Lecture Notes in Management Science (2013) Vol. **5**: 143–152
5th International Conference on Applied Operational Research, Proceedings

ISSN 2008-0050 (Print), ISSN 1927-0097 (Online)

A sampling-based approximation of the objective function of the orienteering problem with stochastic travel and service times

V. Papapanagiotou, D. Weyland, R. Montemanni and L.M. Gambardella

IDSIA – USI/SUPSI, Galleria 2, Manno, Switzerland
roberto@idsia.ch

Abstract. In this paper, a variant of the orienteering problem in which the travel and service times are stochastic, is examined. Given a set of potential customers, a subset of them has to be selected to be serviced by the end of the day. Every time a delivery to a selected customer is fulfilled before the end of the day, a reward is received, otherwise, if the delivery is not completed, a penalty is incurred. The target is to maximise the expected income (rewards-penalties) of the company. The focus of this paper is to evaluate a sampling based way to approximate the objective function which is designed to be later embedded in metaheuristics.

Keywords: Monte Carlo sampling; orienteering problem; stochastic optimization

Introduction

The orienteering problem with stochastic travel and service times (OPSTS) was first introduced in [1]. In this problem there is a starting point that we call the depot. A vehicle goes out of that depot and has to serve some customers. The vehicle will end its route at a destination node and does not have to return to the depot. There is a global deadline and in most cases it is such that it is not possible to visit all the customers before the deadline. For this reason, a subset of customers has to be selected to be served. After selecting this subset, the vehicle tries to serve the customers in the subset. For each service before the deadline it earns a reward, otherwise a penalty is incurred. This paper examines how to approximate the objective function of the

problem more efficiently and with minimal loss in accuracy using Monte Carlo sampling. This makes it ideal to embed it in a metaheuristic.

The rest of the paper is organized as follows. After a literature review in Section 2 and the formal problem definition in Section 3, different ways to compute the objective function are presented in Section 4. In that section, a traditional exact method is presented (Section 4.1) along with our proposed Monte Carlo approximation of the objective function (Section 4.2). Extensive experimental results are presented in Section 5 while Future Work is presented in Section 6 and Conclusions are drawn in Section 7.

Related work

Several variants of the orienteering problem have been studied in the past. A recent survey can be found on [2]. One of the most closely related to this paper is [1], where the same problem is examined. In [1], the authors suggest exact solutions for three simplified versions of the problem and then go on to suggest a variable neighborhood search metaheuristic for the problem. To compute the objective function of the problem, they use an exact method.

To the best of our knowledge apart from [1], there are few other papers on the subject of stochastic variants of the orienteering problem. Related to OPSTS are stochastic variants of the traveling salesman problem. In [3] and [4] Ant Colony systems are proposed to solve the probabilistic traveling salesman problem and the team orienteering problem with time windows which have similarities to OPSTS. Another related paper to OPSTS is the time-constrained traveling salesman problem with stochastic travel and service times (TCTSP). This problem was introduced and solved for the first time in [5]. The problem is formulated as a two-stage stochastic program and an integer L-shaped solution method is proposed for solving it. The algorithm is used to solve problems of up to 35 customers. Another related problem is the stochastic selective travelling salesperson problem (SSTP) introduced in [6]. In SSTP, travel and service times are stochastic as in this paper. However, the deadline is being enforced by chance constraints and not by negative rewards as in this paper. The authors propose both an exact and a heuristic approach to solve the problem. A stochastic version of the orienteering problem is also examined in [7] but the stochasticity is in the profits associated with each customer.

Problem definition

In this section, OPSTS is explained in details. The formal definition of OPSTS is given, which is in accordance with [1]. Then, the cost evaluation is defined and discussed.

Formal definition

Let $N = \{1, \dots, n\}$ be a set of customers. As a first phase, a subset of customers has to be selected to represent the set of customers that it is believed that can be serviced before the deadline. Let $M \subseteq N$ be the set of customers selected to be serviced before the global deadline D. The depot is defined as node 0. We assume that there is an arc (i, j) for all $i, j \in M$. Each customer $i \in M$ is associated with a reward value r_i and a penalty value e_i. To earn the reward r_i ,the customer $i \in M$ must be served before the global deadline D. If the customer $i \in M$ fails to be served before D, a penalty e_i is incurred.

Let $X_{i,j}$ be a non-negative random variable representing the time required to traverse the arc (i, j). We assume that the distribution for $X_{i,j}$ is known for all i and j. Let $\mathrm{S_i}$ be a non-negative random variable representing the service at customer i. Let the random variable $\mathrm{A_i}$ be the arrival time at customer i and $\bar{A}_\mathrm{i}$ a realisation of $\mathrm{A_i}$. Let $R(\bar{A}_\mathrm{i})$ be a function representing the reward earned at customer i when arriving to i at time $\bar{A}_\mathrm{i}$. We assume that $R(\bar{A}_\mathrm{i}) = r_i$ for $\bar{A}_\mathrm{i} \leq \mathrm{D}$, otherwise $R(\bar{A}_\mathrm{i}) = -e_i$ for $\bar{A}_\mathrm{i} > D$.

Cost Evaluation

We define as tour of the customers τ the order in which customers are visited in the selected set M. Then the expected profit of the tour, which is also our objective function, is:

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

We seek a tour τ^* such that $\upsilon(\tau^*) \geq \upsilon(\tau)$ for every τ.

Computing the objective function

In this section we discuss different ways to approximate the objective function (1).

Exact method

The method presented in this section, is the one used in [1] for computing the objective function. Service times can be accounted for by convoluting them with the travel time distribution of the relevant arc and therefore we do not treat them separately. In this method, in order to compute the objective function we use the assumption that travel times from i to j denoted as $X_{i,j}$ are Γ distributed. The Γ distribution takes 2 parameters, a shape parameter k and a scale parameter θ. Each $X_{i,j}$ is Γ distributed with a $k_l, l \in [0, |N|^2]$, parameter which is set as the mean travel time and a θ parameter which for our tests is set to 1.0. Travel times $X_{i,j}$ are all independent from each other.

The arrival time A_i of a node is the sum of the time values $X_{j,k}$of all the arcs j,k in the path from the depot to i. Therefore, A_i is a sum of Γ distributed random variables. If Y_i has a $\Gamma(k_i,\theta)$ distribution for $i = 1,2,\dots,N$ and all Y_i are independent then $\sum_{i=1}^{N} Y_i \sim \Gamma(\sum_{i=1}^{N} k_i,\theta)$. Clearly, since A_i is a sum of Γ distributed random variables, it can be approximated by the Γ function of the sum of all k_l of the relevant $X_{j,k}$ variables added.

Considering that the Cumulative Distribution Function (CDF) of A_j computes the probability $(A_i \leq x)$, and also that A_i is Γ distributed, by using the CDF of the Γ distribution F_{k_i} where k_i is the Γ parameter k of the i^{th} arc, the objective function (1) can be rewritten as:

$$v(\tau) = \sum_{i\in\tau} [F_{k_i}(D) r_i - \left(1 - F_{k_i}(D)\right) e_i] \tag{2}$$

We assume that we have a way to compute F_{k_i} using a function called $F_\Gamma(k,\theta)$. We also assume that we have a function called **distance** that can compute the distance between 2 nodes. We can now construct an algorithm called $ExactCost(sol)$ that computes (2). The algorithm takes as input a solution vector (*sol*). Each member of the solution vector is a node which represents a client that we chose to visit in the particular solution. The nodes are ordered in the vector in the order that they will be visited. The 0^{th} element of the solution vector is always the depot.

Therefore, in order to compute the sum in (2), we loop over all the solution nodes(found in the vector sol), computing the objective function for each arc and summing the results of the calculation.

Monte Carlo method

In this section we discuss our proposed approach to approximate the objective function (1) more efficiently.

The objective function (1) is a core element of our solution and runs thousands of times and therefore even a small improvement in speed can result in substantial overall gains. In the objective function described in Section 4.1, a function $F_\Gamma(k,\theta)$ is used to calculate the CDF of the Γ function. The run time of this function dominates the run time of the algorithm inside the loop, hence, it is the most appropriate candidate for improvement. We can completely avoid using $F_\Gamma(k,\theta)$, by using Monte Carlo Sampling. A first approach would be for each travel time $X_{i,j}$ in each solution to create many realisations $\bar{X}_{i,j}$ and take the average to actually compute each $X_{i,j}$. However, creating many realisations of $\bar{X}_{i,j}$ implies that for each arrival time we use a function to produce distributed random variables (in this case Γ distributed). Because random generating functions also tend to be computationally expensive, this method is not more efficient than the exact. Therefore, we need a method that avoids calling random generating functions of distributions for every arc of our graphs.

We avoid these costly procedures by creating a precomputation matrix for each type of stochastic variable in the problem. For example, in this paper we only have

one type of stochastic variable that represents the convolution of stochastic travel and service times. Each precomputation matrix has as many submatrices as number of samples, that represent realizations of a type of stochastic variable for a sample. As an example, in this paper a submatrix is two dimensional and has realizations of the random variable $X_{i,j}$ which represents the convolution of travel and service times. We must note that in approximating the objective function using Monte Carlo sampling, no property particular to some distribution is used and therefore it can be generalized for any random distribution.

Once the precomputation matrix is created and we have many pregenerated samples from every node to every node, we can compute an approximation of the objective function very fast. In **Algorithm 1** it is presented the implementation of $MonteCarloCost$ which computes the objective function, given the precomputation matrix $vector$. sol (the solution vector) is an array representing the order of the nodes in the solution. $samples$ is the number of samples to use and D is the global deadline. $MonteCarloCost$ returns v which is the approximative cost of the objective function. It should be noticed that for each solution we calculate the sum of all penalties, which is a negative value and we initialize our objective value v with it. Thus, we have precomputed all the penalties and subtracted them so now, while each node is added before deadline we add back its penalty along with its reward.

```
function MonteCarloCost(sol,samples,vector,D):
for all i ∈ [1,...,samples] do
    i←0
    v ← sum_of_penalties
    curTime ← 0
    while curTime ≤ D and i < |S| do
        curTime+ = vector[sample][sol[i]][sol[i + 1]]
        v ←v +ri+1 +ei+1
        i←i+1
    end while
end for
 v ← v /|samples|
return v
```

Algorithm 1. The algorithmic representation of the function *MonteCarloCost* for OPSTS

Experimental results

In this section we present the results of experiments to assess the usefulness of the usage of Monte Carlo sampling and compare it to the exact method. Monte Carlo sampling can offer a reasonable approximation of the objective function in only a fraction of the time and the time gains are increasing over time. The implementation of the solutions is in C++ and the instances ran on a 4-core Intel Core I7-3615QM

2.3GHz using Mac OSX 10.8. The 4 cores shared 16GB of RAM and memory consumption was never a problem. Firstly, we will compare the error of the Monte Carlo approach to the traditional method and then we will compare the running times.

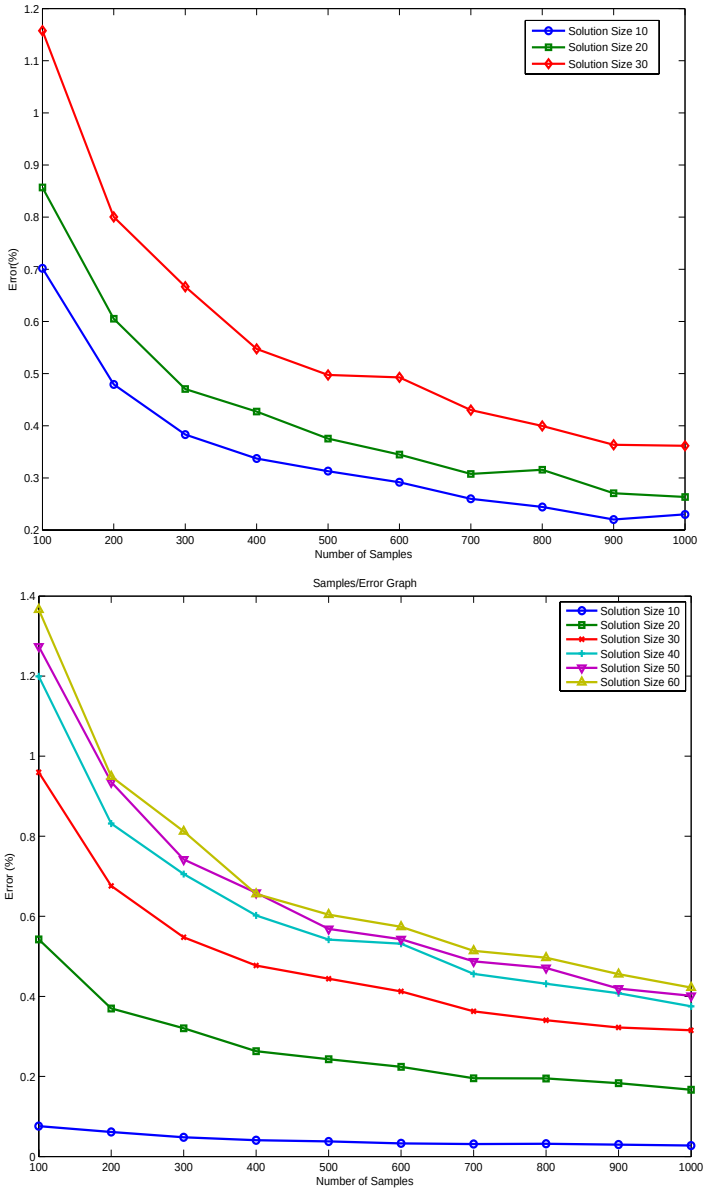

Fig. 1. Two graphs (a and b) showing the reduction of the error with respect to the number of samples used in the Monte Carlo sampling. Different sizes of solutions are considered. The deadline in this set is 50. The dataset size is 32 and 64 nodes respectively.

Datasets and implementation details

Because this problem was introduced in [1], we mostly use the dataset generation procedures used in that paper. Two of the datasets are based on the sets first appearing in [8]. For all the datasets it is assumed that the graph of the customers is fully connected and travel and service times are computed as described in the section 4.1. The penalty values, are generated as a fraction of the rewards in [1] and for comparison purposes the same method is used in this paper. We consider fractions of 0.1 for penalties in our experiments. In these experiments, for distances, the Euclidean distance is used and for probability distribution for the travel and service times, the Γ distribution is used for comparison purposes with [1]. The Γ distribution is computed using the boost math library, which uses Lanczos approximation for the computation [10].

Number of samples in relation to error

To demonstrate the utility of the Monte Carlo approach, first it needs to be shown that it produces a reasonably low error and then choose the number of samples that produce an acceptable error value. For this reason, we run experiments where we measure the relative error of Monte Carlo sampling in relation to the Exact Method, and present this value in relation to the number of samples used in Monte Carlo sampling. We vary the number of samples from 100 to 1000 with step 100. The solutions evaluated are generated randomly and then they are optimized by a simple greedy algorithm. Optimization is used because random solutions can be of such a low quality that not even one client can be served without surpassing the deadline. Such cases generate bigger relative errors between the Monte Carlo approach and exact methods than one would encounter in reality, as Monte Carlo sampling is used in conjunction with an optimization method. The greedy algorithm does not find new solutions but optimizes the order of visiting nodes in a given solution. The results of the experiments can be seen in **Fig.**. The error is in comparison with the exact method presented. Solution Size is the number of nodes of each solution given for evaluation. It can be observed that as we use more samples, the error decreases. Furthermore, for the same number of samples, for larger solution sizes more relative error is obtained. This can be justified if we consider that we approximate the sum of more random variables (than in a smaller solution) and thus we increase the accumulation of error from each estimation of the random variables. Additionally, in both figures it can be seen that the error is less than 1.4% even with 100 samples for all solution sizes. Therefore, since the algorithm gives a reasonable approximation of the objective function, it is worth it investigating it further.

When Monte Carlo sampling is beneficial

In this paper, we use precomputation matrices to speed up Monte Carlo sampling. Precomputation matrices introduce an overhead and it would be desirable to know when we offset that overhead. The average time for 1 evaluation using Monte Carlo sampling is:

$$\frac{avg(setup\_time)}{\#Evaluations} + avg(time\_mc) \tag{3}$$

According to (3) when the number of evaluations is very large $\#Evaluations \rightarrow \infty$, only the cost from the $MonteCarloCost$ ($avg(time\_mc)$) function is important and the cost of building the precomputation matrix $avg(setup\_time)$ does not matter. To examine the problem more realistically, in **Fig.** we have plotted $\frac{MonteCarloCost\ Time}{ExactCost\ Time}$ for solution sizes 30 and 60 using 100 and 200 number of samples respectively. When this fraction is lower than 1 then the average time for 1 Monte Carlo sampling evaluation is less than when using the ExactCost. In **Fig.**, we can observe that for solution size 30 we need to run the objective function 2000 times or more to start having speed gains and for solution size 60 more than 4000. Because the objective function is meant to be called within a metaheuristic, where 100000 calls to the objective functions are typical if not too few, this method has the potential to offer considerable speed gains.

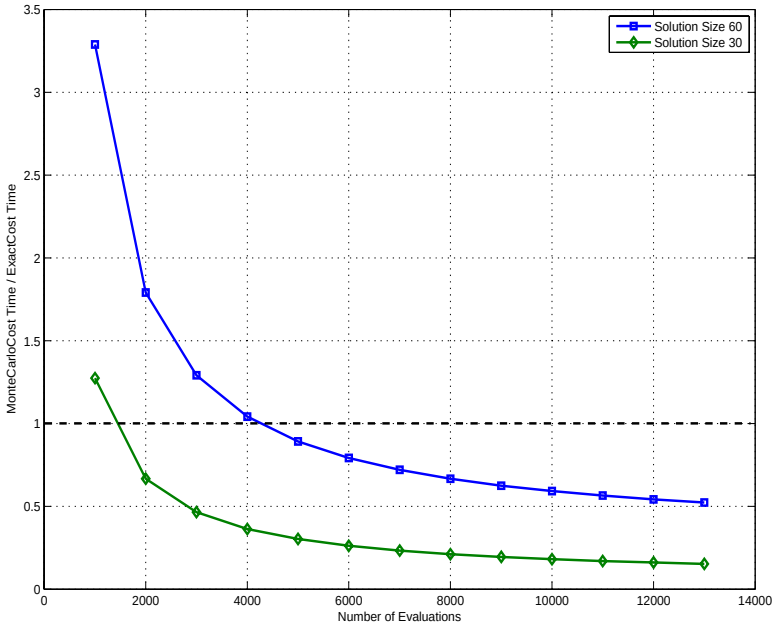

Fig. 2. $\frac{MonteCarloCost\ Time}{ExactCost\ Time}$ in relation to Number of Evaluations for solution sizes 30 and 60. For solution size 30 we use 100 samples and for solution size 60 we use 200.

Running time comparison

In this section we are doing a running time comparison between *ExactCost* and *MonteCarloCost*. **Table** presents the comparison. Times are measured in seconds and are taken as an average after running the respective algorithm 100 *(#Evaluations*) times. In the first column we have the number of samples used in the *MonteCarloCost* algorithm, the second *time_exact* has the running time of the *ExactCost*. Next *setup_time* has the running time of creating the precomputation matrix and the column *time_mc* the time of a typical execution of *MonteCarloCost*.

Table 1. Comparison of running times of the ExactCost and MonteCarloCost for Solution Size 64 and 100 evaluations

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

The last column contains the result of a ratio. When it is smaller than 1 then *MonteCarloCos*t - without taking into account precomputation costs - is faster than the *ExactCost*. According to (3) when the number of evaluations is very large $\#Evaluations \to \infty$, only the cost from the *MonteCarloCos*t function is important. From the column *time_mc/time_exact*, it is easily observable that the time consumed for computing the objective function using Monte Carlo methods is a small fraction of the time of the exact cost until a certain number of samples. Additionally, the error is in most cases sufficiently small even by using 100 samples. This indicates a significant gain and utility for the method.

Conclusions and future work

The objective function of many combinatorial transportation problems is computationally expensive and many times it is the time bottleneck of the algorithm. By using Monte Carlo sampling with precomputation, the procedure can speed up the computation of the objective function considerably with a small amount of error, provided that the objective function is called a large number of times. Additionally, Monte Carlo sampling can be modified easily in order to work with any probability distribution. Finally, because metaheuristics need and use objective functions a large number of times in order to return a solution to a problem, Monte Carlo sampling is especially suited for speeding up a metaheuristic by calculating the objective function more efficiently.

There is a number of improvements that can be made. Firstly, a suitable metaheuristic to solve the problem has to be designed to run on top of the objective function evaluator. Additionally, the metaheuristic can be developed in such a way so that when a perturbation is executed, only a partial evaluation of the objective function is needed. This can be achieved by a careful design of the perturbation functions and by informing the objective function evaluator about the changes. Furthermore, there can be a built-in recommendation mechanism for assisting the choice of the number of samples used during sampling. Finally, Monte Carlo sampling can be parallelized using the GPU. Some relevant work can be found in [9].

References

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

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

Gambardella, L., Montemanni, R., Weyland, D.: Coupling ant colony systems with strong local searches. European Journal of Operational Research 220(3) (2012) 831 – 843

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

Teng, S.Y., Ong, H.L., Huang, H.C.: An integer l-shaped algorithm for time- constrained traveling salesman problem with stochastic travel and service times. Asia-Pacific Journal of Operational Research 21(02) (2004) 241–257

Tang, H., Miller-Hooks, E.: Algorithms for a stochastic selective travelling salesper- son problem. Journal of The Operational Research Society 56 (2005) 439–452

Ilhan, T., Iravani, S.M.R., Daskin, M.S.: The orienteering problem with stochastic profits. Iie Transactions 40 (2008) 406–421

Tsiligirides, T.: Heuristic Methods Applied to Orienteering. Journal of the Operational Research Society 35 (1984) 797–809

Weyland, D., Montemanni, R., Gambardella, L.M.: A metaheuristic framework for stochastic combinatorial optimization problems based on gpgpu with a case study on the probabilistic traveling salesman problem with deadlines. Journal of Parallel and Distributed Computing 73(1) (2013) 74 – 85 Metaheuristics on GPUs.

http://www.boost.org/doc/libs/1_53_0/libs/math/doc/sf_and_dist/html/math_toolkit/backgr ounders/lanczos.html.