A Regret Policy For The Dynamic Vehicle Routing Problem With Time Windows

Peter Dieter
Paderborn University
33098 Paderborn, Germany
peter.dieter@upb.de

Abstract

This document describes team UPB's submission to the EURO Meets NeurIPS 2022 Vehicle Routing Competition. More specifically, we present a *regret* policy for the dynamic vehicle routing problem with time windows (DVRPTW) in which customer order arrivals are revealed over the course of a day. The problem requires two types of decisions: Dispatching orders and planning the respective vehicle routes. The *regret* policy we present in this paper focuses on the former, namely dispatching orders. The basic idea of the policy is to calculate a regret value for each order which represents possible improvements that we would miss when dispatching the order immediately. To attain this regret value, we make use of the customer distribution from which orders are sampled. If the value is below a predefined threshold, the order is dispatched and routes are planned with a state-of-the-art VRPTW solver. The proposed policy outperforms several benchmark policies. Furthermore, due to its high level of explainability, ease to implement, and generalizability, the proposed *regret* policy can serve as a benchmark policy for other studies on anticipatory vehicle routing.

1 Introduction

Due to trends in the logistics industry such as same-day delivery, dynamic vehicle routing is becoming increasingly important [1]. This paper describes a *regret* policy for the dynamic vehicle routing problem with time windows (DVRPTW), which has been the posed problem in the *EURO Meets NeurIPS 2022 Vehicle Routing Competition* [2]. In the considered problem, orders are sampled from a known customer distribution over the course of a day. Once an hour, it needs to be decided which orders need to be dispatched and which ones to postpone to consolidate with future requests that may arrive later. After it has been decided which orders are dispatched, feasible routes for these orders need to be created. Customer time windows are *hard* and an unlimited vehicle fleet is assumed. The overall goal is to minimize the distance traveled during all decision epochs. For a more detailed explanation of the problem, we refer to [2]. The policy presented in this paper only focuses on the dispatching decision. This is done by calculating a regret value for each *undue* order, i.e., orders which can be postponed, considering the total set of customers from which orders are randomly sampled. Furthermore, we describe methods to extend the regret value with information of time windows and other undue orders. The resulting VRPTW instance is then solved by a state-of-the-art hybrid genetic search (HGS) algorithm developed by [3], which is an extension of the HGS solver which has been proven successful for other VRP variants [4, 5]. We show that the proposed policy outperforms two benchmark policies, provided by the organizers of the *EURO Meets NeurIPS 2022 Vehicle Routing Competition*. Due to its high level of explainability, ease to implement, and generalizability, we believe that the proposed policy can serve as a benchmark policy for other studies on dynamic vehicle routing, in which anticipating future customer requests is crucial. Furthermore, several possible extensions of the policy exist and could be considered in future work.

EURO Meets NeurIPS 2022 Vehicle Routing Competition Workshop.

The further structure of the paper is as follows: In Section 2 we present the developed *regret* policy. The policy is compared to two benchmarks in Section 3 and concludes with a summary and an outlook in Section 4.

2 Regret Policy

In this section, we first describe the general algorithmic procedure of the *regret* policy. We then describe the regret function and explain two extensions of the basic regret function, namely the incorporation of information about time windows and undue orders. An open-source Python implementation of the policy is available on GitHub[1].

2.1 General procedure

The general procedure of the algorithm is described in Algorithm 1. The procedure requires a set of due orders D, which are orders that need to be dispatched immediately as postponement would lead to a time window violation. Furthermore, the following is required as input: a set of undue orders U, a set of all customers from which orders are drawn S, and a threshold value t which can further be tuned. To enter the while loop, we set the best (lowest) regret value r to t. While the best regret value is equal or below the threshold, we do the following: For each undue order $u \in U$ we calculate its regret value with the GetRegret function, which takes the order u, D, and S as input. This function is further explained in subsection 2.2. If the lowest regret value is below threshold value t, the respective order c is marked as dispatched, added to the set of due orders D, and removed from the set of undue orders U. This procedure is repeated until the regret values of all orders are above t, or U is an empty set. Therefore, the final set of D includes all orders which are dispatched, and consequently, feasible routes for these orders are determined by the HGS solver [3].

Algorithm 1 Regret policy procedure

Input: Set of due orders D, set of undue orders U, set of all customers S, threshold value t

$r \leftarrow t$
while $r \leq t$ **do**
    $R \leftarrow \{\}$
    for $u \in U$ **do**
        $R \leftarrow R \cup$ *GetRegret*(u, D, S)
    end for
    r, c $\leftarrow$ *min*(R), *argmin*(R)
    if $r \leq t$ **then**
        $D \leftarrow D \cup c$
        $U \leftarrow U \setminus c$
    end if
end while

Output: Set of orders to dispatch D

2.2 Regret function

The motivation behind the proposed regret function is the following: We can expect that an order $u \in U$ which is undue will be served consecutively before or after its closest order $d \in D$ which is marked as dispatched. Let c_{ud} be the distance (costs) from the customer of order u to the customer of order d. We then approximate that dispatching order u will result in additional costs of $\frac{c_{ud}+c_{du}}{2}$, assuming that c_{ud} does not deviate much from c_{du} and no further order being dispatched. We note that this is only a simple approximation, as order u might be on the route to a dispatched order, which might result in lower costs. As order u is not due yet, better (closer) orders might arrive in the future. Let $B \subseteq S$ be those closer customers. The improvement of routing u consecutively after/before $i \in B$ compared to d is given by:

$$\frac{c_{ud} + c_{du}}{2} - \frac{c_{ui} + c_{iu}}{2} \tag{1}$$

[1] `https://github.com/PeterDieter/RegretPolicy_DVRPTW`

These improvements can be seen as a *regret*, i.e., possible improvements which we would miss when dispatching u immediately. As the regret also depends on the probability that a *better* customer will be drawn until order u is due, we adjust the regret by this probability: Let n be the number of orders that are expected to be drawn until order u is due. In the competition, n could be derived logically, as a constant number of orders and time windows was drawn at each epoch, and orders with unsuitable time windows, i.e., time windows that end before the respective order can be served, were removed. The probability that a customer is drawn in a single draw is $\frac{1}{|S|}$ and its counter probability is, therefore, $1 - \frac{1}{|S|}$. The probability that a customer is not drawn in n drawings is then given by $(1 - \frac{1}{|S|})^n$. The probability that a customer is drawn in n drawings is therefore, $1 - (1 - \frac{1}{|S|})^n$. The adjusted regret value R_u of order u is therefore given by:

$$R_u = \sum_{i \in S} (\frac{c_{ud} + c_{du}}{2} - \frac{c_{ui} + c_{iu}}{2} [\frac{c_{ui} + c_{iu}}{2} < \frac{c_{ud} + c_{du}}{2}]) \cdot (1 - (1 - \frac{1}{|S|})^n) \tag{2}$$

where square brackets are *Iverson* brackets, i.e., its inner value is 1 if the condition ($\frac{c_{ui}+c_{iu}}{2} < \frac{c_{ud}+c_{du}}{2}$) holds and 0 otherwise.

2.3 Incorporating time window information

So far, the *regret* policy only takes time windows into account for determining when orders become *due*. However, neglecting time windows can lead to inefficient route plannings, in which vehicles need to wait before an order can be served. Therefore, we multiply the regret value with a term that penalizes the time a vehicle would need to wait if serving order u directly before/after order d. We note that also an additive term could be added, but pretests have shown that a multiplicative term leads to better solutions. Let $[w_d^-, w_d^+]$ be the time window of order d and $[w_u^-, w_u^+]$ of order u, respectively. The minimal waiting time if for serving order u directly before order d is given by $w_{du} = max(0, w_u^- - w_d^+ + c_{du})$ and respectively, w_{ud} is given by $max(0, w_d^- - w_u^+ + c_{ud})$. The minimum waiting time of serving u directly before d or vice versa is then $min(w_{du}, w_{ud})$. Therefore, we update the regret value as follows:

$$R_u \leftarrow R_u \cdot (1 + min(w_{du}, w_{ud})) \cdot h_2 \tag{3}$$

where h_2 is a hyperparameter that determines how much waiting time is penalized. High values of h_2 lead to higher penalizations and its value should be carefully tuned.

2.4 Incorporating undue orders

In the basic formulation of the regret function, other orders which are not due U are not taken into account. However, these known undue orders provide valuable information and should therefore be incorporated. Let us assume the following example: Two orders u and u_1 are very close to each other. Moreover, u is due in one epoch while u_1 is due in 5 epochs. Therefore, the regret value for u_1 is likely to be high and the order might not be dispatched immediately. On the contrary, as order u is due soon, its regret value will likely be low and the order might be dispatched immediately. Order u and u_1 would therefore not be dispatched together, which might result in inefficient routes. The procedure of incorporating U into the regret function is similar to the incorporation of the set of all customers S described in the previous subsection. However, there are two important differences: Instead of taking into account all customers S, we now only consider customers of orders which are in the system and not dispatched, i.e., we consider $B \subseteq U$ where

$$\frac{c_{ui} + c_{iu}}{2} < \frac{c_{ud} + c_{du}}{2} \quad \forall i \in U. \tag{4}$$

Furthermore, as orders B are already in the system, we do not correct the value with a probability term. Therefore, we update the regret value as follows:

$$R_u \leftarrow R_u + \sum_{i \in B} (\frac{c_{ud} + c_{du}}{2} - \frac{c_{ui} + c_{iu}}{2}) \cdot h_1 \tag{5}$$

where h_1 is a hyperparameter that determines the weight given to possible improvements by using information of undue orders.

3 Evaluation

In this section, we evaluate the *regret* policy by comparing it to two benchmark policies provided by the organizers of the *EURO Meets NeurIPS 2022 Vehicle Routing Competition* [2]. These benchmark policies are a greedy dispatching rule which dispatches orders as soon as possible and a lazy dispatching rule which dispatches only orders which are due. To tune hyperparameters, we applied a grid search. This resulted in the following chosen values: $t = 0.08$, $h_1 = 0.01$, and $h_2 = 0.002$. We test the policies on 10 randomly chosen instances with a random seed value of 1 (see [2] for further information on the systems' environment). The time limit for the HGS solver is set to 10 seconds for all policies. Experiments were performed on an Intel Core i7 with 2.6GHz and 12 GB RAM. The results are summarized in Table 1.

Table 1: Results

Instance name	**Greedy benchmark**	**Lazy benchmark**	**Regret policy**
n400-k25	460,478	647,048	403,532
n391-k23	480,232	713,428	436,282
n688-k38	488,929	637,624	434,284
n650-k42	482,211	635,218	429,217
n338-k21	371,730	470,721	346,047
n353-k21	368,997	552,568	337,125
n300-k18	456,518	623,877	389,325
n220-k13	448,380	614,315	384,276
n514-k31	514,268	707,132	458,878
n503-k43	430,597	574,062	376,397

We can see that the *regret* policy consistently outperforms the two benchmarks. On average, the proposed policy leads to improvements of around 11% compared to the *Greedy* benchmark and 35% compared to the *Lazy* benchmark.

4 Conclusion

In this paper, we present a *regret* policy for the dynamic vehicle routing problem with time windows (DVRPTW) as posed in the *EURO Meets NeurIPS 2022 Vehicle Routing Competition* [2]. In the studied problem, orders are sampled from a known customer distribution over the course of a day. Once an hour, it needs to be decided which orders need to be dispatched and which ones to postpone. After it has been decided which orders are dispatched, feasible routes for these orders need to be created. The policy presented in this paper considers the first decision stage, i.e., determining which orders to dispatch, by anticipating orders that might arrive in the future. The developed policy consistently outperforms benchmark policies. Due to its high level of explainability, ease of implementation, and generalizability to other variants of dynamic vehicle routing, the proposed *regret* policy can serve as a benchmark policy for other work on anticipatory vehicle routing. Moreover, multiple extensions of the proposed policy could be investigated. Currently, only the closest due order is regarded to determine the regret value. Future work could look for ways to incorporate the entire set of due (or already dispatched) orders to determine regret values. Furthermore, it could be tried to better account for time windows, e.g. by immediately constructing an initial route plan in the proposed policy. Another possible extension is the development of dynamic thresholds. Even though the time until an order is due is implicitly accounted for in the *regret* policy, dynamic thresholds could further improve the policy. For example, it might be beneficial to release orders more easily when the number of epochs the orders are in the system is low, as this results in wider time windows for the routing decision and consequently, fewer vehicles could be used.

References

[1] M. W. Ulmer, D. C. Mattfeld, and F. Köster, "Budgeting Time for Dynamic Vehicle Routing with Stochastic Customer Requests," *Transportation Science*, vol. 52, no. 1, pp. 20–37, 2018.

[2] W. Kool, L. Bliek, D. Numeroso, R. Reijnen, R. R. Afshar, Y. Zhang, T. Catshoek, K. Tierney, E. Uchoa, T. Vidal, and J. Gromicho, "EURO Meets NeurIPS 2022 Vehicle Routing Competition." `https://euro-neurips-vrp-2022.challenges.ortec.com/assets/pdf/euro-neurips-vrp-2022-rules-2-aug.pdf`, 2022. Accessed: 2022-11-15.

[3] W. Kool, J. O. Juninck, E. Roos, K. Cornelissen, P. Agterberg, J. van Hoorn, and T. Visser, "Hybrid Genetic Search for the Vehicle Routing Problem with Time Windows: a High-Performance Implementation," tech. rep., 12th DIMACS Challenge, 04 2022.

[4] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012.

[5] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "A unified solution framework for multi-attribute vehicle routing problems," *European Journal of Operational Research*, vol. 234, no. 3, pp. 658–673, 2014.