Huang and Ardiansyah [33] illustrates well how even deterministic problems can be hard to solve, when they require the interaction of traditional and crowdsourced supply chain segments. The authors could solve instances with up to 15 delivery points using the Gurobi solver. Although they do not report computing times, the authors state that by using the MIP model they could not solve instances with 20 and 30 customers and sometimes, after a 4-hour time limit, the solver did not even provide an integer feasible solution.

Recent literature also addressed dynamic versions of the problem, in which delivery requests and driver availability become known during the day. Arslan et al. [8] consider a real-time system in which drivers make *trip announcements* and the company can then assign them pickups and deliveries which are compatible with their announced travel (i.e., involving only a small detour) and the recipients' time windows. They tackle the dynamic nature of the problem with a rolling horizon approach, corresponding to a planner who takes decisions at different moments during the day. The decisions consist in matching deliveries to drivers and routing the own fleet for deliveries which were not crowdsourced. With a simulation study the authors show how this approach can produce savings, depending on the time flexibility of the drivers and their willingness to take larger detours. They also conclude that this system is more indicated when all parcels share the same origin, such as a central depot, as this greatly reduces the cost to operate the own fleet.

Dayarian and Savelsbergh [24] study a problem which shares some characteristic of the PTSPC. Their model also addresses the case of using in-store customers as occasional drivers and an own fleet in charge of completing the distribution of parcels. Different from the PTSPC, they simulate each customer individually; if a customer has a destination compatible with a delivery point, they assume that the customer will accept the delivery. The authors consider two cases: a static case, where all customer visits and deliveries are known in advance, and a dynamic case, where information is only known up to a certain time and the planner reoptimises following a rolling horizon approach. Data on the presence and destination of customers is collected while the customers shop inside the store. For the dynamic case, the authors also propose to base the decisions at each epoch on forecasts of future demand and in-store visits, which they obtain averaging sample scenarios (e.g., generated using historical data). Through a simulation study, the authors determine a trade-off between the savings obtained by reducing the own fleet and the risk introduced when relying on customer availability.

Dahle, Andersson, and Christiansen [23] model occasional drivers as stochastic vehicles possibly appearing at random times during the day (according to a known distribution) in a VRP with time windows. The authors use a two-stage stochastic model in which they plan the routes of the own fleet in the first stage and, after OD appearance times are revealed, they amend them and assign deliveries to ODs in the second stage. They also allow the possibility of not fulfilling some deliveries and pay a corresponding penalty. The authors solve instances with 5, 10, 15 or 20 deliveries, 2 own vehicles, and 2 or 3 ODs. To do so, they use a MIP model and enumerate all possible $2^{\mathcal{K}^O}$ scenarios, where $\mathcal{K}^O$ is the set of occasional drivers. The problem proves hard to solve: for example, they cannot find the optimal solution within 2 hours of computation for instances with 10 delivery points and 3 ODs.

Finally, Gdowska, Viana, and Pedroso [30] introduced a multi-vehicle problem for delivery with crowdshipping based on the VRP with ODs [7], which is most similar to our problem. In their work, the authors consider a multi-vehicle fleet of own vehicles and propose an agent-oriented bi-level stochastic model and a local search algorithm for its solution. With computational experiments on instances with 15 deliveries, they show that solutions using crowdsourcing can produce savings, but they cannot assess the accuracy of their heuristic, because they lack exact solutions.

3.2 Probabilistic Travelling Salesman Problem

When Jaillet [35] introduced the PTSP, the main focus of his work was on the case with homogeneous probabilities ($p_i = p$ for all i), while heterogeneous probabilities were first described by Berman and Simchi-Levi [13]. Jaillet [35] also introduced the *a priori* tour approach, which is the one most frequently considered in the literature. Reoptimisation is used to provide a bound on the quality of *a priori* tours, e.g., in [34, 19]. Early works on the PTSP examine theoretical properties of the *a priori* tour, including bounds and asymptotic relations [15], and put the PTSP in the broader context of probabilistic routing problems, where the parameters affected by uncertainty can also be travel times, demands, and time windows [31].

Solving the PTSP with exact methods has proven hard. More than twenty years ago, Laporte, Louveaux,

and Mercure [38] proposed a branch-and-cut algorithm which added L-shaped cuts to a stochastic formulation of the PTSP. They could solve instances with up to 50 customers, but only 5 of them had a non-zero probability of not requiring a visit ($p_i > 0$). Even more recent branch-and-bound algorithms were tested on instances of up to 10 customers [43], 18 customers [4], and 30 customers [3]. Furthermore, the difficulty of the PTSP increases (a) from homogeneous to heterogeneous probabilities [13], (b) when probabilities $1 - p_i$ of requiring a service are low [12], (c) when customers are not geographically uniformly distributed [20]. Recently, Lagos, Klapp, and Toriello [37] proposed a branch-and-price algorithm for a multi-vehicle version of the PTSP, which they tested on instances of up to 40 customers. To the best of our knowledge, these results are only published in draft form [37].

While exact methods are scarce, the literature abounds with heuristics. Devising effective heuristics, however, is still a hard task. For example, one of the main building blocks of heuristic and meta-heuristic algorithms is an efficient local search procedure. But since the evaluation of the PTSP objective function is $O(n^3)$, neighbourhood exploration becomes hard. In a survey, Campbell and Thomas [21] discuss two approximation approaches: truncating the number of terms [54], and aggregating customers [20]. Computation of the objective value can be also accelerated via *sample average approximation* (SAA). The method, introduced by Verweij et al. [55], estimates the expected value of a solution by averaging over a finite number of possible realisations of the Bernoulli random variables associated with the customers (an application of the *Monte Carlo* simulation technique). Birattari et al. [18] use SAA to estimate not the whole cost of a solution, but only the cost variation when moving from one solution to the next via local search. This strategy, called *delta evaluation*, was already used by Bianchi, Knowles, and Bowler [17] and by Bianchi and Campbell [16] for two neighbourhoods ("2-p-opt" and "1-shift") to provide a closed-formula expression for the expected gain of applying the local search move. The formulas were actually published, with mistakes, in [14, 15] and then corrected in [17, 16]. Another approximation approach, proposed by Meier and Clausen [47], consists in transforming the high-degree objective function into a quadratic function and then linearising the resulting problem. The authors show that the method produces nearly-optimal solution for instances up to 80 nodes, but using probabilities $p_i = 0.02, 0.05, 0.1$ (i.e., most customers require a service).

Authors have also applied meta-heuristic paradigms such as scatter search [40], Iterated Local Search [10], multi-started Local Search [39], Greedy Randomised Adaptive Search Procedure [46], Variable Neighbourhood Search [56], swarm optimisation [45], memetic algorithms [10], and ant colony optimisation [9].

3.3 Probabilistic TSP with profits

Zhang et al. [58] introduced the class of Travelling Salesman Problems with Profits and Stochastic Customers (TSPPSC). This class contains three problems, which mirror the three classical problems in the area of TSP with profits: the Orienteering Problem (OP) that maximises the collected profit under an upper bound on tour duration, the Prize-Collecting TSP (PCTSP) that minimises tour duration under a lower bound on collected profit, and the Profitable Tour Problem (PTP) described in Appendix 1.

For each of these problems, the authors describe the corresponding version with stochastic customers. Of the three new problems, the one more related to the PTSPC is the PTP with Stochastic Customers (PTPSC). In this problem the decision-maker has to select a subset of customers and plan an *a priori* tour only visiting the selected customers. When the the random outcomes are revealed, the planner amends the tour skipping the customers not requiring service. If a delivery is not even included in the *a priori* tour, the customer will not be visited and the corresponding profit will not be collected. This is analogous to outsourcing the delivery, paying a fee equal to the lost profit to a provider who is always available to accept requests. This highlights a sort of duality between the proposed problem and our PTSPC. In our problem we have certain customers but uncertain outsourcing; in the PTPSC there are uncertain customers, but certain outsourcing. Looking at probabilities, the decision-maker of the PTPSC has the possibility of setting $p_i = 1$ (exclude from the tour) for those customers he does not select in the *a priori* tour; in contrast, in our problem the decision-maker can set $p_i = 0$ (force in the tour) for those customers he does not offer for crowdsourcing.

Zhang, Wang, and Liu [57] propose a genetic algorithm to solve the PTPSC with homogeneous probabilities. Their analysis is limited to 5 instances with 8, 13, 20, 28 and 50 customers. It shows that, for the two largest instances, the genetic algorithm produces in a few seconds better results than solving the highly non-linear formulation with a commercial solver running for three hours.

Finally, we remark that of the three problems with profits mentioned above, there are no exact algorithms

for their stochastic-customer variants in the literature, while heuristics only exist for the PTPSC [57] and the OP with stochastic customers [58, 5, 48].

4 Algorithms

The most straightforward algorithm to solve optimisation problem (2) is complete enumeration. This requires 2^n outer iterations, one for each possible set $O \subseteq V$ of deliveries to offer, and $2^{|O|}$ inner iterations, one for each possible set $A \subseteq O$ of accepted deliveries. At each inner iteration we have to solve a Travelling Salesman Problem with deliveries $V \setminus A$. We get a speed-up for this last step noticing that sets $V \setminus A$ repeat over the iterations, so we can solve the TSPs once and cache the results. A first question arises: is this approach as untenable as it looks? Up to which instance size can we solve (2) by enumeration, on modern hardware? To answer this question, we generated a test bed of instances (as explained in Section 5.1) and ran a series of computational experiments whose results we report in Section 5.2.

A second question is how much we can scale results by speeding up the enumeration process at the cost of giving up optimality guarantees. We propose two approaches:

1. reduce the time spent evaluating each term $\mathbb{E}_A\big[C(O)\big]$;
2. reduce the number of terms $\mathbb{E}_A\big[C(O)\big]$ to evaluate.

To achieve the two objectives above, we devised heuristic algorithms whose results we report in Sections 5.3 and 5.4.

4.1 Speeding up the computation of $\mathbb{E}_A\big[C(O)\big]$

Computing $\mathbb{E}_A\big[C(O)\big]$ is expensive for large sets O: the number of sets $A \subseteq O$ to consider grows exponentially with the size of O and, for each such A, we have to solve a TSP. We directed our efforts towards reducing the number of sets A to consider in the sum in eq. (1). To do so, we used two techniques: Monte Carlo simulation, and Machine Learning.

4.1.1 Monte Carlo simulation

With this approach, at each computation of $\mathbb{E}_A\big[C(O)\big]$, instead of enumerating all sets $A \subseteq O$, we fix a much smaller collection $\mathcal{A} \subset \mathcal{P}(O)$ ($\mathcal{P}$ indicates the power-set) of fixed size. We compute the sum in eq. (1) over the sets of this smaller collection, and then rescale the result to reflect the large number of terms left out from the sum. This results in an approximation of $\mathbb{E}_A\big[C(O)\big]$, defined as follows:

$$\tilde{\mathbb{E}}^{\mathrm{MC}}_A\big[C(O)\big] = \frac{1}{p_{\mathcal{A}}} \sum_{A \in \mathcal{A}} \left[\left(\prod_{i \in A} p_i \prod_{i \in O \setminus A} (1 - p_i) \right) \cdot \left(\sum_{i \in A} m_i + c_{V \setminus A} \right) \right] \tag{3}$$

where $p_{\mathcal{A}}$ is the probability that a set of $\mathcal{A}$ is realised, i.e. $p_{\mathcal{A}} = \sum_{A \in \mathcal{A}} \prod_{i \in A} p_i \prod_{i \in O \setminus A} (1 - p_i)$. The size of $\mathcal{A}$, denoted as n_{MC}, is a parameter which allows to trade-off estimation accuracy and computing time. We used a fixed value of $n_{\mathrm{MC}} = 20$, although more complex approaches are possible: for example, Montemanni et al. [48] use a neural network to learn good values for n_{MC} when using Monte Carlo simulation for the Probabilistic Orienteering Problem.

4.1.2 Machine Learning approaches

Predicting the value of a complex objective function using machine learning techniques is a recent approach which is gaining popularity (ess, e.g., Fischetti and Fraccaro [28]). In our case, ecause computation of $\mathbb{E}_A\big[C(O)\big]$ is harder for large sets O, we can compute the exact value of the expected cost for small sets and then use this information to *predict* the expected cost of larger O's. This means training a regression model over data collected on small sets O and then using the model to extrapolate an estimate of $\mathbb{E}_A\big[C(O)\big]$ for large sets. Note how the training and the prediction happen on-line: we train a specific model for each instance, while we solve that instance.

In the following we define which features (i.e., independent variables) we use for regression and which models we tried. To list the features used, we first have to introduce some notation. Let $\bar{c}(W), \hat{c}(W), \underline{c}(W)$ be, respectively, the largest, the average, and the smallest distance of a delivery point in set $W \subseteq V$ from the depot:

$$\bar{c}(W) = \max_{i \in W} c_{0i}, \quad \hat{c}(W) = \frac{1}{|W|} \sum_{i \in W} c_{0i}, \quad \underline{c}(W) = \min_{i \in W} c_{0i}$$

Let $m(W)$ be the sum of crowdsourcing fees for deliveries in W, $m(W) = \sum_{i \in W} m_i$. Finally, let $d(W)$ be the diameter of W, $d(W) = \max_{i,j \in W} c_{ij}$. We define, then, the following features:

- One binary feature for each delivery point $i \in V$, with value 1 iff $i \in O$.
- One feature representing the fraction of crowdsourcing fees of offered deliveries, $m(O)/m(V)$.
- Three features, representing the ratios between largest, average, and smallest distances from the depot, between delivery points in O and all delivery points: $\frac{\bar{c}(O)}{\bar{c}(V)}$, $\frac{\hat{c}(O)}{\hat{c}(V)}$, $\frac{\underline{c}(O)}{\underline{c}(V)}$.
- Three features, similar to those above, but referring to $V \setminus O$: $\frac{\bar{c}(V \setminus O)}{\bar{c}(V)}$, $\frac{\hat{c}(V \setminus O)}{\hat{c}(V)}$, $\frac{\underline{c}(V \setminus O)}{\underline{c}(V)}$.
- Two features, representing the ratio between the diameters of, respectively, O and $V \setminus O$, and the diameter of V: $\frac{d(O)}{d(V)}$, $\frac{d(V \setminus O)}{d(V)}$.

In a preliminary analysis (see Section 5.3) we compare the following regression models:

- A linear regression model *LR*.
- A quadratic regression model with a LASSO regularisation term, *QRL*, leaving out the trivial quadratic terms relative to the binary features (since zero squared is zero, and one squared is one). We tuned the regularisation hyperparameter via 5-fold cross-validation using a grid search over 100 values.
- A multi-layer perceptron with ReLU activators, *MLP*. We chose the number of layers and size of each layer via 5-fold cross-validation using three options: one layer with 30 neurons, two layers with 15 neurons each, three layers with 10 neurons each.
- A single regression tree *RT*, and a regression forest *RF* trained with the Gradient Boosting algorithm [29]. We used the Mean Squared Error as the splitting criterion and, for the Gradient Boosting algorithm, we used 100 estimators of maximum depth 3.
- Because a regression tree is not able to extrapolate predicted values outside the range of the labels in the training set, we experimented training linear regression models on the leaf nodes of *RT*, thus obtaining a new model *RT+LR*. This model is analogous to the "M5" model introduced by Quinlan [51].

The independent variable used for training was the exact expected cost $\mathbb{E}_A\big[C(O)\big]$. Then, we used the trained model to yield predictions $\tilde{\mathbb{E}}_A^{\mathrm{ML}}\big[C(O)\big]$ relative to new sets O not seen during training.

4.2 Reducing the number of terms $\mathbb{E}_A\big[C(O)\big]$ to compute

To reduce the number of terms $\mathbb{E}_A\big[C(O)\big]$ to evaluate, we propose four heuristics which decrease the number of sets O considered.

- The *forward stepwise* heuristic starts with $O = \emptyset$ and iteratively adds the customer i for which $\mathbb{E}_A\big[C(O \cup \{i\})\big]$ is smallest, until no improvement is possible.
- The *backward stepwise* heuristic is analogous, but it starts with $O = V$ and removes the customer i for which $\mathbb{E}_A\big[C(O \setminus \{i\})\big]$ is smallest.
- The *alternating forward-backward* heuristic starts with $O = \emptyset$ and alternates one forward and one backward step. This method uses the same basic idea of the heuristic presented by Gdowska, Viana, and Pedroso [30]. Analogously, the *alternating backward-forward* heuristic starts with $O = V$ and alternates one backward and one forward step.

4.3 Bringing all together

Note how some of the techniques described in Sections 4.1 and 4.2 are *orthogonal*, in the sense that we can combine them independently. For example, we might choose to adopt a full enumeration of sets O but estimate each term $\mathbb{E}_A\big[C(O)\big]$ with a Monte Carlo simulation, or to use a forward stepwise heuristic and calculate the expected costs exactly. We expect Machine Learning algorithms to be more effective in combination with full enumeration, as it allows us to include more data in the training dataset (because the algorithm "sees" all sets O of small size that we use to train the model).

Note that when using an approximation of $\mathbb{E}_A\big[C(O)\big]$ to determine a set $\hat{O}$ of deliveries to offer for crowdsourcing, differently from classical heuristics, we don't know the true value of the objective function associated with $\hat{O}$. Recomputing the exact value of the expected cost, even for a single set $\hat{O}$, can be expensive if the set is large. A decision support system, thus, might return the set $\hat{O}$ and the approximate expected cost already computed. In our case, because our interest is in comparing different approaches, we added a further step at the end of the algorithms using approximate values: we compute the exact value of $\mathbb{E}_A\big[C(\hat{O})\big]$, to be able to gauge the accuracy of the approximation.

5 Computational results

In this section we report on the computational difficulty of this problem and on the quality of the heuristic techniques outlined in Appendix 4. We ran all experiments on a cluster with Xeon 2.4GHz processors. We wrote the code in C++. To solve the TSP problems, we used Concorde [6] via the Discorde API [44] and cached the results in a Patricia Trie [49]. We wrote the code dealing with the machine learning models in Python using the scikit-learn library [50], and we embedded it using the Boost Python library [1]. Both code and instances are available on-line [53].

5.1 Instances

We generated a test set selecting four base instances in the TSPLIB [52], truncating them to the first $n + 1$ vertices (one depot and n delivery points) for different values of n, and assigning values to p_i and m_i according to different criteria. Because we used six criteria, six values for n, and four base instances, our data-set counts with 144 instances. The number of delivery points used are $n \in \{15, 16, 17, 18, 19, 20\}$.

We used the following methods to assign values p_i and m_i, guided by two principles: using realistic criteria, and avoiding that all instances in a set have an "extreme" optimal solution ($O^{\text{opt}} = V$ or $O^{\text{opt}} = \emptyset$).

Instance Set A. Probability p_i is linearly proportional to the vertex's distance from the depot, with $p_i = 0$ for the depot itself and $p_i = 0.95$ for the farthest delivery point; crowdsourcing fee m_i is set to c_{0i}/n.

Instance Set B. As in set A, but we increase fees m_i by 25%. Increasing the fees is a straightforward way of obtaining instances in which the optimal set O^{opt} is smaller.

Instance Set C. As in set A, but we assigned probabilities with inverse proportionality to their distance from the depot. The rationale is that, in real applications, far delivery points might be inaccessible and harder to crowdsource.

Instance Set D. As in set A, but again probabilities are not assigned according to a direct proportion. Let L be the distance between the depot and the farthest delivery point. Delivery points whose distance from the depot is $0.25 \cdot L$ or smaller got assigned $p_i = 0.5$; otherwise if the distance is $0.75 \cdot L$ or smaller, they got $p_i = 0.75$; otherwise they got $p_i = 0.95$. This is a *step-wise* version of the criterion used in set A, motivated by the fact that we can group delivery points into "close", "distant", and "intermediate" groups.

Instance Set E. Combines elements of sets C and D. We assigned probabilities as in set D, but in reverse (e.g., close delivery points had 0.95). This also reflects the fact that centric deliveries are easier to crowdsource, while those in the suburbs can be harder.

	Set A			Set B			Set C			Set D			Set E			Set F		
$\|V\|$	FO	Sav	Time	FO	Sav	Time	FO	Sav	Time	FO	Sav	Time	FO	Sav	Time	FO	Sav	Time
15	1.00	23.27	132.13	0.87	20.79	131.86	0.60	11.85	131.64	0.87	37.51	130.95	0.78	23.10	130.69	0.67	18.31	133.71
16	0.83	23.46	299.31	0.80	21.15	307.46	0.59	11.71	304.16	0.88	37.79	302.90	0.80	22.69	304.79	0.66	16.42	305.02
17	0.84	22.33	721.50	0.79	20.11	727.20	0.63	12.01	888.71	0.90	36.82	719.54	0.97	21.46	718.64	0.65	13.48	721.30
18	0.85	21.84	1896.97	0.81	19.81	1881.88	0.65	12.12	1646.20	0.90	33.99	1752.09	0.97	23.91	1655.59	0.74	14.32	1913.27
19	0.86	22.09	4421.65	0.78	20.00	4447.92	0.68	12.07	4374.25	0.92	32.88	4292.36	0.96	23.31	4384.22	0.59	11.88	4316.65
20	0.86	21.30	7338.38	0.76	19.18	7508.06	0.68	12.03	7562.05	0.91	33.25	7495.46	0.74	22.16	7655.88	0.70	14.76	7587.90

Table 1: Full enumeration results. Columns FO report the fraction of vertices belonging to the optimal set O^{opt}; columns Sav list the expected savings (in percentage) when using crowdsourcing, compared to serving all deliveries with the own vehicle; columns Time list the runtime in seconds. Each row refers to the average over 4 base instances.

Instance Set F. In this case, we selected probabilities and fees uniformly at random. For probabilities the lower and upper bounds were 0 and 1, while for fees they were the shortest and largest travel cost between the depot and a delivery point.

We also checked that no delivery can be trivially fixed to always (or never) being offered. We used the following method which, in real-life instances, can be used to reduce the solution space. Consider a vertex $i \in V$ and let $j^*, k^* \in V'$ be the two vertices between which inserting i has the least cost

$$j^*, k^* = \arg\min_{j,k \in V'} \{c_{ji} + c_{ik}\}$$

If the crowdsourcing fee is smaller than the smallest insertion cost, $m_i < c_{j^*i} + c_{ik^*}$, then it will always be convenient to offer i for crowdsourcing. Analogously, if the crowdsourcing fee is larger than the largest insertion cost (obtained swapping arg min with arg max) it will always be convenient to visit i with the own vehicle.

5.2 Enumeration results

We ran the full enumeration algorithm for three reasons. First, to check that the runtime grows exponentially with the instance size, as we assumed in Appendix 2. Second, to compute exact solutions which we can use to assess the quality of the heuristic algorithms. Third, to see how the optimal sets O^{opt} vary depending on the instance sets.

Table 1 gives a summary of the results. Columns "FO" report the fraction of vertices in the optimal offered set ($|O^{\text{opt}}|/|V|$). Columns "Sav" list the percentage savings obtained when using crowdsourcing, i.e. the gap between the expected cost of the optimal solution and the cost of a TSP tour visiting all customers:

$$\text{Sav} = \frac{c_V - \mathbb{E}_A\left[C(O^{\text{opt}})\right]}{\mathbb{E}_A\left[C(O^{\text{opt}})\right]} \cdot 100$$

Columns "Time" list the runtime in seconds.

The savings reported in Table 1 for all instance sets show that, no matter the method used to generate the instances, the adoption of crowdsourcing can lead to significant savings in the final expected cost. The results show that instances with step-wise probability of acceptance produce larger savings (compare sets A with D, and C with E). The important factor is that, in step-wise instances, the probability of acceptance for distant delivery points increases, showing that it's important for companies to incentivise crowdsourcing of the farthest deliveries with appropriate pricing strategies. Comparing sets A and B shows that such pricing strategies are feasible, as increasing the rewards by 25% gives only minor decreases in over-all expected savings.

We provide a visualisation of the algorithm runtime in Figure 1, where each point corresponds to one of the 144 instances. The x axis reports the instance size, with points of different instance sets slightly shifted from each other. The y axis represents the runtime, and is in logarithmic scale. Figure 1 shows that the runtime grows exponentially in the size of V, as an upward straight line in the figure corresponds to exponential growth along the y axis.

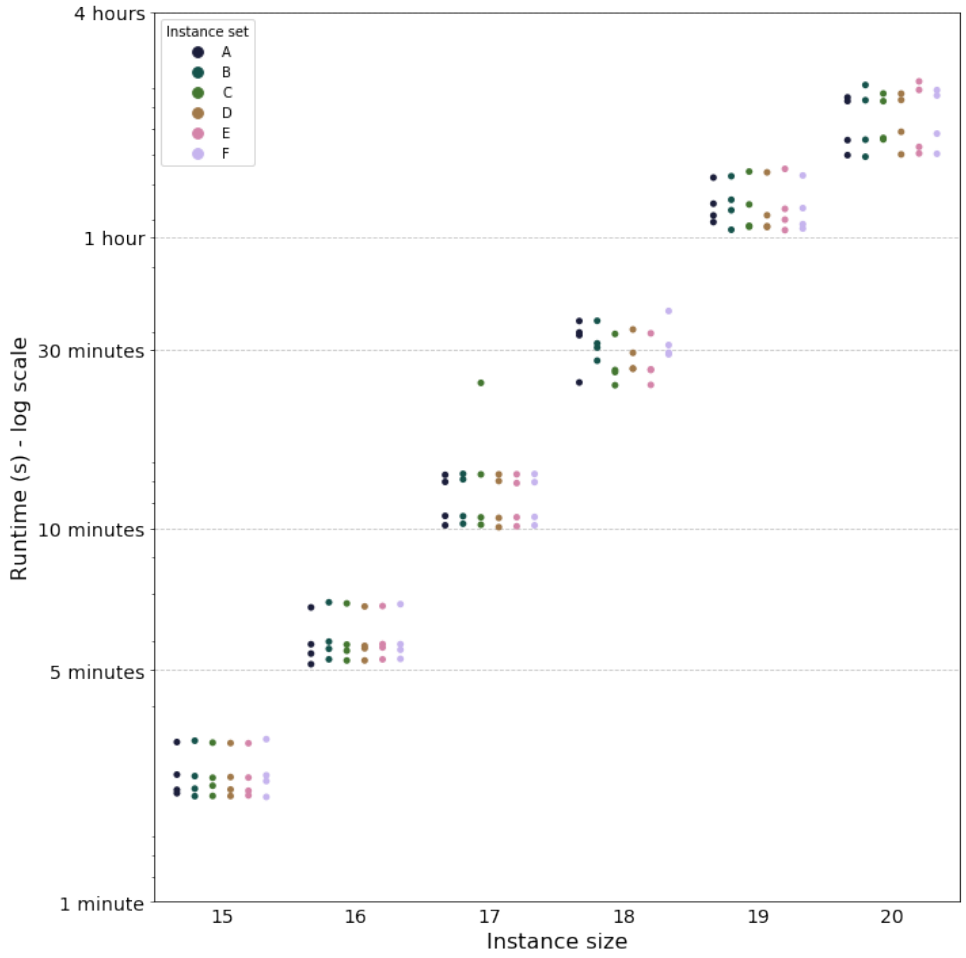

Figure 1: Runtime for the full enumeration algorithm. The y-axis is in logarithmic scale.

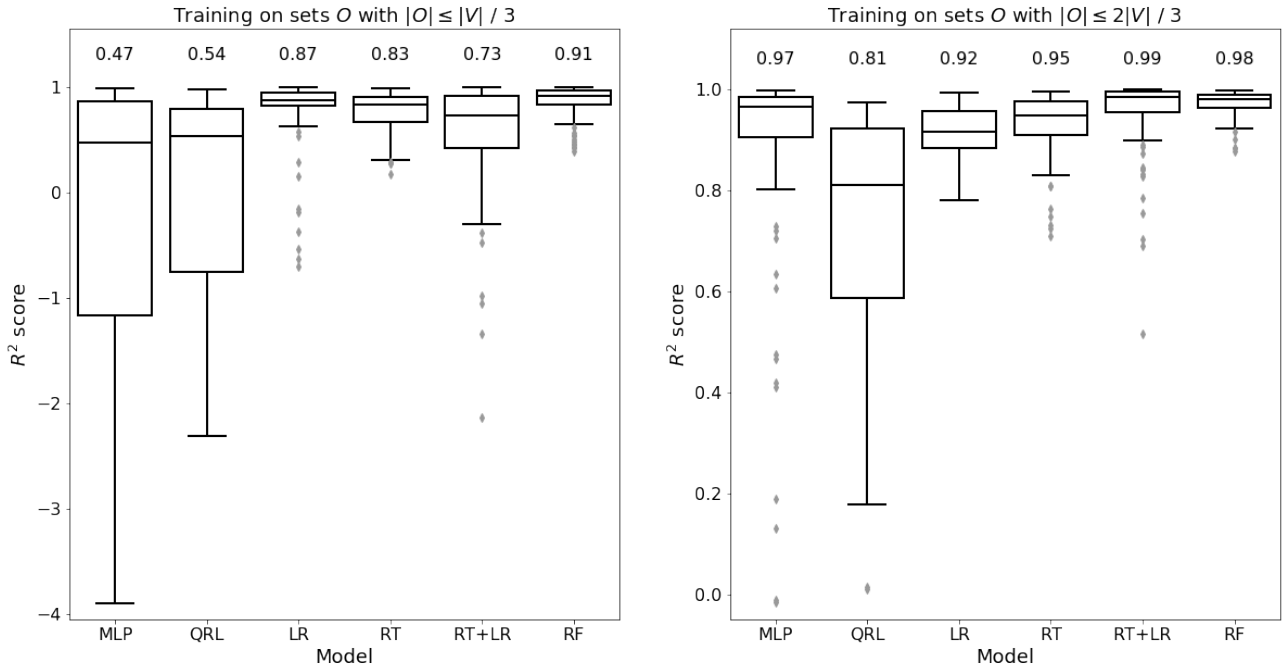

Figure 2: R^2 score of the Machine Learning models used off-line. The boxes denote instances within the two central quartiles and whiskers extend to the rest of the distribution, except for outliers denoted by diamond filers. The solid line inside the boxes and the numbers at the top refer to the median score.

5.3 Evaluation of Machine Learning and Monte Carlo estimators

We ran two sets of experiments to assess the quality of the Monte Carlo and Machine Learning estimators of $\mathbb{E}_A\big[C(O)\big]$. In the first set, we measure the accuracy of the estimation *off-line*, i.e., without bundling the estimators in an enumeration algorithm. The aim of these experiments is to understand how much the estimates $\tilde{\mathbb{E}}_A^{\mathrm{ML}}\big[C(O)\big]$ (Machine Learning) and $\tilde{\mathbb{E}}_A^{\mathrm{MC}}\big[C(O)\big]$ (Monte Carlo) differ from the exact value $\mathbb{E}_A\big[C(O)\big]$. In the second set, we use the estimators *on-line* within the enumeration algorithm. In this case, we are not only interested in the accuracy of the estimation, but also in the trade-off between speed and accuracy.

5.3.1 Estimations using Machine Learning

For the first set of experiments, we trained the machine learning models on an off-line dataset. We created this dataset by running the complete enumeration algorithm over all instances and, each time we evaluated a term $\mathbb{E}_A\big[C(O)\big]$, we also recorded the corresponding independent variables mentioned in Section 4.1.2.

Next, to reflect how we would use the models bundling them into the enumeration algorithm, we split the dataset in two parts. The training dataset consists of data relative to small sets O, while the test dataset contains data points relative to larger sets O. We experimented with two different splits: in the first case we put in the training dataset information for sets such that $|O| \leq \frac{1}{3}|V|$, and in the second case for sets such that $|O| \leq \frac{2}{3}|V|$. This means that, e.g., in the first case we would run the exact enumeration algorithm for sets O containing up to one third of the delivery points and then use the machine learning models to predict the expected cost for larger sets. These cases reflect two different trade-offs between the size of the training datasets and the time required to generate them (i.e., the time spent calculating the exact expected costs).

Figures 2 to 4 report the findings of this first set of experiments. In each figure the left chart refers to the first case ($|O| \leq \frac{1}{3}|V|$) and the right chart to the second case ($|O| \leq \frac{2}{3}|V|$). Figure 2 reports the R^2 scores obtained by each of the models presented in Section 4.1.2. The boxes represent the two central quartiles of scores across all the instances; the whiskers extend to the rest of the scores distribution, except for outliers which are denoted by diamond fliers. The solid line inside each box demarcates the median score, which is also reported as a number on the top part of the chart. As we can expect, scores improve with the larger training dataset size. The median R^2 values are high, considering we are trying to predict the complicated function $\mathbb{E}_A\big[C(O)\big]$ but, for smaller training set, they are sometimes negative (i.e., using the model is worse

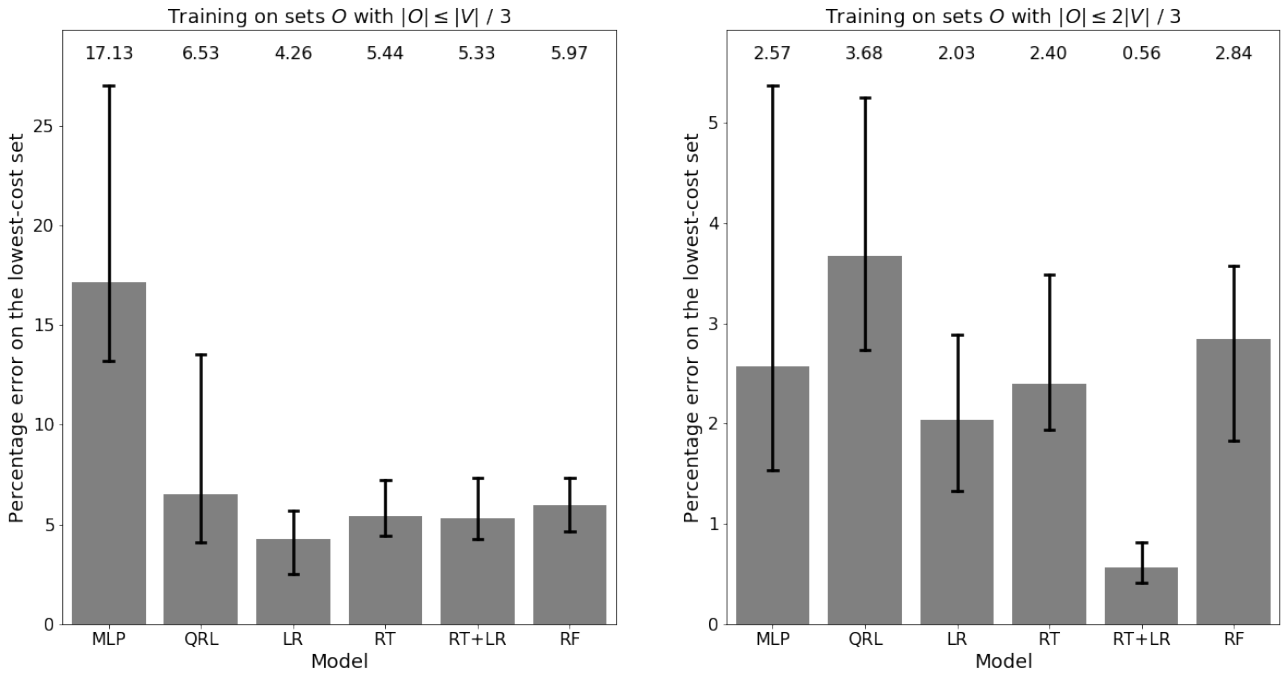

Figure 3: Percentage error on the estimate of the expected cost of the lowest-cost set in the test dataset, when using the Machine Learning models off-line. The height of the bar and the numbers at the top report the median error. The whiskers show a 95% confidence interval obtained by bootstrapping.

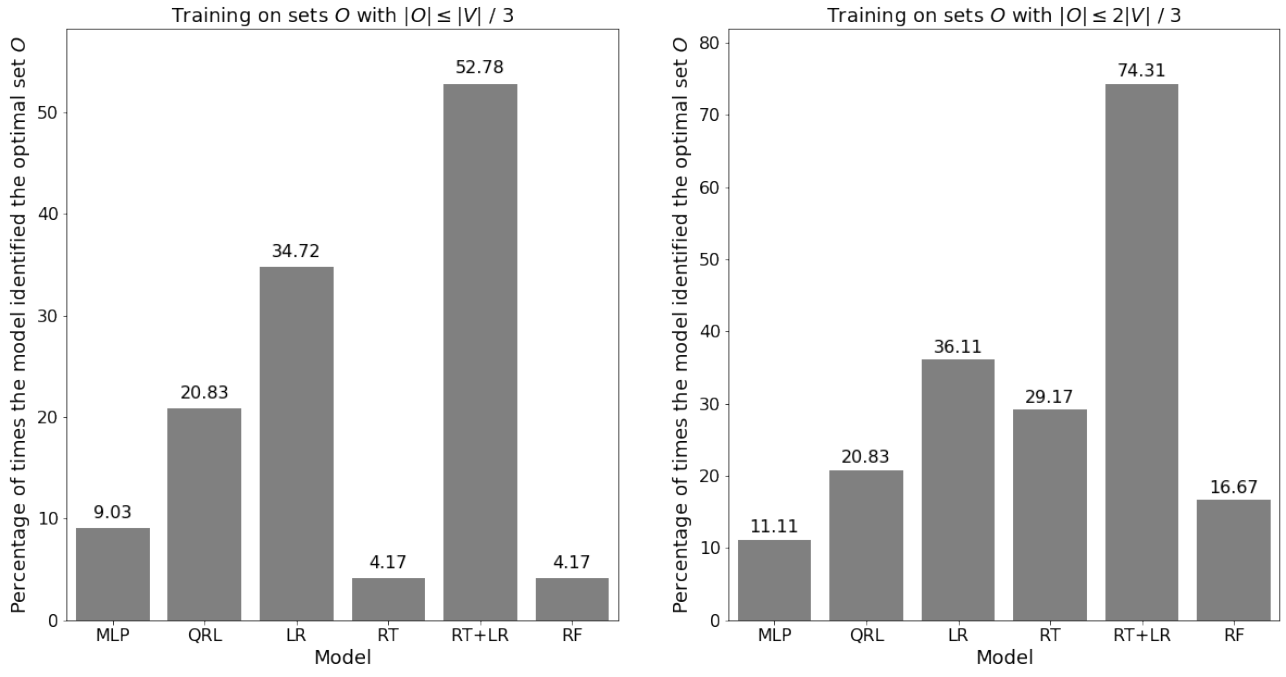

Figure 4: Number of instances (in percentage) for which each Machine Learning model, used off-line, identified the lowest-cost set in the test dataset.

Instance size	Avg Time (s)	Avg Gap (%)	Opt found (%)
15	62.34	4.61	62.50
16	68.71	8.72	8.33
17	311.55	6.05	33.30
18	332.22	7.90	8.33
19	373.39	7.84	12.50
20	479.44	8.56	12.50

Table 2: Results for the Machine Learning model *RT+LR* used on-line whithin the full enumeration algorithm. Each row presents results averaged over all instances of the given size. Column Avg Gap (%) lists the percentage gaps between the real expected cost $\mathbb{E}_A\big[C(\hat{O})\big]$ of the best set $\hat{O}$ found by the algorithm and the optimum. Column Opt found (%) reports the number of instances (in percentage) for which the algorithm identified the optimal set.

than predicting the mean). We attribute the high median scores to a careful selection of features and to the fact that we heavily fit to the distribution associated with the instance we train the model on. Fitting to a particular instance, unlike *overfitting* to a particular training set, is not a problem because we train each model from scratch for each instance and don't plan to reuse the same trained model for new instances.

The figure shows that, for the smaller training dataset, Gradient Boosted trees (model *RF*) perform the best, followed by Linear Regression (model *LR*). For the larger training dataset, the Regression Trees with Linear Regression models on their leaves (model *RT+LR*) achieves an almost perfect R^2 score.

Figure 3 reports the percentage error made by each model on the lowest-cost set O^* in the test dataset $\mathcal{T} \subset \mathcal{P}(V)$. Note that we know this set because we have computed all expected costs with the complete enumeration algorithm; also, if the optimal set happens to be in the test set, then $O^* = O^{\text{opt}}$. We define the percentage error on O^* as

$$\frac{\left|\mathbb{E}_A\big[C(O^*)\big] - \tilde{\mathbb{E}}_A^{\text{ML}}\big[C(O^*)\big]\right|}{\mathbb{E}_A\big[C(O^*)\big]}$$

This metric is important because identifying the lowest-cost set and estimating its cost is the main job of an estimator. The figure reports the median percentage error as the height of the bars; these numbers are also present at the top of each chart. The whiskers denote 95% confidence intervals around the median, obtained via bootstrapping. Models *LR* and *RT+LR* give the best performances, respectively, on the smaller and larger training datasets (although the confidence interval for *LR* on the smaller datasets overlaps with other intervals, including the one of *RT+LR*).

Because we want to find the set $O^* \in \mathcal{T}$ which minimises $\mathbb{E}_A\big[C(O)\big]$, another important metric that defines a good model is to count how many times it managed to identify this set in the test dataset. A model is successful in doing so if

$$\hat{O} \coloneqq \arg\min_{O \in \mathcal{T}} \tilde{\mathbb{E}}_A^{\text{ML}}\big[C(O)\big] = \arg\min_{O \in \mathcal{T}} \mathbb{E}_A\big[C(O)\big] =: O^* \tag{4}$$

Note how finding the correct set O^* doesn't necessarily say anything on the quality of the other metrics. For example, a model consistently giving an estimate $\tilde{\mathbb{E}}_A^{\text{ML}}\big[C(O)\big] = \frac{1}{1000}\mathbb{E}_A\big[C(O)\big]$ would perhaps have a low R^2 score and a high percentage error, but would always be able to identify O^*. Figure 4 reports the number of instances (in percentage) for which each model succeeded at satisfying Equation (4). In this case, model *RT+LR* is, by far, the best model in both cases and thus it is a good candidate to use in combination with the enumeration algorithm.

In the second set of experiments, we focused on the *RT+LR* model and we used it within the full enumeration of sets $O \subseteq V$. Because in a decision support system the user will require a solution within a short timeframe, we used a hard time limit as the stopping criterion to halt the construction of the training set. For instances of sizes 15 and 16 we used a 1-minute time limit, while for other instances we used a 5-minute limit. After this time limit we trained the model (the *RT+LR* model trains in less than 5 seconds) and then we started using it to approximate $\mathbb{E}_A\big[C(O)\big]$. Note that this strategy is analogous to the one used in the off-line experiments, because in Equation (2) we visit sets O by ascending size.

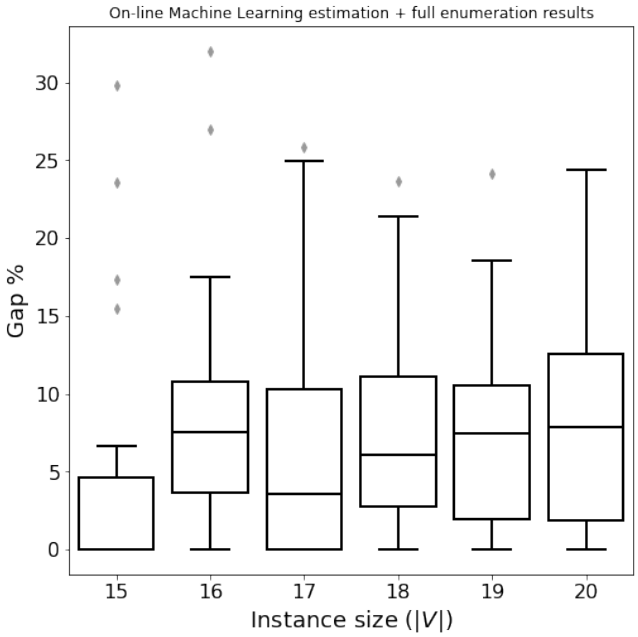

Figure 5: Percentage gaps obtained using the on-line Machine Learning estimator *RT+LR*. Each box refers to a different instance size. The box spans the two central quartiles of the gap distributions, with the straight horizontal line demarcating the median. The whiskers extend to the rest of the distribution, except for outliers drawn as diamond fliers.

Table 2 reports the results of applying the Machine Learning *RT+LR* estimator on-line during the full enumeration outer loop. Each row presents results averaged over all instances of the given size. Column "Avg Time (s)" reports the solution time including: (1) running the enumeration algorithm and computing the exact expected cost to build the training set, (2) training the Machine Learning model, and (3) completing the enumeration using the model's predictions. Column "Avg Gap (%)" lists the percentage gaps between the expected cost $\mathbb{E}_A\big[C(\hat{O})\big]$ of the set $\hat{O}$ returned by the algorithm, and the expected cost $\mathbb{E}_A\big[C(O^{\text{opt}})\big]$ of the optimal set O^{opt}. Note that the algorithm returns $\tilde{\mathbb{E}}_A^{\text{ML}}\big[C(\hat{O})\big]$, so we computed $\mathbb{E}_A\big[C(\hat{O})\big]$ in a further step, after the experiment was over; the time used to compute $\mathbb{E}_A\big[C(\hat{O})\big]$ is not included in the time reported under "Avg Time (s)". Column "Opt found (%)" reports the number of instances (in percentage) for which the algorithm identified the optimal set (i.e., $\hat{O} = O^{\text{opt}}$).

Note that most of the time is spent building the training set, and the total runtime slightly exceeds the training time (60 seconds for sizes 15 and 16, 300 seconds for larger instances). All average gaps are under 10% and there is no clear trend as a function of the instance size. Even when the algorithm is not successful at identifying the optimal set, it still gives a solution of high quality, as testified by the low gaps. Figure 5 shows the full distribution of gaps as a function of the instance size. All median values are under 10% and the third quartile is always well under 15%. The almost flat distribution of gaps as instance size increases suggests that this method can scale well for larger instances.

5.3.2 Estimations using Monte Carlo simulation

Analogously to what we have done for the Machine Learning estimator, we report results on two sets of experiments. With the first set, we analyse the accuracy of Monte Carlo estimates $\tilde{\mathbb{E}}_A^{\text{MC}}\big[C(O)\big]$ when the estimator is not yet bundled into the enumeration algorithm (off-line). The second set will show the performance and the quality of the Monte Carlo estimator when used inside a decision support system (on-line).

Table 3 reports the results obtained in the first set of experiments. For every instance and every set $O \subseteq V$, we compare the true expected cost $\mathbb{E}_A\big[C(O)\big]$ with its approximation $\tilde{\mathbb{E}}_A^{\text{MC}}\big[C(O)\big]$. Note that we left out sets O with $|O| \leq 4$ because, in that case, $2^{|O|} \leq n_{\text{MC}} = 20$ and so the number of subsets $A \subseteq O$ is smaller than the number of Monte Carlo iterations. Each row of the table displays results averaged over all instances with the given size (column "Instance size"). Column "Mean Err (%)" reports the percentage error (or *gap*) over all sets O and all instances, whereas column "Opt Err (%)" refers to the percentage error on the optimal sets O^{opt}.

Instance size	Mean Err (%)	Opt Err (%)	Opt found (%)
15	1.58	7.33	12.5
16	1.61	7.79	0.0
17	1.58	11.04	16.7
18	1.50	7.20	4.2
19	1.47	7.81	8.3
20	1.46	8.67	12.5

Table 3: Statistics on off-line Monte Carlo approximation with $n_{\text{MC}} = 20$ samples. Column Mean Err (%) reports the percentage error averaged over all sets O in all instances with the given size. Column Opt Err (%) reports the percentage error for the optimal set, averaged over all instances with the given size. Column Opt found (%) lists the number of instances, in percentage, for which the set with the lowest approximate cost coincides with the optimal set.

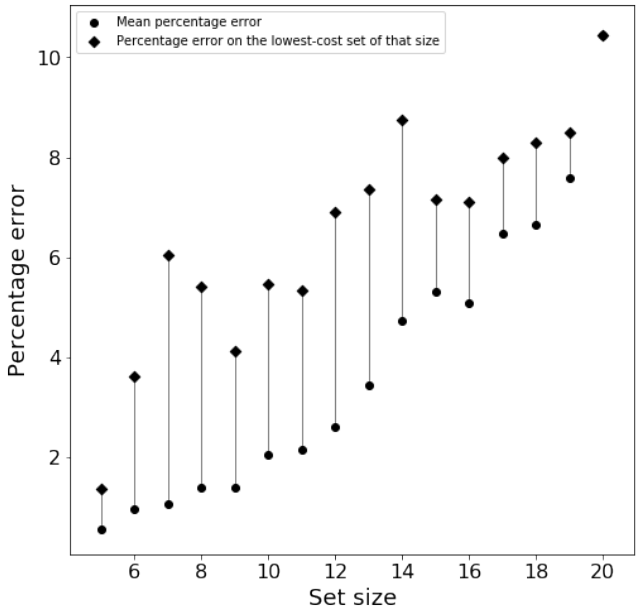

Figure 6: Percentage errors by size of the set O whose cost was approximated, for the off-line Monte Carlo estimator. Points with a round marker report the mean percentage error over all sets of the given size and over all instances. Points with a diamond marker display the percentage error over the lowest-cost set of the given size for each instance, averaged over all instances.

More formally, for each instance the mean percentage error is

$$\frac{1}{|\mathcal{P}_4(V)|} \sum_{O \in \mathcal{P}_4(V)} \frac{\left|\mathbb{E}_A\big[C(O)\big] - \tilde{\mathbb{E}}_A^{\text{MC}}\big[C(O)\big]\right|}{\mathbb{E}_A\big[C(O)\big]} \quad \text{where } \mathcal{P}_4(V) = \{O \subseteq V \ : \ |O| > 4\}$$

and the percentage error on the optimal set is

$$\frac{\left|\mathbb{E}_A\big[C(O^*)\big] - \tilde{\mathbb{E}}_A^{\text{MC}}\big[C(O^*)\big]\right|}{\mathbb{E}_A\big[C(O^*)\big]} \quad \text{where } O^* = \operatorname*{arg\,min}_{O \in \mathcal{P}_4(V)} \mathbb{E}_A\big[C(O)\big]$$

Column "Opt Found (%)" reports the number of instances (in percentage) in which the set $\hat{O}$ with the lowest approximate cost coincides with the optimal set:

$$\hat{O} := \operatorname*{arg\,min}_{O \in \mathcal{P}_4(V)} \tilde{\mathbb{E}}_A^{\text{MC}}\big[C(O)\big] = \operatorname*{arg\,min}_{O \in \mathcal{P}_4(V)} \mathbb{E}_A\big[C(O)\big] =: O^* \tag{5}$$

Comparing the values in column "Opt Found (%)" of Table 3 with those reported in Figure 4 we can see how the Machine Learning estimator is considerably better at finding the lowest-cost set, compared with Monte Carlo simulation.

Instance size	Avg Time (s)	Avg Gap (%)	Opt found (%)
15	107.48	4.41	0
16	326.11	4.66	0
17	418.01	5.14	0
18	514.99	3.66	0
19	1024.80	3.01	4.17
20	2742.19	3.39	0

Table 4: Results for the on-line Monte Carlo simulation method. Each row presents results averaged over all instances of the given size. Column Avg Gap (%) lists the percentage gaps between the real expected cost $\mathbb{E}_A\left[C(\tilde{O})\right]$ of the best set $\tilde{O}$ found by the algorithm and the optimum. Column Opt found (%) reports the number of instances (in percentage) for which the algorithm identified the optimal set.

We also notice a sharp increase in the percentage error when computing its average over all sets O (column "Mean Err (%)") and over the optimal sets (column "Opt Err (%)"). Because the accuracy of the Monte Carlo method decreases with $|O|$ (for larger sets n_{MC} becomes smaller compared to $2^{|O|}$) we were unsure if the lower precision was due to the optimal sets O^{opt} being generally large. To answer this question, in Figure 6 we plot the percentage error as a function of the size of O. Points with a round marker report the mean percentage error over all sets of the given size, averaged over all instances. Points with a diamond marker display the percentage error over the lowest-cost set of the given size for each instance, averaged over all instances. The diamond points always correspond to larger errors compared to the corresponding round points — except for size 20, because there is only one set O with such size, i.e., when $O = V$ in instances where $|V| = 20$, and so the two markers correspond to the same results. This means that, independently of the size of sets O (and so of the size of O^{opt}), the estimate is worse for *good* sets than it is on average. This consideration suggests that, in the on-line experiments, we could see fewer instances in which Monte Carlo simulation can find the optimum, compared to using the Machine Learning estimator.

The results of the on-line experiments, indeed, confirm this hypothesis. Table 4 reports these results using the same column headings as Table 2. The data shows three trends. First, the number of times the heuristic produced the optimal solution is larger using the *RT+LR* model compared to Monte Carlo simulation. Second, the Monte Carlo approach gives lower gaps than the Machine Learning model. The two estimation methods show, in fact, complementary strengths: one is better at identifying the optimum but, when it fails doing so, its error can be large; the other rarely identifies the optimal set but returns solutions which, on average, are better. The third consideration is that the Monte Carlo approach doesn't scale well and its runtime is high. Indeed, using the Machine Learning model the user has more control over the runtime because the training phase can be stopped at any time. We expect that, if we trained the *RT+LR* model for the same amount of time required by the solver using Monte Carlo simulation, the solution quality would increase considerably.

Given the above considerations, we think that the Machine Learning approach is a better alternative; the main advantage of using Monte Carlo simulation lies in its simplicity and minimal implementation effort.

5.4 Heuristic enumeration results

In this section we present the results obtained when reducing the number of sets $O \subseteq V$ we consider in the outer loop described in Section 2.2 to solve minimisation problem Equation (2). We applied the four heuristics described in Section 4.2: *forward stepwise* (Fwd), *backward stepwise* (Bck), *forward-backward stepwise* (Fwd-bck) and *backward-forward stepwise* (Bck-fwd). In all cases we computed the exact expected cost $\mathbb{E}_A\left[C(O)\right]$. We compare these results with those of the other three methods introduced in this section, which use the full enumeration approach with, respectively, the exact value and the Machine Learning and Monte Carlo cost estimators.

Table 5 gives a summary of the results. Column "Alg" denotes the algorithm used for the outer loop. Column "Est" indicates the estimator of the expected cost, with ML denoting Machine Learning and MC denoting Monte Carlo. For each instance size, we report three metrics: the optimality gap in percent (column "Gap"); the runtime in seconds (column "Time"); and the number of instances solved to optimality, in percent (column "Opt"). The numbers in bold mark the best result for each metric and instance size.

		Instance size																	
		15			16			17			18			19			20		
Alg	**Est**	**Gap**	**Time**	**Opt**	**Gap**	**Time**	**Opt**	**Gap**	**Time**	**Opt**	**Gap**	**Time**	**Opt**	**Gap**	**Time**	**Opt**	**Gap**	**Time**	**Opt**
Full	Exact	—	131.83	—	—	303.94	—	—	749.48	—	—	1791.00	—	—	4372.84	—	—	7524.62	—
Full	ML	4.61	62.34	62.50	8.72	**68.71**	8.33	6.05	311.55	33.30	7.90	332.22	8.33	7.84	**373.39**	12.50	8.56	**479.44**	12.50
Full	MC	4.41	107.48	0.00	4.66	326.11	0.00	5.14	418.01	0.00	3.66	514.99	0.00	3.01	1024.80	4.17	3.39	2742.19	0.00
Fwd	Exact	1.72	**53.55**	45.83	1.71	84.21	37.50	1.76	205.63	33.33	1.97	337.51	37.50	1.48	446.94	41.67	1.26	1484.07	41.67
Bck	Exact	**0.17**	149.70	**75.00**	**0.34**	303.39	**70.83**	**0.21**	733.66	**75.00**	**0.15**	1589.72	**79.17**	**0.14**	2917.80	**83.33**	**0.25**	6439.35	**75.00**
Fwd-bck	Exact	1.69	55.34	45.83	1.66	86.57	37.50	1.71	**187.18**	33.33	1.94	**329.43**	37.50	1.47	604.57	45.83	1.25	1690.15	41.67
Bck-fwd	Exact	**0.17**	157.19	**75.00**	**0.34**	316.54	**70.83**	0.31	749.56	71.43	**0.15**	1569.04	**79.17**	**0.14**	3381.74	**83.33**	**0.25**	7633.81	**75.00**

Table 5: Summary of the results for the exact and heuristic methods. Columns Alg and Est list the algorithm used to loop through sets $O \subseteq V$ and to calculate the expected costs, respectively. For each value of the instance size, column Gap reports the optimality gap in percent, column Time lists the runtime in seconds, and column Opt reports the number of instances solved to optimality, in percent.

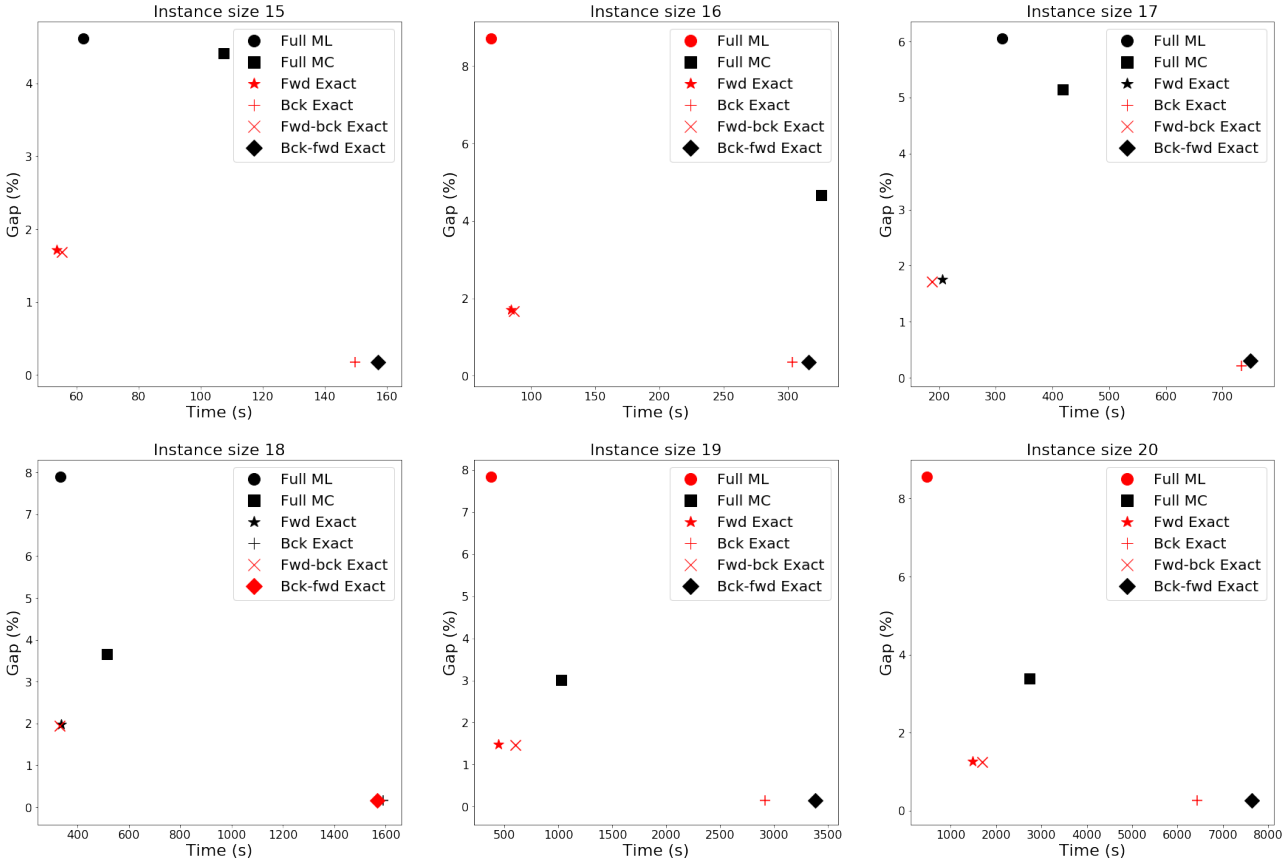

Figure 7: Optimality gaps and runtime for each heuristic algorithm. Each chart corresponds to a given instance size. The x axes reports the runtime in seconds, while the y axes represent percentage optimality gaps. Each algorithm is identified by a different marker; a lighter colour indicates an algorithm with Pareto-optimal performance.

Because the optimal sets O^* tend to be large, heuristic *Bck* gives lower gaps than *Fwd*. *Bck* is also more time-consuming because evaluating the expected cost for large sets O is harder than for small sets. Heuristics *Bck* and *Bck-fwd* tend to give both the lowest gaps and highest number of optima. The drawback is that they are basically as expensive as the full enumeration algorithm. This is because the evaluation of $\mathbb{E}_A\left[C(O)\right]$ for $O = V$, at which the backward algorithms start, takes exponentially more time than the evaluations for, e.g., sets with $|O| = |V| - 1$. Sometimes the average times are even higher for *Bck-fwd* than for full enumeration, due to variability in runtime.

To better assess the relative performance of these methods, then, we plotted the gap against the runtime in Figure 7. Each chart corresponds to an instance size and each point to an algorithm. The x coordinate represents runtime in seconds and the y coordinate is the average percentage gap. Points in lighter colour refer to a Pareto-optimal algorithm for the given instance size, i.e., there is no other algorithm which achieves both lower gap and shorter runtime simultaneously.

Note that the full enumeration algorithm with the Machine Learning estimator is Pareto-optimal in three cases, *Bck* in five cases and *Bck-fwd* in one. It is, perhaps, surprising that *Fwd* and *Fwd-bck* are also often Pareto-optimal (four and six times, respectively) mainly due to their fast runtime. Their gaps manage to be low because they conduct a thorough search over sets O of small size. The algorithm using Monte Carlo simulation is always dominated by some other method.

For instances of the size considered, the "forward" heuristics (*Fwd* and *Fwd-bck*) perform well and can run within a time compatible with their use in a decision support system. For larger instances, the method incorporating the Machine Learning estimator looks more attractive because one can tune its runtime by allowing larger or smaller training sets. Remark that all methods we devised gave gaps well under 10% and are thus attractive, given the complexity of the problem considered.

6 Conclusions and future work

In this paper we have introduced a stochastic generalisation of the Travelling Salesman Problem with important applications to crowdsourcing in last-mile delivery. In particular, we focused on retailers who want to crowdsource the delivery of some products to their own in-store customers. The problem is unconstrained in the sense that any subset of deliveries offered for crowdsourcing corresponds to a valid solution, but its objective function is complex and expensive to evaluate. To address this issue, we developed heuristic algorithms that reduce the number of evaluations by either considering fewer candidate solutions or by predicting their cost in constant time. Computational results on a varied set of instances show that even heuristic solutions can result in significant savings compared to the case of no crowdsourcing.

The model assumed that the probability of succesfully crowdsourcing a delivery and the corresponding fee paid to the courier-customer are fixed and known. In the future, we would like to consider the case in which the amount paid influences the probability of acceptance. For example, one could consider an agent-based approach in which ech customer who visits the store has an intended destination, and accepts or refuses a delivery based on their price sensitivity, the prize offered and the rerouting needed to visit the delivery site.

Acknowledgements

Alberto Santini was partially supported by grant "RTI2018-095197-B-I00" from the Spanish Ministry of Economy and Competitiveness. The other authors are funded by the ERDF European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project "POCI-01-0145-FEDER-028611".

A A priori tour approach

When the set of delivery points is stable across different planning periods, it might be more convenient for operational reasons to devise a fixed tour. At the end of each day, after the planner knows the set of accepted requests, he can amend the tour removing delivery points which do not require a visit. Such an approach has

benefits which are not easy to quantify: for example, because tours on successive days are similar, the driver's route knowledge increases.

Let us assume, wlog, that the *a priori* tour is $T = (0, 1, \ldots, n, n+1)$ where we denote the depot both as 0 at the beginning, and as $n+1$ at the end of the tour. We use the convention $p_0 = p_{n+1} = 0$, and $p_i = 0$ for all deliveries $i \in V \setminus O$ that the planner did not offer for crowdsourcing. The owned vehicle uses edge $\{i, i+1\}$ if the planner did not crowdsource neither i nor $i+1$ (either because they were not offered, or because no provider accepted the offer). The vehicle uses edge $\{i, j\}$ for $j > i+1$ if the planner did not crowdsource neither i nor j, but crowdsourced all deliveries between them. From these considerations the probability that the vehicle uses edge $\{i, j\}$ is

$$p_{\{i,j\}} = (1 - p_i)(1 - p_j) \prod_{k=i+1}^{j-1} p_k$$

which is valid for $i = 0, \ldots, n$ and $j = i+1, \ldots, n+1$. Denoting as $C_{\mathrm{t}}(O, T)$ the cost of the tour obtained amending the *a priori* tour T when the probabilities are realised, the expression for its expected value is

$$\mathbb{E}_A\big[C_{\mathrm{t}}(O,T)\big] = \sum_{i=0}^{n} \sum_{j=i+1}^{n+1} \left[p_{\{i,j\}} \cdot \left(c_{ij} + \sum_{k=i+1}^{j-1} m_k \right) \right]$$

In this case, the expected cost of offering a set of deliveries O for crowdsourcing corresponds to the expected cost $\mathbb{E}_A\big[C_{\mathrm{t}}(O,T)\big]$ when selecting the best possible *a priori* tour:

$$\mathbb{E}_A\big[C(O)\big] = \min \left\{ \mathbb{E}_A\big[C_{\mathrm{t}}(O,T)\big] \ : \ T \text{ is a tour in } G \right\} \tag{6}$$

We can then write the complete optimisation problem as

$$O^{\mathrm{opt}} = \operatorname*{arg\,min}_{O \subseteq V} \ \mathbb{E}_A\big[C(O)\big] \tag{7}$$

The difficulty in solving (7) comes both from the exponential number of sets $O \subseteq V$ and from having to solve a Probabilistic TSP (6) with the *a priori* tour method, to get the cost associated to each set O.

References

[1] David Abrahams and Stefan Seefeld. *Boost Python*. 2019. URL: https://www.boost.org/doc/libs/1_71_0/libs/python/doc/html/index.html.

[2] Aliaa Alnaggar, Fatma Gzara, and James Bookbinder. "Crowdsourced delivery: A review of platforms and academic literature". In: *Omega* (2019), pp. 102–139.

[3] Mohamed Abdellahi Amar, Walid Khaznaji, and Monia Bellalouna. "A Parallel Branch and Bound Algorithm for the Probabilistic TSP". In: *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2018, pp. 437–448.

[4] Mohamed Abdellahi Amar, Walid Khaznaji, and Monia Bellalouna. "An exact resolution for the probabilistic traveling salesman problem under the a priori strategy". In: *Procedia Computer Science* 108 (2017), pp. 1414–1423.

[5] Enrico Angelelli et al. "The probabilistic orienteering problem". In: *Computers & Operations Research* 81 (2017), pp. 269–281.

[6] David Applegate et al. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[7] Claudia Archetti, Martin Savelsbergh, and Maria Grazia Speranza. "The vehicle routing problem with occasional drivers". In: *European Journal of Operational Research* 254.2 (2016), pp. 472–480.

[8] Alp Arslan et al. "Crowdsourced Delivery – A Dynamic Pickup and Delivery Problem with Ad Hoc Drivers". In: *Transportation Science* 53.1 (2019), pp. 222–235.

[9] Prasanna Balaprakash et al. "Estimation-based ant colony optimization and local search for the probabilistic traveling salesman problem". In: *Swarm Intelligence* 3.3 (2009), pp. 223–242.

[10] Prasanna Balaprakash et al. “Estimation-based metaheuristics for the probabilistic traveling salesman problem”. In: *Computers & Operations Research* 37.11 (2010), pp. 1939–1951.

[11] Miguel Barbosa. “A data-driven compensation scheme for last-mile delivery with crowdsourcing”. MA thesis. University of Porto, 2019. URL: https://repositorio-aberto.up.pt/bitstream/10216/124212/2/367287.pdf.

[12] William Benton and Manuel Rossetti. “The vehicle scheduling problem with intermittent customer demands”. In: *Computers & Operations Research* 19.6 (1992), pp. 521–531.

[13] Oded Berman and David Simchi-Levi. “Finding the optimal a priori tour and location of a traveling salesman with nonhomogeneous customers”. In: *Transportation Science* 22.2 (1988), pp. 148–154.

[14] Dimitris Bertsimas. “Probabilistic combinatorial optimization problems”. PhD thesis. Massachusetts Institute of Technology, 1988.

[15] Dimitris Bertsimas and Louis Howell. “Further results on the probabilistic traveling salesman problem”. In: *European Journal of Operational Research* 65.1 (1993), pp. 68–95.

[16] Leonora Bianchi and Ann Melissa Campbell. “Extension of the 2-p-opt and 1-shift algorithms to the heterogeneous probabilistic traveling salesman problem”. In: *European Journal of Operational Research* 176.1 (2007), pp. 131–144.

[17] Leonora Bianchi, Joshua Knowles, and Neill Bowler. “Local search for the probabilistic traveling salesman problem: Correction to the 2-p-opt and 1-shift algorithms”. In: *European Journal of Operational Research* 162.1 (2005), pp. 206–219.

[18] Mauro Birattari et al. “Estimation-based local search for stochastic combinatorial optimization using delta evaluations: a case study on the probabilistic traveling salesman problem”. In: *INFORMS Journal on Computing* 20.4 (2008), pp. 644–658.

[19] Neill Bowler, Thomas Fink, and Robin Ball. “Characterization of the probabilistic traveling salesman problem”. In: *Physical Review E* 68.3 (2003), p. 036703.

[20] Ann Melissa Campbell. “Aggregation for the probabilistic traveling salesman problem”. In: *Computers & Operations Research* 33.9 (2006), pp. 2703–2724.

[21] Ann Melissa Campbell and Barrett Thomas. “Challenges and advances in a priori routing”. In: *The vehicle routing problem: latest advances and new challenges*. Springer, 2008, pp. 123–142.

[22] Vincent Castillo et al. “Crowdsourcing last mile delivery: strategic implications and future research directions”. In: *Journal of Business Logistics* 39.1 (2018), pp. 7–25.

[23] Lars Dahle, Henrik Andersson, and Marielle Christiansen. “The Vehicle Routing Problem with Dynamic Occasional Drivers”. In: *Computational Logistics*. Ed. by Tolga Bekta et al. Springer International Publishing, 2017, pp. 49–63.

[24] Iman Dayarian and Martin Savelsbergh. “Crowdshipping and Same-day Delivery:Employing In-store Customers to Deliver Online Orders”. In: *Optimization Online* (2017). URL: http://www.optimization-online.org/DB_HTML/2017/07/6142.html.

[25] Mauro Dell’Amico, Francesco Maffioli, and Peter Värbrand. “On prize-collecting tours and the asymmetric travelling salesman problem”. In: *International Transactions in Operational Research* 2.3 (1995), pp. 297–308.

[26] Aashwinikumar Devari, Alexander Nikolaev, and Qing He. “Crowdsourcing the last mile delivery of online orders by exploiting the social networks of retail store customers”. In: *Transportation Research Part E: Logistics and Transportation Review* 105 (2017), pp. 105–122.

[27] Dominique Feillet, Pierre Dejax, and Michel Gendreau. “Traveling salesman problems with profits”. In: *Transportation science* 39.2 (2005), pp. 188–205.

[28] Martina Fischetti and Marco Fraccaro. “Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks”. In: *Computers & Operations Research* 106 (2019), pp. 289–297.

[29] Jerome Friedman. “Stochastic gradient boosting”. In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[30] Katarzyna Gdowska, Ana Viana, and João Pedro Pedroso. “Stochastic last-mile delivery with crowd-shipping”. In: *Transportation research procedia* 30 (2018), pp. 90–100.

[31] Michel Gendreau, Gilbert Laporte, and René Séguin. “Stochastic vehicle routing”. In: *European Journal of Operational Research* 88.1 (1996), pp. 3–12.

[32] Árni Halldórsson et al. “Comparative analysis of the carbon footprints of conventional and online retailing”. In: *International Journal of Physical Distribution & Logistics Management* (2010).

[33] Kuancheng Huang and Muhammad Nashir Ardiansyah. “A decision model for last-mile delivery planning with crowdsourcing integration”. In: *Computers & Industrial Engineering* 135 (2019), pp. 898–912.

[34] Patrick Jaillet. “A priori solution of a traveling salesman problem in which a random subset of the customers are visited”. In: *Operations research* 36.6 (1988), pp. 929–936.

[35] Patrick Jaillet. “Probabilistic traveling salesman problems”. PhD thesis. Massachusetts Institute of Technology, 1985.

[36] Kafle Kafle, Bo Zou, and Jane Lin. “Design and modeling of a crowdsource-enabled system for urban parcel relay and delivery”. In: *Transportation Research Part B: Methodological* 99 (2017), pp. 62–82.

[37] Felipe Lagos, Mathias Klapp, and Alejandro Toriello. “Branch-and-Price for Probabilistic Vehicle Routing”. In: *Draft* (2017).

[38] Gilbert Laporte, Francois Louveaux, and Hélene Mercure. “A priori optimization of the probabilistic traveling salesman problem”. In: *Operations research* 42.3 (1994), pp. 543–549.

[39] Weiqi Li. “A Simulation-Based Algorithm for the Probabilistic Traveling Salesman Problem”. In: *EVOLVE – A Bridge between Probability, Set Oriented Numerics and Evolutionary Computation VII*. Springer, 2017, pp. 157–183.

[40] Yu-Hsin Liu. “A hybrid scatter search for the probabilistic traveling salesman problem”. In: *Computers & Operations Research* 34.10 (2007), pp. 2949–2963.

[41] Giusy Macrina et al. “Crowd-shipping with time windows and transshipment nodes”. In: *Computers & Operations Research* 113 (2020).

[42] Giusy Macrina et al. “The Vehicle Routing Problem with Occasional Drivers and Time Windows”. In: *Optimization and Decision Science: Methodologies and Applications*. Ed. by Antonio Sforza and Claudio Sterle. Springer International Publishing, 2017, pp. 577–587.

[43] Soumaya Sassi Mahfoudh, Walid Khaznaji, and Monia Bellalouna. “A branch and bound algorithm for the porbabilistic traveling salesman problem”. In: *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2015, pp. 1–6.

[44] André Maravilha. *Discorde*. 2018. URL: `https://github.com/andremaravilha/discorde-tsp`.

[45] Yannis Marinakis and Magdalene Marinaki. “A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem”. In: *Computers & Operations Research* 37.3 (2010), pp. 432–442.

[46] Yannis Marinakis, Athanasios Migdalas, and Panos Pardalos. “Expanding neighborhood search–GRASP for the probabilistic traveling salesman problem”. In: *Optimization Letters* 2.3 (2008), pp. 351–361.

[47] Fabian Meier and Uwe Clausen. “Solving the probabilistic traveling salesman problem by linearising a quadratic approximation”. In: *Optimization Online* (2015).

[48] Roberto Montemanni et al. “Machine Learning and Monte Carlo Sampling for the Probabilistic Orienteering Problem”. In: *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE. 2018, pp. 14–18.

[49] Donald Morrison. “PATRICIA - Practical algorithm to retrieve information coded in alphanumeric”. In: *Journal of the ACM* 15.4 (1968), pp. 514–534.

[50] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12 (2011), pp. 2825–2830.

[51] John Quinlan. “Learning with continuous classes”. In: *5th Australian joint conference on Artificial Intelligence*. Vol. 92. 1992, pp. 343–348.

[52] Gerhard Reinelt. “TSPLIB A traveling salesman problem library”. In: *ORSA Journal on Computing* 3.4 (1991), pp. 376–384.

[53] Alberto Santini. *alberto-santini/ptspc*. Version 1.0. Jan. 2020. DOI: 10.5281/zenodo.3631678. URL: https://doi.org/10.5281/zenodo.3631678.

[54] Hao Tang and Elise Miller-Hooks. “Approximate procedures for probabilistic traveling salesperson problem”. In: *Transportation Research Record: Journal of the Transportation Research Board* 1882 (2004), pp. 27–36.

[55] Bram Verweij et al. “The sample average approximation method applied to stochastic routing problems: a computational study”. In: *Computational Optimization and Applications* 24.2-3 (2003), pp. 289–333.

[56] Christoph Weiler et al. “Heuristic Approaches for the Probabilistic Traveling Salesman Problem”. In: *International Conference on Computer Aided Systems Theory*. Springer, 2015, pp. 342–349.

[57] Mengying Zhang, John Wang, and Hongwei Liu. “The Probabilistic Profitable Tour Problem”. In: *International Journal of Enterprise Information Systems (IJEIS)* 13.3 (2017), pp. 51–64.

[58] Mengying Zhang et al. “Traveling salesman problems with profits and stochastic customers”. In: *International Transactions in Operational Research* 25.4 (2018), pp. 1297–1313.