Article

A Dynamic Strategy for Home Pick-Up Service with Uncertain Customer Requests and Its Implementation

Yu Wu ^{1,2}, Bo Zeng ³ and Siming Huang ^{1,*}

¹ Institutes of Science and Development, Chinese Academy of Sciences, Beijing 100190, China; wuyu14@mailsucas.edu.cn

² University of Chinese Academy of Sciences, Beijing 100049, China

³ Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA; bzeng@pitt.edu

* Correspondence: simhua@casipm.ac.cn

Received: 1 March 2019 ; Accepted: 2 April 2019 ; Published: 7 April 2019

Abstract: In this paper, a home service problem is studied, where a capacitated vehicle collects customers' parcels in one pick-up tour. We consider a situation where customers, who have scheduled their services in advance, may call to cancel their appointments, and customers, who do not have appointments, also need to be visited if they request for services as long as the capacity is allowed. To handle those changes that occurred over the tour, a dynamic strategy will be needed to guide the vehicle to visit customers in an efficient way. Aimed at minimizing the vehicle's total expected travel distance, we model this problem as a multi-dimensional Markov Decision Process (MDP) with finite exponential scale state space. We exactly solve this MDP via dynamic programming, where the computing complexity is exponential. In order to avoid complexity continually increasing, we aim to develop a fast looking-up method for one already-examined state's record. Although generally this will result in a huge waste of memory, by exploiting critical structural properties of the state space, we obtain an $O(1)$ looking-up method without any waste of memory. Computational experiments demonstrate the effectiveness of our model and the developed solution method. For larger instances, two well-performed heuristics are proposed.

Keywords: home service; pick up; routing; Markov decision process (MDP); dynamic programming; uncertain request; service cancellation

1. Introduction

Nowadays, home pick-up service is very common, which involves passenger pick-up service, such as company's staff commute, as well as item pick-up service, such as household trash and express parcels. Among these home services, home pick-up express parcel service is becoming more and more popular, especially in China. In 2019, there are already almost 1200 express firms to provide home pick-up parcel services in China [1] and the number is continually increasing. Hence, to handle the constantly increasing demands and also the more competitive pressure, the service providers must optimize their activities to decrease their costs, improve service qualities and enhance productivities. In addition, as denoted in [2], this business is actually in a low margin market and is very challenging. Hence, these providers have to serve customers faster and more flexibly. Therefore, they need to deal with dynamic changes and uncertainty.

In this paper, we will focus on dynamic changes and uncertainty in customer requests occurring in home parcel pick-up services. One common uncertainty is the arrival of new customer requests during the pick-up service tour. Another more common but often ignored uncertainty in home pick-up service is the cancellation of a pre-scheduled service. For example, the United Parcel Service (UPS), a well-known express company, clearly says on its webpage [3] that "you can cancel or modify a UPS

On-Call Pickup any time before the driver arrives". Thus, in order to make our study more close to the practical situation, we take those two types of uncertainty on customer requests into account, i.e., cancellations of pre-scheduled services and new requests for unscheduled services. Aimed at minimizing the vehicle's total travel distance within one pick-up tour, we will try to find a dynamic strategy to guide the vehicle to collect parcels in response to various realization of customers' requests.

Clearly, the presented problem is a variant of the well-known Vehicle Routing Problem (VRP). More specifically, as denoted in [4], the problem belongs to the class of Dynamic Vehicle Routing Problem with Stochastic Customers (DVRPSC). As one of the hardest problems in operations research, VRP has received considerable attention. For the latest research on VRP, please see the literature [5–18]. For the latest development review, please see the literature [19–21]. Since the literature related with the VRP is very vast, it is difficult to review all the developments of VRP. Hence, we only confine our attention to the following review on DVRPSC. DVRPSC problems mainly arise in an environment where some customer requests are known in advance but others are revealed during the operational process. Due to its practical relevance, DVRPSC has attracted a lot of attention. For thorough reviews, we refer to Larsen's PhD dissertation [22] and the literature review [23] by Pillac et al. Generally, such problems are solved by computing sequences of deterministic problems (routing among customer requests that are already known). As noted in [24], although such sequential deterministic approaches allow for the direct use of existing methodologies developed for static optimization problems, the research community has realized that it is necessary to look for algorithms that recognize customer requests that might become known, leading to behaviors where vehicles are kept close to customer groups who are likely to place orders. Potvin et al. [25] and Ichoua et al. [26] investigate some classes of policy function approximations, which introduce rules for reserving vehicles for orders that have not yet become known. In the studies [27,28], similar algorithmic strategies are referred to as online algorithms, which often are implemented as simple rules that compute in real time to react to new information as new customer request arrives. Another strategy to deal with this uncertainty is to generate scenarios of potential customer requests in the future. This strategy has been used in [29] by Bent and Van Hentenryck and in [30] by Hvattum et al., where multiple scenarios of future customer requests are used to improve decisions for now. Another strategy is based on sampling, such as Monte Carlo sampling on future customer requests. This idea is investigated in [29] by Bent and Van Hentenryck and [31] by Mercier and Van Hentenryck in [32]. We also mention that Powell et al. provide a detailed tutorial on approximate dynamic programming for DVRPSC in [24].

Obviously, the existing studies on DVRPSC either produce a preprocessed decision through a static approach (e.g., [2,33]), or generate an online decision by an approximated dynamic approach (e.g., [29,34]). To the best of our knowledge, there are no studies that focus on an exact dynamic approach to simultaneously deal with two types of stochastic customers: customers whose requests might be cancelled, and customers who will make new requests in the operational stage.

Thus, in this paper, in response to the aforementioned two types of uncertainty on customer service requests, subject to vehicle's limited capacity, an exact dynamic strategy on how to collect customers' parcels within one pick-up tour and its implementation will be studied. Since Markov Decision Process (MDP) is an ideal dynamic optimization tool and has been well studied both in theory and applications (e.g., [35–38]), we formulate this problem as a multi-dimensional MDP with a finite exponential-scale state space, where the defined state consists of four components in terms of two sets and two numbers. In order to facilitate looking-up and storage operations in implementation, we have successfully designed a key only consisting of three numbers for each state. Then, by using Dynamic Programming (DP), an exact one-tour dynamic strategy with the minimum expected travel distance is obtained, in which the computing complexity is exponential. Since during the solution of MDP by DP, the already-examined states' records need to be iteratively looked up, in order to avoid complexity continually increasing, we aim to develop a fast looking-up method to obtain an already-examined state's record. Although generally this will result in a huge waste of memory space, in this paper, based on our designed state's key, by exploiting some critical structural properties of the state space, we obtain an $O(1)$ looking-up method without any waste of memory space. Finally, for larger instances

which are not applicable to be solved by DP, we propose two well-performed heuristic methods. These heuristic methods are evaluated by comparing their computational results with the results obtained through using strategy from DP on some computable instances.

The main contribution of this paper is the development of an MDP model that simultaneously considers two types of uncertainty on customer service requests occurred in vehicle routing problems with pick-up services. To the best of our knowledge, this is the first time to tackle uncertainties due to cancellations and new requests by an exact dynamic approach. Second, for the complexly and diversely constituted state in which numbers and sets coexist, we develop a mapping technology such that a key consisting of only three numbers is obtained for each state. This largely facilitates the operations during the implementation. Furthermore, in solving via dynamic programming, to avoid the increasing computing complexity, we exploit critical structural properties of the state space and obtain an $O(1)$ looking-up method without any waste of memory space. Finally, two well-performed heuristics are proposed for larger instances.

The remainder of this paper is organized as follows. Section 2 presents the basic setting, the development of the mathematical model, and the solution method. In Section 2, mapping technology is described, structural properties of the state space are exploited, $O(1)$ looking-up method is described, and the experimental results are presented. In Section 4, two heuristic methods are proposed and evaluated. Section 5 concludes this paper with a discussion on future research directions.

2. A Markov Decision Process Model and Solution Method

2.1. The Basic Setting

The background setting for this study is stated as the following: a capacitated vehicle is used to collect customers' parcels within one pick-up tour. Specifically, the vehicle starts from a depot, then visits customers one by one, and finally returns to the depot. There is a set of customers whose pick-up services have been scheduled before the vehicle starts. Hence, unless they call for cancelling their appointments, those customers must be visited. We define them as "D-customer"s. For customers who do not have pre-scheduled appointments, they may call for a pick-up service during the courier's shift. However, they will be visited only in the situation where there is some remaining capacity in the vehicle. We define such customer as "P-customer". We define k as the vehicle's capacity for P-customers' request. Note that k can be zero and it is not necessary to keep k being same across tours. Actually, k depends on the vehicle's capacity limit and the total capacity of D-customers' parcels in the current tour. Certainly, if many unscheduled services are requested from P-customers, they will be dealt with in another tour. We say that each D-customer is always *active* until his service is finished or his service is cancelled, while we say that a P-customer will become *active* when his call for service is accepted and will become *inactive* after his service is finished.

Given that calls for cancellations from D-customers and for unscheduled services from P-customers occurring during the vehicle's shift, which are actually stochastic over time, we aim to find an effective strategy to support the vehicle on how to visit customers in response to those calls so that the total expected travel distance is minimized. Before presenting our MDP model, we give the following mild assumptions that support our model development. It is noted that the first three generally match the reality. Assumptions 4–5 are very common among most MDP models with stochastic arrivals. The last two assumptions are introduced to simplify the complexity of our MDP for a better tractability. Nevertheless, in most instances, the assumption 6 also matches the reality. For example, in China, most express firms have basic requirements on customers' parcel capacity [39].

1. Pick-up service is limited to a certain geographical district such that the location of customers can be known in advance. Moreover, the travel distance between any two customers or between one customer and the depot is certain in advance, which is proportional to the travel time by rate 1. Service time of each customer is also certain in advance.

2. When the vehicle finishes serving one customer, it will be instructed by the dispatcher as to which active customer it needs to visit next. At the same time, the next customer, either an active D-customer or an active P-customer, will be informed as well.

3. If there is no active customer after the vehicle finishes serving one customer, it will return to the depot.

4. Calls for pick-up service from one inactive P-customer follow a Poisson Process with rate λ_p .

5. During the vehicle's shift, every D-customer can call to cancel his appointment independently before it is informed as the upcoming customer. Once the service is cancelled, this D-customer will become inactive and will not call for service again. We assume that every D-customer's service cancellation calls are generated by a Poisson process with the same rate λ_d .

6. Every active P-customer's parcel has the same unit capacity. Thus, " k capacity for P-customers" means the vehicle can pick up at most k P-customers' parcels.

7. A cancelled appointment from a D-customer does not lead to a new pick-up service opening for any P-customers, i.e., k is fixed regardless of cancellations.

2.2. Modelling via Markov Decision Process

Taking reference from literature [40], where "A decision epoch begins when the vehicle arrives at a location and observes new customer requests", we indirectly adopt this idea and regulate that a decision is made when the vehicle arrives at one customer and finishes his pick-up service. Based on this basic thought, we divide the whole service tour into multiple stages and define one stage as the travel from one customer to his subsequent customer, including service for the latter. We restrict that the first stage starts at the depot and the last ends up at the depot. As similar as [40], we define that decision epoch begins when the vehicle finishes serving one customer. Clearly, under this definition, combined with the assumption 3, the P-customers' requests arriving in the last stage will only be serviced in another tour.

2.2.1. Definition of State, Action and Cost

- *State*: Still taking reference from literature [40], where one state contains information about the vehicle's current location, and each customer's current status, in our model, we also define that the state contains information about the vehicle's current location and each customer's status. In our problem, due to there being exactly two statuses for each customer (active and inactive), we only describe the customers who are in the "active" status. Since in this study two different types of customers are considered, we respectively express the active customers for each type. In addition, in literature [40], the current status of its main constraint "time" is also contained as a component in a state. In our problem, since the vehicle's capacity acts as the main constraint, so, combined with assumption 6, the state should also contain the current number of already accepted P-customers' service requests. To sum up, we define a *state* at the start of each stage as a quadruple, and it contains (a) the number of already accepted P-customers' service requests; (b) the set of currently active D-customers; (c) the customer whose service has been just finished, namely the position where the vehicle is; and (d) the set of currently active P-customers. Clearly, such defined state satisfies the Markov property. The state space consists of all possible states within all stages.
- *Action*: In response to each state, we define its *action* as the selection of an active customer that will be directly visited by the vehicle next. Obviously, from the current state, the subsequent state will be resulted from this action and some new arrival calls from both active D-customers (for cancellations) and inactive P-customers (requesting for unscheduled services) during the time of the nurse's travelling and service.
- *Cost*: We define the *cost* of a state as the expected value of the future contribution to the tour length, assuming that all subsequent actions are optimal.

2.2.2. Notations

At the beginning of the whole service tour:

- D the set of customers whose pick-up services have been pre-scheduled in advance, $|D| = d$;
 P the set of all potential pick-up requesting customers, $|P| = p$;
 d_{ij} travel distance (time) between customer i and j ;
 t_i service time needed by customer $i \in D \cup P$;
 λ_d Poisson rate of each active D-customer generating calls for service cancellations;
 λ_p Poisson rate of each P-customer generating calls for unscheduled services;
 k the vehicle's capacity for P-customers' pick-up service.

At the beginning of one stage:

- location* the customer whose service has just been finished. For the first stage, we define its *location* to be o , which denotes the depot;
 $Dset$ the set of all active D-customers at the present moment, $Dset \in D$;
 $Pset$ the set of all active P-customers at the current moment, $Pset \in P$;
 c the number of all pick-up requests already accepted from P-customers;
 $s = (c, Dset, location, Pset)$ one state to describe the historic aggregated situation;
 $\Delta P = P \setminus Pset$ the set of current inactive P-customers;
 $\Delta D = Dset$ the set of current active D-customers;
 $\Delta D^j = Dset \setminus \{j\}, j \in Dset$ the subset of $Dset$ just without element j ;
 h the number of active D-customers who call to cancel their scheduled services in the current stage;
 $\Delta D_h (|\Delta D_h| = h, h \leq |\Delta D|)$ one subset of ΔD in the size of $h, 0 \leq h \leq |Dset|$;
 $\Delta D_h^j (|\Delta D_h^j| = h, h \leq |\Delta D^j|)$ one subset of ΔD^j in the size of $h, 0 \leq h \leq |Dset| - 1, j \in Dset$;
 l the number of inactive P-customers whose calls for pick-up services are accepted in the current stage;
 $\Delta P_l (|\Delta P_l| = l, l \leq |\Delta P|)$ one subset of ΔP in the size of l ;
 G_{P_l} the collection of all ΔP_l s;
 G_{D_h} the collection of all ΔD_h s;
 $G_{D_h^j}$ the collection of all ΔD_h^j s;
 C_s the cost of state s ;
 s^a the optimal action to state s , which is an active customer from $Pset \cup Dset$ to be visited next.

2.2.3. State Transfer Equation and Probability

A state can be expressed as $(c, Dset, location, Pset)$. Given state $s = (c, Dset, i, Pset)$ and an action $j, j \in Pset \cup Dset$, the subsequent state will depends on both h , the number of active D-customers cancelling their scheduled services, and l , the number of requests accepted for unscheduled services from inactive P-customers, during the current stage. Thus, the State Transfer Equation can be obtained as Formulas (1)–(3). Based on those equations, the action s^a is just the customer $j (j \in Dset \cup Pset)$ which makes C_s achieved. The initial state is $(0, D, o, \emptyset)$ and its cost is the objective value:

$$C_s = \min_{j \in Dset \cup Pset} \{d_{ij} + E(j)\}, \quad (1)$$

$$E(j) = \begin{cases} \sum_{h=0}^{|Dset|-1} \sum_{l=0}^{k-c} E_{h,l}(j), j \in Dset, \\ \sum_{h=0}^{|Dset|} \sum_{l=0}^{k-c} E_{h,l}(j), j \in Pset, \end{cases} \quad (2)$$

$$E_{h,l}(j) = \begin{cases} P_d^1 \cdot P_p \cdot \sum_{\Delta D_h^j \in G_{P_h^j}} \sum_{\Delta P_l \in G_l} C_{s_1}, j \in Dset, \\ P_d^2 \cdot P_p \cdot \sum_{\Delta D_h \in G_{P_h}} \sum_{\Delta P_l \in G_l} C_{s_2}, j \in Pset. \end{cases} \quad (3)$$

In the formulas above, $s_1 = (c + l, Dset \setminus \{j\} \cup \Delta D_{h,j}^l, Pset \cup \Delta P_l)$, and $s_2 = (c + l, Dset \setminus \Delta D_{h,j}, Pset \cup \Delta P_l \setminus \{j\})$. $P_d^1 = P_r^1(h|s, j)$, $P_d^2 = P_r^2(h|s, j)$, which gives the probabilities that given state s and action j, h specific active D-customers cancel their scheduled services during the time T_{ij} , $T_{ij} = d_{ij} + t_j$. $P_m = P_r(l|s, j)$, which gives the probability that given state s and action j, l specific inactive P-customers' calls for unscheduled services are accepted during the time T_{ij} . Next, we will discuss how to compute P_d^1 , P_d^2 , and P_p , respectively, in two cases of $l < k - c$ and $l = k - c$.

(i) In the case of $l < k - c$

Clearly, for a given s , $|\Delta P|$ is certain. Since the capacity has not been reached ($c + l < k$), there is no P-customer's call being rejected. Thus, those l particular P-customers generate calls that are all accepted. Other $|\Delta P| - l$ P-customers have not generated calls. Thus, we can get the probability that during T_{ij} , l P-customers are accepted is

$$P_p = (1 - e^{-\lambda_p T_{ij}})^l (e^{-\lambda_p T_{ij}})^{|\Delta P| - l}, l < k - c \quad (4)$$

in which the term $(1 - e^{-\lambda_p T_{ij}})$ is the probability of one P-customer generating at least one call during T_{ij} . Since no limit on how many active D-customers can cancel their services, there are exactly h such customers generating calls. Thus, we can get P_d^1 and P_d^2 as

$$P_d^1 = (1 - e^{-\lambda_p T_{ij}})^h (e^{-\lambda_p T_{ij}})^{|\Delta P| - 1 - h}, j \in Dset, \quad (5)$$

$$P_d^2 = (1 - e^{-\lambda_p T_{ij}})^h (e^{-\lambda_p T_{ij}})^{|\Delta P| - h}, j \in Pset. \quad (6)$$

(ii) In the case of $l = k - c$

We use $\mathcal{B}$ to denote the event of at least l inactive P-customers generating calls during T_{ij} . Its complementary event $\bar{\mathcal{B}}$ can be easily obtained as the sum of l different sub-event $\bar{\mathcal{B}}_q$ s ($0 \leq q \leq l - 1$). $\bar{\mathcal{B}}_q$ represents during T_{ij} there are exactly q inactive P-customers generating calls. Thus, the probability of $\bar{\mathcal{B}}_q$ can be obtained as

$$P_r(\bar{\mathcal{B}}_q) = \binom{|\Delta P|}{q} (1 - e^{-\lambda_p T_{ij}})^q (e^{-\lambda_p T_{ij}})^{|\Delta P| - q}, 0 \leq q \leq l - 1. \quad (7)$$

Then, we can get the probability of event $\mathcal{B}$ as

$$P_r(\mathcal{B}) = 1 - P_r(\bar{\mathcal{B}}) = 1 - \sum_{q=0}^{l-1} P_r(\bar{\mathcal{B}}_q). \quad (8)$$

Thus, in the case of $l = k - c$, we can get P_m as

$$P_m = \frac{P_r(\mathcal{B})}{\binom{|\Delta P|}{l}}. \quad (9)$$

Clearly, P_d^1 and P_d^2 are the same as the case of $l < k - c$.

Time complexity on probability computing Since in the implementation, the bit width of each variable is always subject to computer's basic restriction, we assume that time complexity on one multiplication operation and one division operation is respectively $O(1)$. In addition, we assume the time complexity on computing e^x is $O(1)$. Thus, given state s and its alternative decision $j, j \in Dset \cup Pset$, on the assumption that all values of $\binom{|\Delta P|}{s}, 0 \leq s \leq k$, are given, time consumed on computing all related P_d^1 s and P_d^2 s is $O(d)$ and time consumed on computing all related P_p s is $O(k)$.

2.3. Solving via Dynamic Programming

In this paper, the cost of a state has been defined as the expected remaining length of the service tour given that all subsequent actions are optimal. The goal is to find the costs and optimal actions at all states, which will define the optimal dynamic strategy.

2.3.1. Algorithm

We divide all states into five categories and then solve the problem using dynamic programming in six steps. The whole frame is stated in Algorithm 1 (A1). The detailed operations for the last four steps are described in Algorithm 2 (A2).

Algorithm 1 Solving by Dynamic Programming.

- Step 0:** Compute $2^j, j = 0, 1, \dots, d-1$ and $(m_j^{-g}), j = 0, 1, \dots, \min\{m-g, k\}, 0 \leq g < m$, which will be multiply used in later steps.
- Step 1:** For s with $|Dset \cup Pset| = 0$, let $s^a := o$ and $C_s := d_{location,o}$.
- Step 2:** For s with $c = k, |Dset| = 0$ and $|Pset| \neq 0$, compute C_s and s^a .
- Step 3:** For s with $c = k, |Dset| \neq 0$, compute C_s and s^a .
- Step 4:** For s with $c < k, location \neq o$ and $|Dset \cup Pset| \neq 0$, compute C_s and s^a .
- Step 5:** For the initial state $(0, D, o, \emptyset)$, compute its cost and action. The cost is the optimal objective value.
-

From A1, we can see that, in Step 1, states' costs and actions can be obtained directly, which will act as the initial values for the dynamic programming process. Probability of service being cancelled needs to be taken into account from Step 3. Probability of new service request needs to be taken into account from Step 4. In Step 5, there is only one initial state $(0, D, o, \emptyset)$ that needs to be computed. From A2, we can see that a state's cost cannot be directly obtained but depends on other already obtained states' costs which are stored in the memory space.

Algorithm 2 Operations for a given state in Steps 2–5.

- Input: $s = (c, Dset, i, Pset): |Dset| + |Pset| \neq 0, c \leq k$
1. For any $j \in Dset$
 2. Compute all $P_d^1 s, 0 \leq h < |Dset|$ and all $P_p s, 0 \leq l \leq k - c$
 3. For $h = |Dset| - 1$ to $h = 0$
 4. For $l = 0$ to $l = k - c$
 5. For each $\Delta D_h^j \in \Delta D_j = Dset \setminus \{j\}$
 6. For each $\Delta P_l \in \Delta P = P \setminus \{Pset\}$
 7. Look up the cost of state $s^1 = (c + l, Dset \setminus (\Delta D_h^j \cup \{j\}), j, Pset \cup \Delta P_l)$, denoted as $E_{\Delta D_h^j, \Delta P_l}(j)$
 8. End for
 9. Obtain $E_{\Delta D_h^j}(j) = \sum_{\Delta P_l} E_{\Delta D_h^j, \Delta P_l}(j)$
 10. End for
 11. Obtain $P_r^1(h|s, j)P_r(l|s, j) \sum_{\Delta D_h^j} E_{\Delta D_h^j}(j)$, denoted as $E_{h,l}(j)$
 12. End for
 13. Obtain $\sum_l E_{h,l}(j)$, denoted as $E_h(j)$
 14. End for
 15. Obtain $E(j) = \sum_h E_h(j)$
 16. End for
 17. For any $j \in Pset$
 18. Compute all $P_d^2 s, 0 \leq h \leq |Dset|$ and all $P_p s, 0 \leq l \leq k - c$
 19. For $h = |Dset|$ to $h = 0$
 20. For $l = 0$ to $l = k - c$
-

Algorithm 2 Cont.

-
21. For each $\Delta D_h \in \Delta D = Dset$
 22. For each $\Delta P_l \in \Delta P = P \setminus \{Pset\}$
 23. Look up the cost of state $s^2 = (c + l, Dset \setminus \Delta D_h, j, Pset \cup \Delta P_l \setminus \{j\})$, denoted as $E_{\Delta D_h, \Delta P_l}(j)$
 24. End for
 25. Obtain $E_{\Delta D_h}(j) = \sum_{\Delta P_l} E_{\Delta D_h, \Delta P_l}(j)$
 26. End for
 27. Obtain $P_r^2(h|s, j) P_r(l|s, j) \sum_{\Delta D_h} E_{\Delta D_h}(j)$, denoted as $E_{h,l}(j)$
 28. End for
 29. Obtain $\sum_l E_{h,l}(j)$, denoted by $E_h(j)$
 30. End for
 31. Obtain $E(j) = \sum_h E_h(j)$
 32. End for
 33. Obtain $C_s = \min \{d_{ij} + E(j) : j \in Dset \cup Pset\}$ and s^a as the j which makes C_s achieved.
-

2.3.2. Complexity Analysis

In Step 0, $2^j s$, $j = 0, 1, \dots, d-1$, can be computed in $O(d)$ and $\binom{p-g}{j} s$, $j = 0, 1, \dots, \min\{p-g, k\}$, $0 \leq g < p$, can be computed in $O(pk)$, so the time complexity in this step is $O(d + pk)$. In Step 1, since there are totally at most $(k+1)(d+p)$ states, the time consumed is $O((k+1)(d+p))$.

Currently, we suppose time consumed for looking up one already obtained state's cost in memory space is $O(1)$, which will be proved in Section 3. In Step 2, given a state, its time consumed is $O(|Pset|)$. Due to $|Pset|$ can vary from 1 to k and the number of states with $|Pset| = j$ ($1 \leq j \leq k-1$) is $d\binom{p}{j} + (j+1)\binom{p}{j+1}$ and the number of states with $|Pset| = k$ is $d\binom{p}{k}$, the total time consumed in Step 2 is:

$$\begin{aligned}
 & \sum_{j=1}^{k-1} O(j) \left[d\binom{p}{j} + (j+1)\binom{p}{j+1} \right] + O(k)p\binom{p}{k} \\
 &= O \left(dp \sum_{j=0}^{k-1} \binom{p-1}{j} \right) + O \left(p(p-1) \sum_{j=0}^{k-2} \binom{p-2}{j} \right) \\
 &= O(dp^k) + O(p(p-1)^{k-1}) = O((d+1)p^k).
 \end{aligned} \tag{10}$$

In Step 3, given a state with $(c, Dset, i, Pset)$, $c = k$ and $|Dset| \neq 0$, under a specific action $j \in Dset \cup Pset$, the time consumed on computing probability is $O(|Dset|)$, and there are totally $O(2^{|Dset|})$ times looking-up-memory operations. Thus, clearly the time consumed on computing probability is dominated and hence, given one state, time consumed for its cost and action is:

$$(|Pset| + |Dset|)O(2^{|Dset|}) = O((d+k)2^d). \tag{11}$$

Since, in this step, there are totally no more than $(d+p)(p+1)^k 2^d$ states, so the time complexity in Step 3 is:

$$\begin{aligned}
 & (d+p)(p+1)^k 2^d \cdot O((d+k)2^d) \\
 &= O((d+p)(d+k)(p+1)^k 4^d).
 \end{aligned} \tag{12}$$

In Step 4, given a state $(c, Dset, i, Pset)$, $|Dset| + |Pset| \neq 0$, $c < k$, similar as analysis on Step 3, we can obtain that time consumed on computing its cost and action is:

$$\begin{aligned}
& (|Dset| + |Pset|) O \left(\sum_{h=0}^{|Dset|} \binom{|Dset|}{h} \sum_{l=0}^{k-c} \binom{|\Delta P|}{l} \right) \\
& = (|Dset| + |Pset|) O \left(2^{|Dset|} (|\Delta P| + 1)^{k-c} \right) \\
& = O \left((d+k)(p+1)^k 2^d \right).
\end{aligned} \tag{13}$$

Since, in Step 4, there are totally no more than $k(d+p)(p+1)^k 2^d$ states, time complexity in Step 4 is $O(k(d+p)(d+k)(p+1)^{2k} 4^d)$. Because, in Step 5, there is only the initial state needed to be computed and the process is similar as for one given state in Step 4, time consumed is completely dominated by the time complexity in Step 4.

To sum up, including the time consumed for computing the probability of states' transfer and time consumed for looking up the already computed states' cost ($O(1)$), we can get the total time complexity by using dynamic programming to solve this problem is $O(k(d+k)(d+p)(p+1)^{2k} 4^d)$.

3. Implementation

3.1. State's Key and Its Computation

Clearly, to look up one already obtained state's cost in memory space includes two steps; one is to design its key and the other is to look for the record in the memory by using the key. Next, we first define state's key and describe how to obtain the key. Second, based on the key, we give a specific storage method for all states' costs and actions, and then describe how to look for a specific state's cost and action in memory. Finally, time complexity for all these operations will be analysed.

We index D-customers from 1 to d and P-customers from $d+1$ to $d+p$. In order to make the description more clear, in this section, we give a specific example and always exemplify our presentation through this example:

Example : $D = \{1, 2\}$, $P = \{3, 4, 5\}$, $k = 2$.

For one given state $(c, Dset, location, Pset)$, we give a triple $i_1|i_2|i_3$, in which $i_1 = c$, $i_2 = \sum_{j \in Dset} 2^{j-1}$ (noting that $\sum_{\emptyset} = 0$) and i_3 is related with state's components of *location* and *Pset*. Clearly, for any two states whose *Dsets* are complementary, the sum of their i_2 s is $2^d - 1$. In order to save computing time, in our implementation, we compute an array of numbers consisting of 2^j , $j = 0, 1, \dots, d-1$ at a time in advance with time consumed $O(d)$. Then, for any given state, its i_2 can be obtained by no more than d times additional operations.

It is easy to see that, for any two different states, if they have the same *Dset* and c , the unions of *location* and *Pset* must be different, but if they have different *Dset* or c , the unions of *location* and *Pset* are probably the same. Thus, first we gather all possible different unions of *location* and *Pset* probably appearing in all states, and arrange them by a certain rule, so that a one-to-one correspondence between these unions and natural numbers can be formed. Then, we take these natural numbers as states' i_3 values. Note that a *possible union* is the union in which the number of P-customers is no more than the capacity k .

All possible unions of *location* and *Pset* are arranged according to the rules: (1) unions with D-customers as their *locations* are indexed earlier than unions with P-customers as their *locations*; (2) unions with a smaller size of *Pset* are indexed earlier; (3) unions with smaller index of *location* are indexed earlier; and (4) unions with the same *location* and the same size of *Pset* are indexed by applying alphabet order on their *Psets*. Under this rule, we can see that there are totally $\sum_{j=0}^k d \binom{p}{j} + \sum_{j=1}^k j \binom{p}{j}$ different indices for i_3 , varying from 0 to $\sum_{j=0}^k d \binom{p}{j} + \sum_{j=1}^k j \binom{p}{j} - 1$.

Example : Indices of i_2 and i_3 are respectively shown in Tables 1 and 2.

Table 1. i_2 : Indices for $Dset$.

Index	0	1	2	3
$Dset$	$\emptyset$	$\{1\}$	$\{2\}$	$\{1,2\}$

Table 2. i_3 : Indices for unions of $location$ and $Pset$.

Index	Union	Index	Union	Index	Union
0	$(1, \emptyset)$	8	$(1, \{3,4\})$	16	$(5, \emptyset)$
1	$(2, \emptyset)$	9	$(1, \{3,5\})$	17	$(3, \{4\})$
2	$(1, \{3\})$	10	$(1, \{4,5\})$	18	$(3, \{5\})$
3	$(1, \{4\})$	11	$(2, \{3,4\})$	19	$(4, \{3\})$
4	$(1, \{5\})$	12	$(2, \{3,5\})$	19	$(4, \{3\})$
5	$(2, \{3\})$	13	$(2, \{4,5\})$	20	$(4, \{5\})$
6	$(2, \{4\})$	14	$(3, \emptyset)$	21	$(5, \{3\})$
7	$(2, \{5\})$	15	$(4, \emptyset)$	22	$(5, \{4\})$

According to the arranging rules, the number of unions with D-customer *locations* and specific size l ($l \leq k$) of $Pset$ is $d \binom{p}{l}$. The number of states with P-customers *locations* and specific size l ($l < k$) of $Pset$ is $(l+1) \binom{p}{l+1}$. Therefore, given a union of *location* and $Pset$ as $(i, \{j_1, j_2, \dots, j_l\})$, i_3 's lower bound denoted by $i_{3,L}$ can be quickly obtained by Formula (14). Obviously, if $l = 0$, the lower bound is just the value of i_3 :

$$i_{3,L} = \begin{cases} d \sum_{j=0}^{l-1} \binom{p}{j} + (i-1) \binom{p}{l}, & i \leq d, \\ d \sum_{j=0}^k \binom{p}{j} + \sum_{j=1}^{l-1} j \binom{p}{j} + (i-d-1) \binom{p-1}{l}, & i > d. \end{cases} \quad (14)$$

Then, based on the 4th arranging rule, we get the exact i_3 value through an iterative method, which is based on the assumption that all customers in $Pset$ are arranged increasingly by their indices. As a preliminary, first, we give an algorithm to compute the order for a specific combination given that all $\binom{p}{l}$ combinations are lexicographically ordered. This algorithm is named as *kSubsetLexRank* Algorithm 3 (A3) proposed by Kreher and Stinson [41].

Algorithm 3 *kSubsetLexRank*.

1. Input p, l , and combination $(j_1, j_2, \dots, j_l)$ in which $j_1 < j_2 < \dots < j_l$
 2. Let $j_0 = 0$ and $r = 0$
 3. For i from 1 to l
 4. If $j_{i-1} + 1 \leq j_i - 1$
 5. For g from $j_{i-1} + 1$ to $j_i - 1$:
 6. $r = r + \binom{p-g}{l-i}$
 7. End for
 8. End if
 9. End for
 10. Output r
-

Then, by using *kSubsetLexRank* Algorithm, the method to find the exact i_3 is given as Algorithm 4 (A4). In order to save computing time, similar to computation on i_2 , during the implementation, we also compute values of $\binom{p-g}{j}, j = 0, 1, \dots, \min\{p-g, k\}, 0 \leq g < p$, at a time in advance with time consumed $O(pk)$.

Algorithm 4 Exact i_3 Computing Algorithm.

-
1. Input: $(i, [j_1, j_2 \cdots j_l])(l \neq 0), d$ and p
 2. Set $i_3 = i_{3,L}$. Set $Pset[l] = \{j_1, j_2 \cdots j_l\}$ and $Pset'[l] = \emptyset$.
 3. If $i > d$, go to 5; else go to 4.
 4. $i_3 = i_3 + kSubsetLexRank(p, l, Pset[l])$, go to 9
 5. For g from 0 to l
 6. If $Pset[g] < i$, $Pset'[g] = Pset[g]$; else $Pset'[g] = Pset[g] - 1$, end if;
 7. End for
 8. $i_3 = i_3 + kSubsetLexRank(p - 1, l, Pset'[l])$;
 9. Output i_3 .
-

Complexity for Computing $i_1|i_2|i_3$ We distinguish time consumed for one-time preparing computation in advance for all states and time consumed in process on each state. As stated before, time consumed on all prepared computation is $O(d + pk)$. Based on these prepared values, given one state, its i_1 can be directly obtained and its i_2 can be obtained in $O(d)$. Lower bound of i_3 can be computed in $O(k)$ and since time consumed on A3 is $O(p)$, so time consumed on A4 is also $O(p)$. Hence, time consumed on i_3 is $O(p)(k \leq p)$. To sum up, time complexity on computing each given state's key is $O(d + p)$.

Next, based on A2, we will show that time consumed on computing states' key values is dominated by the complexity $O((d + k)(p + 1)^k 2^d)$ obtained in Formula (13). In A2, taking one state s as input, computation on state's key values includes two aspects. The first is the key of this given state such that, by using the key, the given state's cost and action obtained in Step 33 can be stored in the memory space. The second are the keys of all possible states which could be resulted from the given state, such that, by using these keys, these states' already known costs can be looked up from the memory space in Step 7 and Step 23. Since the first computation can be done with $O(d + p)$ after the given state's cost and action has been obtained, obviously the time consumed is dominated by $O((d + k)(p + 1)^k 2^d)$. For the second computation, since the low bound of one state's i_3 is only related with the *location* element and the size of *Pset*, in A2, given a state $(c, Dset, i, Pset)$, for each alternative decision $j, j \in Dset$ (or $j \in Pset$), we can compute all possible subsequent states' lower bound $i_{3,L}$ s in advance in Step 2 (or Step 18). Then, since in our implementation $\Delta D_{i,s}$ s in Step 5 (or $\Delta D_{i,s}$ s in Step 21) and $\Delta P_{i,s}$ s in Step 6 (or Step 22) are lexicographically generated, in Step 7 (or Step 23), one state's i_2 and i_3 can directly obtained from its close precedent state with constant operations. Thus, clearly, the complexity $O((d + k)(p + 1)^k 2^d)$ also dominates the time consumed for the second computation.

Until now, for the defined state with four components in terms of two numbers and two sets, we have successfully designed a key only consisted of three numbers. Moreover, the computation on the key does not make any improvement to the complexity. Thus, we can say that the key can be obtained with $O(1)$ time consumed. If we can directly use the key to look up (store) a state's record (cost and action) in the memory space without any additional computation, clearly an $O(1)$ looking-up method for one state's record has been obtained. Obviously, we can achieve it by using a three-dimensional matrix to store states' records, where the key $i_1|i_2|i_3$ can be taken respectively as indices in three dimensions. However, such a storage method will result in a huge waste of the memory space, as exemplified by the following example. Hence, in order to obtain an $O(1)$ looking-up(storage) method without any waste of memory space, we need to propose a specific storage method.

Example: We use a three-dimensional matrix to store states' records and obtain the storage result as shown in Tables A1 and A2 in Appendix A, where one state's record is represented by the state itself in the form of $(c, Dset, location, Pset)$, gray cells are idle memory and cells between two red lines are used for states with P-customer *locations*. From Tables A1 and A2,

we can see that this direct storage method will result in 56.88% of applied memory being wasted.

3.2. Structural Properties and Storage Method

In this section, we first exploit the structural properties of the state space. Then, based on the properties, we propose a special storage method such that an $O(1)$ time consumed looking-up method still holds and moreover no memory is wasted at all. The example in the last section is still used to exemplify our presentation.

3.2.1. Structural Properties

Observation 1. For each state, there is only one triple $i_1|i_2|i_3$, which can be obtained from this state.

It is easy to see Observation 1 by the definition of triple $i_1|i_2|i_3$. Observation 1 is also the reason why we take $i_1|i_2|i_3$ as the state's key.

Observation 2. Not every triple $i_1|i_2|i_3$ is the key of one state. For some triples, there are no states corresponding to them.

It is also easy to see Observation 2. In our example, we can see that although i_1 can vary from 0 to 2, i_2 can vary from 0 to 3, and i_3 can vary from 0 to 22, there are no states corresponding to triples in the form of $i_1 = 0 | i_2 | i_3 \geq 2$.

For one triple $i_1|i_2|i_3$, we say that it is valid if there is one state corresponding to it; otherwise, we say that it is invalid. In addition, we define a triple as D-triple if its i_3 is obtained from a union where the location is a D-customer, while we define a triple as P-triple if its i_3 is obtained from a union where the location is a P-customer.

Observation 3. For any two D-triples which have complimentary i_2 (the sum of their i_2 s is $2^p - 1$) and the same i_1 and i_3 , exactly one of them is valid.

Observation 4. For all P-triples with the same i_1 and i_3 , they are either completely valid or completely invalid.

Observation 5. All P-triples with $i_1 = 0$ are all invalid. In addition, all P-triples with $i_1 = k$ are all valid.

Observations 3–5 can be easily seen from the definition of $i_1|i_2|i_3$. We can also further verify them through the example as shown in Tables A1 and A2 (Appendix A).

Lemma 1. The total number of valid D-triples with $i_1 = i$ and $i_1 = k - 1 - i$, $0 \leq i \leq k - 1$, is no more than the number of valid D-triples with $i_1 = k$.

Proof of Lemma 1. The number of all D-triples (valid and invalid) with $i_1 = k$ is $d \cdot 2^d \cdot \sum_{j=0}^k \binom{p}{j}$. In addition, the total number of all D-triples with $i_1 = i$ and $i_1 = k - 1 - i$ ($0 \leq i \leq k - 1$) is $d \cdot 2^d \cdot \sum_{j=0}^i \binom{p}{j} + d \cdot 2^d \sum_{j=0}^{k-1-i} \binom{p}{j}$. Clearly, $\sum_{j=0}^i \binom{p}{j} + \sum_{j=0}^{k-1-i} \binom{p}{j} < \sum_{j=0}^k \binom{p}{j}$. Combined with Observation 3, this Lemma holds. $\square$

Lemma 2. The total number of valid P-triples with $i_1 = i$ and $i_1 = k - i$, $1 \leq i \leq k - 1$, is no more than the number of valid P-triples with $i_1 = k$.

Proof of Lemma 2. The number of all P-triples (valid and invalid) with $i_1 = k$ is $\sum_{j=1}^k j \binom{p}{j}$. The total number of all P-triples with $i_1 = i$ and $i_1 = k - i$ ($1 \leq i \leq k - 1$) is $\sum_{j=1}^{k-i} j \binom{p}{j} + \sum_{j=1}^i j \binom{p}{j}$. Clearly, $\sum_{j=1}^{k-i} j \binom{p}{j} + \sum_{j=1}^i j \binom{p}{j} \leq \sum_{j=1}^k j \binom{p}{j}$. Combined with Observation 4, this lemma holds. $\square$

3.2.2. Storage Method

We store valid D-triples and valid P-triples, respectively. Each are stored within a three-dimensional matrix. First, we describe how to store D-triples and then describe how to store P-triples. Note that “store valid triple” always means “store the record (cost and action) of the state corresponding to this triple”. Thus, in the later part of this paper, we do not distinguish them.

To store valid D-triples

We give three numbers $D_{1,d}$, $D_{2,d}$ and $D_{3,d}$, which are obtained by the following formula. Then, we show all valid D-triples can be stored in a three-dimensional matrix in the size of $D_{1,d} \times D_{2,d} \times D_{3,d}$:

$$D_{1,d} = \begin{cases} 1 + \frac{k}{2}, & k \text{ is even,} \\ 1 + \frac{k+1}{2}, & k \text{ is odd,} \end{cases} \quad D_{2,d} = 2^{d-1}, \quad D_{3,d} = d \sum_{j=0}^k \binom{p}{j}. \quad (15)$$

We use (I_1^d, I_2^d, I_3^d) to denote the indices of one cell in this matrix.

In the case of k being odd For cells in the indices of $(I_1^d, :, :), 0 \leq I_1^d < \frac{k-1}{2}$, D-triples with $i_1 = I_1^d$ and $i_1 = k - 1 - I_1^d$ are stored. For cells in the indices of $(I_1^d = \frac{k-1}{2}, :, :)$, D-triples with $i_1 = \frac{k-1}{2}$ are stored. For cells in the indices of $(I_1^d = \frac{k+1}{2}, :, :)$, D-triples with $i_1 = k$ are stored. For cells in the indices of $(I_1^d, I_2^d, :), 0 \leq I_2^d < 2^{d-1}$, D-triples with $i_2 = I_2^d$ and $i_2 = 2^d - 1 - I_2^d$ are stored (based on *Observation 3*). In detail, given one valid D-triple $i_1|i_2|i_3$, it will be stored according to the following rules:

- If $i_1 \leq \frac{k-1}{2}$ and $i_2 < 2^{d-1}$, it is stored in the cell of (i_1, i_2, i_3) .
- If $i_1 \leq \frac{k-1}{2}$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(i_1, 2^d - 1 - i_2, i_3)$.
- If $\frac{k-1}{2} < i_1 < k$ and $i_2 < 2^{d-1}$, it is stored in the cell of $(k - 1 - i_1, 2^{d-1} - 1 - i_2, D_{3,d} - 1 - i_3)$.
- If $k > i_1 > \frac{k-1}{2}$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(k - 1 - i_1, i_2 - 2^{d-1}, D_{3,d} - 1 - i_3)$.
- If $i_1 = k$ and $i_2 < 2^{d-1}$, it is stored in the cell of $(\frac{k+1}{2}, i_2, i_3)$.
- If $i_1 = k$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(\frac{k+1}{2}, 2^d - 1 - i_2, i_3)$.

In the case of k being even For cells in the indices of $(I_1^d, :, :), 0 \leq I_1^d < \frac{k}{2}$, D-triples with $i_1 = I_1^d$ and $i_1 = k - 1 - I_1^d$ are stored. For cells in the indices of $(I_1^d = \frac{k}{2}, :, :)$, D-triples with $i_1 = k$ are stored. For cells in the indices of $(I_1^d, I_2^d, :), 0 \leq I_2^d < 2^{d-1}$, D-triples with $i_2 = I_2^d$ and $i_2 = 2^d - 1 - I_2^d$ are stored (based on *Observation 3*). In detail, given one valid D-triple $i_1|i_2|i_3$, it will be stored according to the following rules:

- If $i_1 < \frac{k}{2}$ and $i_2 < 2^{d-1}$, it is stored in the cell of (i_1, i_2, i_3) .
- If $i_1 < \frac{k}{2}$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(i_1, 2^d - 1 - i_2, i_3)$.
- If $\frac{k}{2} \leq i_1 < k$ and $i_2 < 2^{d-1}$, it is stored in the cell of $(k - 1 - i_1, 2^{d-1} - 1 - i_2, D_{3,d} - 1 - i_3)$.
- If $\frac{k}{2} \leq i_1 < k$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(k - 1 - i_1, i_2 - 2^{d-1}, D_{3,d} - 1 - i_3)$.
- If $i_1 = k$ and $i_2 < 2^{d-1}$, it is stored in the cell of $(\frac{k}{2}, i_2, i_3)$.
- If $i_1 = k$ and $i_2 \geq 2^{d-1}$, it is stored in the cell of $(\frac{k}{2}, 2^d - 1 - i_2, i_3)$.

It is easy to see that Lemma 1 and *Observation 1* guarantee that there is no overlapped cell in this matrix. Thus, it is clear that all valid D-triples can stored in a three-dimensional matrix of $D_{1,d} \times 2^{d-1} \times D_{3,d}$.

Example: $D_{1,d} = 2$, $D_{2,d} = 2$ and $D_{3,d} = 14$. k is even. The storage result of all valid D-triples in the matrix of 2 by 2 by 14 is shown in Table A3 (Appendix A), where cells stored by operation g have white background, cells stored by operation h have blue background, cells stored by operation i have green background, cells stored by operation j have red background, cells stored by operation k have yellow background, and cells stored by operation l have purple background. The gray cells are applied but wasted memory space.

To store valid P-triples

We give three numbers $D_{1,p}$, $D_{2,p}$ and $D_{3,p}$, which are obtained by the following formula. Then, we show all valid P-triples can be stored in a three-dimensional matrix in the size of $D_{1,p} \times D_{2,p} \times D_{3,p}$:

$$D_{1,p} = \begin{cases} 1 + \frac{k}{2}, & k \text{ is even,} \\ \frac{k+1}{2}, & k \text{ is odd,} \end{cases} \quad D_2^p = 2^d, \quad D_{3,p} = \sum_{j=0}^k j \binom{p}{j}. \quad (16)$$

We use (I_1^p, I_2^p, I_3^p) to denote the indices of one cell in this matrix.

In the case of k being odd For cells in the indices of $(I_1^p, :, :), 0 \leq I_1^p < \frac{k+1}{2}$, P-triples with $i_1 = I_1^p$ and $i_1 = k - I_1^p$ are stored. In detail, given one valid P-triple $i_1|i_2|i_3$, it will be stored according to the following rules: (Note: $i'_3 = i_3 - D_{3,d}$)

- m) If $i_1 < \frac{k+1}{2}$, it is stored in the cell of (i_1, i_2, i'_3) .
- n) If $i_1 \geq \frac{k+1}{2}$, it is stored in the cell of $(k - i_1, i_2, D_{3,p} - 1 - i'_3)$.

In the case of k being even For cells in the indices of $(I_1^p, :, :), 0 \leq I_1^p < \frac{k}{2}$, P-triples with $i_1 = I_1^p$ and $i_1 = k - I_1^p$ are stored. For cells in the indices of $(I_1^p = \frac{k}{2}, :, :)$, valid P-triples with $i_1 = \frac{k}{2}$ are stored. In detail, given one valid P-triple $i_1|i_2|i_3$, it will be stored according to the following rules:

- o) If $i_1 \leq \frac{k}{2}$, it is stored in the cell of (i_1, i_2, i'_3) .
- p) If $i_1 > \frac{k}{2}$, it is stored in the cell of $(k - i_1, i_2, D_{3,p} - 1 - i'_3)$.

It is easy to see that *Observation 4*, *Observation 5* and *Lemma 2* guarantee there is no overlapped cell in this matrix. Thus, it is clear that all valid P-triples can be stored in a three-dimensional matrix of $D_{1,p} \times 2^d \times D_{3,p}$.

Example: $D_{1,p} = 2$, $D_{2,p} = 4$ and $D_{3,p} = 9$. k is even. The storage result of all valid P-triples in the matrix of 2 by 4 by 9 is shown in Table A4 (Appendix A), where cells stored by operation o have *white* background, and cells stored by operation p have *green* background. The gray cells are applied but wasted memory space.

Until now, from this storage method, we can easily get an $O(1)$ looking-up method which can largely save the memory space. However, there is still some space being wasted (as the gray cells in Tables A3 and A4 in Appendix A within our example). Thus, in order to achieve the situation of no memory space being wasted, we will do the following operations.

Operations: We set the two matrices $D_{1,d} \times D_{2,d} \times D_{3,d}$ and $D_{1,p} \times D_{2,p} \times D_{3,p}$ to have a dynamic size $D'_{3,d}$ and $D'_{3,p}$ in their third dimensions. $D'_{3,d}(D'_{3,p})$ is set to be the function of the index $I_1^d(I_1^p)$ in the first dimension as follows:

$$D'_{3,d} = \begin{cases} p \sum_{j=0}^k \binom{p}{j}, & I_1^p = D_{1,d} - 1, \\ p \sum_{j=0}^{I_{1,d}} \binom{p}{j} + p \sum_{j=0}^{k-1-I_{1,d}} \binom{p}{j}, & I_1^d < D_{1,d} - 1 \ \& \ I_1^p \neq k - 1 - I_{1,d}, \\ p \sum_{j=0}^{I_{1,d}}, & I_1^d < D_{1,d} - 1 \ \& \ I_1^d = k - 1 - I_{1,d}, \end{cases} \quad (17)$$

$$D'_{3,p} = \begin{cases} \sum_{j=0}^k j \binom{p}{j}, & I_1^m = D_{1,p} - 1, \\ \sum_{j=1}^{k-I_1^p} j \binom{p}{j} + \sum_{j=1}^{I_1^p} j \binom{p}{j}, & I_1^p < D_{1,p} - 1 \ \& \ I_1^p \neq k - I_{1,p}, \\ \sum_{j=1}^{I_1^p} j \binom{p}{j}, & I_1^p < D_{1,p} - 1 \ \& \ I_1^p = k - I_{1,p}. \end{cases} \quad (18)$$

From Lemmas 1 and 2, we can see that, in the updated matrices, there are no overlapped cells, and, furthermore, all idle cells existing in the original matrices have been deleted.

By now, since $D_{1,d}$, $D_{1,p}$, $D_{2,d}$, $D_{2,p}$, $D'_{3,d}$ and $D'_{3,p}$ can be computed at a time in advance, based on the rules from $a)$ to $p)$, for one valid triple $i_1|i_2|i_3$, we can store or look it up within memory space with

$O(1)$ time consumed. Moreover, there is no memory space being wasted. In addition, we have already shown that, for one state, we can get its key $i_1|i_2|i_3$ with $O(1)$ time consumed. Therefore, we conclude that an $O(1)$ looking-up method for one already obtained state's record without any memory space wasted has been obtained.

3.3. Experimentation

These computational experiments are carried out on a computer in the setting of Intel(R) Core(TM) i7-7700 CPU, 3.60 GHz and 16.0 GB RAM in the computing laboratory of Hillman Library at University of Pittsburgh (Pittsburgh, PA, USA). For each instance, the position points of the depot and all customers are randomly generated in a unit square, so that the distance between any two points is not more than 2. The service time for customers are randomly generated through a uniform distribution on $[0,1]$.

As shown in Table 3, values of $d + k$ are set to be no more than 10, values of λ_d and λ_p are set to be both respectively 0.02 and 0.01, the value of k is set from 1 to 4, the value p are set to be, respectively, 30 and 40 and the total time consumed is no more than 10 min. In the column of "Memory (MB)", the memory consumed for storing costs and actions of states in DP implementation is shown.

Table 3. Time and memory consumed for computing exact strategies by Dynamic Programming.

d	p	k	Memory (MB)	Time Consumed (Seconds)			
				$\lambda_p = 0.02$ $\lambda_d = 0.02$	$\lambda_p = 0.02$ $\lambda_d = 0.01$	$\lambda_p = 0.01$ $\lambda_d = 0.02$	$\lambda_p = 0.01$ $\lambda_d = 0.01$
8	30	1	0.4707	0.3734	0.2002	0.1968	0.3892
9	30	1	1.0352	1.0406	0.6127	0.5688	1.0704
8	40	1	0.6172	0.5140	0.2625	0.2547	0.5095
9	40	1	1.3574	1.3593	0.8297	0.7703	1.4048
7	30	2	3.9215	2.2858	2.3251	2.2437	2.2984
8	30	2	8.5723	9.8186	6.7532	6.7219	6.8282
7	40	2	6.8328	4.3111	4.0812	3.9592	4.0202
8	40	2	14.9297	11.7779	11.7549	11.7485	11.7641
6	30	3	21.3038	17.4626	17.7750	18.3299	18.7280
7	30	3	46.2872	52.2798	53.5157	61.6470	54.8407
6	40	3	49.4947	41.7407	41.6329	48.4828	41.8578
7	40	3	107.459	126.8250	125.8310	137.7250	149.2990
5	30	4	83.9003	108.6730	119.3270	108.1410	109.2810
6	30	4	181.3340	299.6640	340.6050	321.9810	334.8110
5	40	4	261.4100	350.4090	350.8170	338.7410	354.9410

4. Heuristics

Since solving the formulated MDP problem for an exact dynamic strategy requires exponential time complexity, for larger instances, it is clear that to find the strategy through Dynamic Programming is not applicable. Thus, in this section, we will propose two heuristics to find the approximated dynamic strategy for larger instances. We will evaluate these heuristics by comparing their computational results with the results obtained through using strategy from Dynamic Programming on some computable instances.

Heuristic 1: Before the vehicle's starting off, find the optimal TSP sequence of all D-customers, a sequence which will result in the minimum total travel distance if no cancellation or no request for unscheduled service occurs. After visiting a customer, if there are some scheduled services being cancelled or new accepted P-customers' calls, re-compute the optimal TSP sequence of all current active customers.

For each combination of d , p , k , λ_p and λ_d , we randomly generate 10 instances. For each instance, we randomly simulate 2000 realizations of all customers' call-occurring situations. For each realization, we both apply the heuristic method and the obtained exact strategy and obtain two total travel

distances respectively from Heuristic 1, denoted by Dis^{Heu1} , and from the MDP strategy, denoted by Dis^{MDP} . Then, for each instance, we respectively compute its two average total travel distances over 2000 realizations, denoted by $\overline{Dis^{Heu1}}$ and $\overline{Dis^{MDP}}$. We take $\frac{(\overline{Dis^{Heu1}} - \overline{Dis^{MDP}})}{\overline{Dis^{MDP}}}$ as the performance of Heuristic 1 on this instance. Finally, for each combination of d , p , k , λ_m and λ_p , we respectively compute the average value and the worst (maximal) value of $\frac{(\overline{Dis^{Heu1}} - \overline{Dis^{MDP}})}{\overline{Dis^{MDP}}}$ over its 10 instances to evaluate the heuristic method's performance on this combination. The concrete results on some various combinations are shown in Table 4.

From Table 4, we can see that, over all these combinations, the worst average performance of Heuristic 1 is 0.0783. Moreover, there is not an apparent tendency that the average performance will deteriorate as the problem's complexity increases. However, for the worst performance, there is almost a tendency that the output will deteriorate as the problem's complexity increases.

Table 4. Performance of Heuristic 1.

d	p	k	Setting 1		Setting 2		Setting 3		Setting 4	
			$\lambda_p = 0.02$	$\lambda_d = 0.02$	$\lambda_p = 0.02$	$\lambda_d = 0.01$	$\lambda_p = 0.01$	$\lambda_d = 0.02$	$\lambda_p = 0.01$	$\lambda_d = 0.01$
			Average	Worst	Average	Worst	Average	Worst	Average	Worst
8	30	1	0.0125	0.0528	0.0178	0.0378	0.0097	0.0454	0.0097	0.0444
9	30	1	0.0050	0.0361	0.0087	0.0492	0.0048	0.0364	0.0049	0.0341
8	40	1	0.0098	0.0414	0.0186	0.0584	0.0116	0.0410	0.0126	0.0424
9	40	1	0.0063	0.0189	0.0095	0.0426	0.0076	0.0242	0.0080	0.0268
7	30	2	0.0387	0.0803	0.0283	0.0779	0.0293	0.0694	0.0295	0.0701
8	30	2	0.0368	0.1232	0.0254	0.0734	0.0319	0.0974	0.0319	0.0978
7	40	2	0.0315	0.0605	0.0465	0.1579	0.0260	0.0626	0.0263	0.0610
8	40	2	0.0453	0.1354	0.0150	0.0363	0.0461	0.1449	0.0458	0.1437
6	30	3	0.0431	0.1306	0.0671	0.1481	0.0305	0.0887	0.0315	0.0896
7	30	3	0.0663	0.2226	0.0404	0.1112	0.0468	0.1694	0.0478	0.1686
6	40	3	0.0205	0.0783	0.0385	0.1626	0.0326	0.1024	0.0157	0.0587
7	40	3	0.0239	0.0539	0.0680	0.2346	0.0394	0.1604	0.0266	0.1021
5	30	4	0.0780	0.1651	0.0783	0.1636	0.0536	0.1105	0.0537	0.1101
6	30	4	0.0444	0.2315	0.0617	0.1710	0.0432	0.1858	0.0441	0.1930
5	40	4	0.0580	0.2980	0.0574	0.3007	0.0408	0.0998	0.0412	0.1030

Heuristic 2: Before the vehicle's starting off, compute the optimal TSP sequence of all D-customers and keep this sequence unchanged all the time. After visiting a customer, if some active D-customers' calls for service cancellations are received or some P-customers' calls for services are accepted, first directly delete these D-customers from the remaining sequence and then optimally insert the new accepted P-customers into the currently remaining TSP sequence by their appearing order.

The same as Heuristic 1, we can get its performance on some various combinations as shown in Table 5. From Table 5, we can see that the worst average performance of Heuristic 2 is 0.0845. The same as Heuristic 1, there is not an apparent tendency that the average performance deteriorates with the problem's complexity but the worst performance does.

By making a comparison on these two heuristics, we can conclude that Heuristic 1 performs a little bit of better than Heuristic 2 both on average and on worst performance. However, since in Heuristic 1 an optimal TSP sequence probably needs to be computed many times, this will result in much more time consumed than Heuristic 2, especially when the number of customers is large. Thus, for instances with a relatively moderate number of customers, Heuristic 1 is recommended, while, for instances with a very large number of customers, Heuristic 2 is recommended.

Table 5. Performance of Heuristic 2.

<i>d</i>	<i>p</i>	<i>k</i>	Setting 1		Setting 2		Setting 3		Setting 4	
			$\lambda_p = 0.02$	$\lambda_d = 0.02$	$\lambda_p = 0.02$	$\lambda_d = 0.01$	$\lambda_p = 0.01$	$\lambda_d = 0.02$	$\lambda_p = 0.01$	$\lambda_d = 0.01$
			Average	Worst	Average	Worst	Average	Worst	Average	Worst
8	30	1	0.0189	0.0664	0.0244	0.0657	0.0143	0.0557	0.0141	0.0544
9	30	1	0.0081	0.0385	0.0110	0.0525	0.0069	0.0374	0.0070	0.0351
8	40	1	0.0137	0.0436	0.0254	0.0636	0.0143	0.0424	0.0151	0.0441
9	40	1	0.0106	0.0220	0.0140	0.0520	0.0110	0.0277	0.0110	0.0302
7	30	2	0.0449	0.0875	0.0356	0.0826	0.0322	0.0698	0.0325	0.0706
8	30	2	0.0406	0.1282	0.0297	0.0780	0.0337	0.0998	0.0337	0.0998
7	40	2	0.0417	0.0926	0.0516	0.1618	0.0322	0.0685	0.0321	0.0660
8	40	2	0.0486	0.1429	0.0220	0.0417	0.0482	0.1469	0.0477	0.1462
6	30	3	0.0509	0.1331	0.0732	0.1556	0.0350	0.0899	0.0359	0.0908
7	30	3	0.0743	0.2288	0.0479	0.1206	0.0511	0.1720	0.0521	0.1719
6	40	3	0.0259	0.0806	0.0459	0.1697	0.0358	0.1037	0.0187	0.0597
7	40	3	0.0308	0.0646	0.0775	0.2412	0.0447	0.1629	0.0297	0.1036
5	30	4	0.0842	0.1670	0.0845	0.1655	0.0566	0.1111	0.0567	0.1107
6	30	4	0.0512	0.2379	0.0678	0.1896	0.0467	0.1962	0.0476	0.2037
5	40	4	0.0633	0.3030	0.0628	0.3064	0.0468	0.1008	0.0471	0.1039

5. Conclusions

5.1. Conclusions and Further Discussion

This is the first time to exactly focus on an exact dynamic strategy for home pick-up services under consideration of capacitated vehicle, stochastic cancellations of pre-scheduled services, and stochastic requests for unscheduled services within one tour. Aimed at minimizing the vehicle's total expected travel distance, the problem is formulated as a multi-dimensional MDP, where the defined state consists of four components in terms of two numbers and two sets. In order to facilitate operations on states, we have successfully designed a key only consisting of three numbers for each state. When solving via Dynamic Programming, in order to avoid complexity from continually increasing, based on our designed key, we propose an $O(1)$ time consumed looking-up method for one historic state's record. Although generally this will result in a huge waste of memory, by exploiting the structural properties of the state space, we obtain an $O(1)$ looking-up method without any waste of memory. Finally, for larger instances which are challenging for DP, two well-performed heuristic methods are proposed.

To the best of our knowledge, this is the first time to study a vehicle routing problem exactly with the aforementioned two types of uncertainty on customer requests. As we are not aware of any similar study in the existing literature, we do not provide performance comparisons of our proposed model and the exact DP solution method with others. Nevertheless, numerical studies on our exact DP and two heuristic algorithms show that DP is able to derive exact solutions with a clearly better performance, and heuristics are more scalable to large-scale instances.

In addition, although a dynamic strategy within one tour is studied, the approach by which the strategy is obtained is not specific-tour dependent and hence can be sustainably used tour by tour. Thus, in this sense, the strategy can also be seen as a sustainable strategy. In addition, since the exploited structural properties result in an $O(1)$ looking-up method without any waste of memory, this strategy on moderate instances can be efficiently computed by more general computational devices such as personal mobile phone or laptop, which makes this strategy more convenient when used in real life.

5.2. Future Research

Although in this paper the objective of minimizing vehicle's total travel distance is considered, in fact, home pick-up service providers are confronted with multiple, often conflicting, objectives. Thus, future research can consider customers' preferences or a weighted objective containing both

provider's interest and customers' preferences. Furthermore, in the future, more practical considerations can be given to vehicles and customers. For example, a customer may request that his service should be completed in a time window. Some customers may have parcels of which the capacity is uncertain until the vehicle arrives at the customer's location. In addition, the travel of the vehicle is interrupted by some unforeseen situations so that the travel time is uncertain. Clearly, those situations are more involved, which demand more advanced modeling tools and computational strategies to provide effective decision support. Hence, some new structural properties and a new storage method need to be exploited, which is also a future research direction.

Author Contributions: Conceptualization, Y.W. and B.Z.; methodology, Y.W.; validation, Y.W.; formal analysis, Y.W.; investigation, Y.W. and B.Z.; writing—original draft preparation, Y.W.; writing—review and editing, Y.W. and B.Z.; supervision, B.Z. and S.H.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDP	Markov Decision Process
DP	Dynamic Programming
VRP	Vehicle Routing Problem
DVRPSC	Dynamic Vehicle Routing Problem with Stochastic Customers
TSP	Traveling Salesman Problem

Appendix A

Table A1. Storage result in a three-dimensional matrix (Part 1).

i_3	$i_1 = 0$				$i_1 = 1$			
	$i_2 = 0$	$i_2 = 1$	$i_2 = 2$	$i_2 = 3$	$i_2 = 0$	$i_2 = 1$	$i_2 = 2$	$i_2 = 3$
0	(0,0,1,0)		(0,{2},0,0)		(1,0,1,0)		(1,{2},1,0)	
1	(0,0,2,0)	(0,{1},2,0)			(1,0,2,0)	(1,{1},2,0)		
2					(1,0,1,{3})		(1,{2},1,{3})	
3					(1,0,1,{4})		(1,{2},1,{4})	
4					(1,0,1,{5})		(1,{2},1,{5})	
5					(1,0,2,{3})	(1,{1},2,{3})		
6					(1,0,2,{4})	(1,{1},2,{4})		
7					(1,0,2,{5})	(1,{1},2,{5})		
8								
9								
10								
11								
12								
13								
14					(1,0,3,0)	(1,{1},3,0)	(1,{2},3,0)	(1,{1,2},3,0)
15					(1,0,4,0)	(1,{1},4,0)	(1,{2},4,0)	(1,{1,2},4,0)
16					(1,0,5,0)	(1,{1},5,0)	(1,{2},5,0)	(1,{1,2},5,0)
17								
18								
19								
20								
21								
22								

Table A2. Storage result in a three-dimensional matrix (Part 2).

i_3	$i_1 = 2$			
	$i_2 = 0$	$i_2 = 1$	$i_2 = 2$	$i_2 = 3$
0	(2,∅,1,∅)		(2,{2},1,∅)	
1	(2,∅,2,∅)	(2,{1},2,∅)		
2	(2,∅,1,{3})		(2,{2},1,{3})	
3	(2,∅,1,{4})		(2,{2},1,{4})	
4	(2,∅,1,{5})		(2,{2},1,{5})	
5	(2,∅,2,{3})	(2,{1},2,{3})		
6	(2,∅,2,{4})	(2,{1},2,{4})		
7	(2,∅,2,{5})	(2,{1},2,{5})		
8	(2,∅,1,{3,4})		(2,{2},1,{3,4})	
9	(2,∅,1,{3,5})		(2,{2},1,{3,5})	
10	(2,∅,1,{4,5})		(2,{2},1,{4,5})	
11	(2,∅,2,{3,4})	(2,{1},2,{3,4})		
12	(2,∅,2,{3,5})	(2,{1},2,{3,5})		
13	(2,∅,2,{4,5})	(2,{1},2,{4,5})		
14	(2,∅,3,∅)	(2,{1},3,∅)	(2,{2},3,∅)	(2,{1,2},3,∅)
15	(2,∅,4,∅)	(2,{1},4,∅)	(2,{2},4,∅)	(2,{1,2},4,∅)
16	(2,∅,5,∅)	(2,{1},5,∅)	(2,{2},5,∅)	(2,{1,2},5,∅)
17	(2,∅,3,{4})	(2,{1},3,{4})	(2,{2},3,{4})	(2,{1,2},3,{4})
18	(2,∅,3,{5})	(2,{1},3,{5})	(2,{2},3,{5})	(2,{1,2},3,{5})
19	(2,∅,4,{3})	(2,{1},4,{3})	(2,{2},4,{3})	(2,{1,2},4,{3})
20	(2,∅,4,{5})	(2,{1},4,{5})	(2,{2},4,{5})	(2,{1,2},4,{5})
21	(2,∅,5,{3})	(2,{1},5,{3})	(2,{2},5,{3})	(2,{1,2},5,{3})
22	(2,∅,5,{4})	(2,{1},5,{4})	(2,{2},5,{4})	(2,{1,2},5,{4})

Table A3. Storage result of all valid D-triples.

I_3^p	$I_1^p = 0$		$I_1^p = 1$	
	$I_2^p = 0$	$I_2^p = 1$	$I_2^p = 0$	$I_2^p = 1$
0	(0,∅,1,∅)	(0,{2},1,∅)	(2,∅,1,∅)	(2,{2},1,∅)
1	(0,∅,2,∅)	(0,{1},2,∅)	(2,∅,2,∅)	(2,{1},2,∅)
2			(2,∅,1,{3})	(2,{2},1,{3})
3			(2,∅,1,{4})	(2,{2},1,{4})
4			(2,∅,1,{5})	(2,{2},1,{5})
5			(2,∅,2,{3})	(2,{1},2,{3})
6	(1,{1},2,{5})	(1,∅,2,{5})	(2,∅,2,{4})	(2,{1},2,{4})
7	(1,{1},2,{4})	(1,∅,2,{4})	(2,∅,2,{5})	(2,{1},2,{5})
8	(1,{1},2,{3})	(1,∅,2,{3})	(2,∅,1,{3,4})	(2,{2},1,{3,4})
9	(1,{2},1,{5})	(1,∅,1,{5})	(2,∅,1,{3,5})	(2,{2},1,{3,5})
10	(1,{2},1,{4})	(1,∅,1,{4})	(2,∅,1,{4,5})	(2,{2},1,{4,5})
11	(1,{2},1,{3})	(1,∅,1,{3})	(2,∅,2,{3,4})	(2,{1},2,{3,4})
12	(0,{1},2,∅)	(1,∅,2,∅)	(2,∅,2,{3,5})	(2,{1},2,{3,5})
13	(1,{2},1,∅)	(1,∅,1,∅)	(2,∅,2,{4,5})	(2,{1},2,{4,5})

Table A4. Storage result of all valid P-triples.

I_3^m	$I_1^m = 0$				$I_1^m = 1$			
	$I_2^m = 0$	$I_2^m = 1$	$I_2^m = 2$	$I_2^m = 3$	$I_2^m = 0$	$I_2^m = 1$	$I_2^m = 2$	$I_2^m = 3$
0	(2,∅,5,{4})	(2,{1},5,{4})	(2,{2},5,{4})	(2,{1,2},5,{4})	(1,∅,3,∅)	(1,{1},3,∅)	(1,{2},3,∅)	(1,{1,2},3,∅)
1	(2,∅,5,{3})	(2,{1},5,{3})	(2,{2},5,{3})	(2,{1,2},5,{3})	(1,∅,4,∅)	(1,{1},4,∅)	(1,{2},4,∅)	(1,{1,2},4,∅)
2	(2,∅,4,{5})	(2,{1},4,{5})	(2,{2},4,{5})	(2,{1,2},4,{5})	(1,∅,5,∅)	(1,{1},5,∅)	(1,{2},5,∅)	(1,{1,2},5,∅)
3	(2,∅,4,{3})	(2,{1},4,{3})	(2,{2},4,{3})	(2,{1,2},4,{3})				
4	(2,∅,3,{5})	(2,{1},3,{5})	(2,{2},3,{5})	(2,{1,2},3,{5})				
5	(2,∅,3,{4})	(2,{1},3,{4})	(2,{2},3,{4})	(2,{1,2},3,{4})				
6	(2,∅,5,∅)	(2,{1},5,∅)	(2,{2},5,∅)	(2,{1,2},5,∅)				
7	(2,∅,4,∅)	(2,{1},4,∅)	(2,{2},4,∅)	(2,{1,2},4,∅)				
8	(2,∅,3,∅)	(2,{1},3,∅)	(2,{2},3,∅)	(2,{1,2},3,∅)				

References

- Express Process Query for Chinese Express Firms. Available online: <http://www.kuaidi100.com/all/> (accessed on 28 March 2019).
- Ulmer, M.W.; Brinkmann, J.; Mattfeld, D.C. Anticipatory planning for courier, express and parcel services. In *Logistics Management*; Springer: Berlin, Germany, 2015; pp. 313–324.
- UPS Help and Support Center. Available online: <https://www.ups.com/us/en/help-center/sri/change-pickup-or-collection.page> (accessed on 28 March 2019).
- Ritzinger, U.; Puchinger, J.; Hartl, R.F. A survey on dynamic and stochastic vehicle routing problems. *Int. J. Prod. Res.* **2016**, *54*, 215–231. [[CrossRef](#)]
- Hoffmann, B.; Chalmers, K.; Urquhart, N.; Guckert, M. Athos-A Model Driven Approach to Describe and Solve Optimisation Problems: An Application to the Vehicle Routing Problem with Time Windows. In Proceedings of the 4th ACM International Workshop on Real World Domain Specific Languages, Washington, DC, USA, 17 February 2019; p. 3.
- Stavropoulou, F.; Repoussis, P.; Tarantilis, C. The Vehicle Routing Problem with Profits and consistency constraints. *Eur. J. Oper. Res.* **2019**, *274*, 340–356. [[CrossRef](#)]
- Breunig, U.; Baldacci, R.; Hartl, R.F.; Vidal, T. The electric two-echelon vehicle routing problem. *Comput. Oper. Res.* **2019**, *103*, 198–210. [[CrossRef](#)]
- Zhang, S.; Zhang, W.; Gajpal, Y.; Appadoo, S. Ant Colony Algorithm for Routing Alternate Fuel Vehicles in Multi-depot Vehicle Routing Problem. In *Decision Science in Action*; Springer: Berlin, Germany, 2019; pp. 251–260.
- Nikolopoulou, A.I.; Repoussis, P.P.; Tarantilis, C.D.; Zachariadis, E.E. Adaptive memory programming for the many-to-many vehicle routing problem with cross-docking. *Oper. Res.* **2019**, *19*, 1–38. [[CrossRef](#)]
- Macrina, G.; Laporte, G.; Guerriero, F.; Pugliese, L.D.P. An energy-efficient green-vehicle routing problem with mixed vehicle fleet, partial battery recharging and time windows. *Eur. J. Oper. Res.* **2019**, *276*, 971–982. [[CrossRef](#)]
- Karagul, K.; Sahin, Y.; Aydemir, E.; Oral, A. A Simulated Annealing Algorithm Based Solution Method for a Green Vehicle Routing Problem with Fuel Consumption. In *Lean and Green Supply Chain Management*; Springer: Berlin, Germany, 2019; pp. 161–187.
- Salavati-Khoshghalb, M.; Gendreau, M.; Jabali, O.; Rei, W. An exact algorithm to solve the vehicle routing problem with stochastic demands under an optimal restocking policy. *Eur. J. Oper. Res.* **2019**, *273*, 175–189. [[CrossRef](#)]
- Crainic, T.G.; Mancini, S.; Tadei, R.; Perboli, G. Reactive GRASP with Path Relinking for the Two-Echelon Vehicle Routing Problem. *Adv. Metaheuristics* **2013**, *101*, 113–125.
- Macrina, G.; Pugliese, L.D.P.; Guerriero, F.; Laporte, G. The green mixed fleet vehicle routing problem with partial battery recharging and time windows. *Comput. Oper. Res.* **2019**, *101*, 183–199. [[CrossRef](#)]
- Tadei, R. Two-Echelon Vehicle Routing Problem: Asatellite Location Analysis. *Procedia-Soc. Behav. Sci.* **2010**, *2*, 5944–5955.
- Fink, M.; Desaulniers, G.; Frey, M.; Kiermaier, F.; Kolisch, R.; Soumis, F. Column generation for vehicle routing problems with multiple synchronization constraints. *Eur. J. Oper. Res.* **2019**, *272*, 699–711. [[CrossRef](#)]
- Froger, A.; Mendoza, J.E.; Jabali, O.; Laporte, G. Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. *Comput. Oper. Res.* **2019**, *104*, 256–294. [[CrossRef](#)]
- Rodríguez-Martín, I.; Salazar-González, J.J.; Yaman, H. The periodic vehicle routing problem with driver consistency. *Eur. J. Oper. Res.* **2019**, *273*, 575–584. [[CrossRef](#)]
- Dascioglu, B.G.; Tuzkaya, G. A Literature Review for Hybrid Vehicle Routing Problem. In *Industrial Engineering in the Big Data Era*; Springer: Berlin, Germany, 2019; pp. 249–257.
- Gayialis, S.P.; Konstantakopoulos, G.D.; Tatsiopoulos, I.P. Vehicle Routing Problem for Urban Freight Transportation: A Review of the Recent Literature. In *Operational Research in the Digital Era—ICT Challenges*; Springer: Berlin, Germany, 2019; pp. 89–104.
- Schiffer, M.; Schneider, M.; Walther, G.; Laporte, G. Vehicle Routing and Location Routing with Intermediate Stops: A Review. *Transp. Sci.* **2019**. [[CrossRef](#)]

22. Larsen, A.; Madsen, O.B. The Dynamic Vehicle Routing Problem. 2000. Available online: <http://orbit.dtu.dk/files/5261816/imm143.pdf> (accessed on 4 April 2019)
23. Pillac, V.; Gendreau, M.; Guret, C.; Medaglia, A.L. A review of dynamic vehicle routing problems. *Eur. J. Oper. Res.* **2013**, *225*, 1–11. [[CrossRef](#)]
24. Powell, W.B.; Simao, H.P.; Bouzaiane-Ayari, B. Approximate dynamic programming in transportation and logistics: a unified framework. *EURO J. Transp. Log.* **2012**, *1*, 237–284. [[CrossRef](#)]
25. Potvin, J.Y.; Xu, Y.; Benyahia, I. Vehicle routing and scheduling with dynamic travel times. *Comput. Oper. Res.* **2006**, *33*, 1129–1137. [[CrossRef](#)]
26. Ichoua, S.; Gendreau, M.; Potvin, J.Y. Exploiting knowledge about future demands for real-time vehicle dispatching. *Transp. Sci.* **2006**, *40*, 211–225. [[CrossRef](#)]
27. Van Hentenryck, P.; Bent, R.; Upfal, E. Online stochastic optimization under time constraints. *Ann. Oper. Res.* **2010**, *177*, 151–183. [[CrossRef](#)]
28. Hentenryck, P.V.; Bent, R. *Online Stochastic Combinatorial Optimization*; The MIT Press: Cambridge, MA, USA, 2009.
29. Bent, R.W.; Van Hentenryck, P. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Oper. Res.* **2004**, *52*, 977–987. [[CrossRef](#)]
30. Hvattum, L.M.; Lkktangen, A.; Laporte, G. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transp. Sci.* **2006**, *40*, 421–438. [[CrossRef](#)]
31. Mercier, L.; Van Hentenryck, P. An anytime multistep anticipatory algorithm for online stochastic combinatorial optimization. *Ann. Oper. Res.* **2011**, *184*, 233–271. [[CrossRef](#)]
32. Schilde, M.; Doerner, K.F.; Hartl, R.F. Metaheuristics for the dynamic stochastic dial-a-ride problem with expected return transports. *Comput. Oper. Res.* **2011**, *38*, 1719–1730. [[CrossRef](#)] [[PubMed](#)]
33. Pavone, M.; Frazzoli, E.; Bullo, F. Adaptive and distributed algorithms for vehicle routing in a stochastic and dynamic environment. *IEEE Trans. Autom. Control* **2011**, *56*, 1259–1274. [[CrossRef](#)]
34. Zhu, L.; Rousseau, L.M.; Rei, W.; Li, B. Paired cooperative reoptimization strategy for the vehicle routing problem with stochastic demands. *Comput. Oper. Res.* **2014**, *50*, 1–13. [[CrossRef](#)]
35. Nguyen, V.H.; Vuong, Q.H.; Tran, M.N. *Central Limit Theorem for Functional of Jump Markov Processes*; Springer: Berlin, Germany, 2005.
36. Howard, R.A. *Dynamic Programming and Markov Processes*; John Wiley: Oxford, UK, 1960.
37. Fianu, S.; Davis, L.B. A Markov decision process model for equitable distribution of supplies under uncertainty. *Eur. J. Oper. Res.* **2018**, *264*, 1101–1115. [[CrossRef](#)]
38. Papadimitriou, C.H.; Tsitsiklis, J.N. The complexity of Markov decision processes. *Math. Oper. Res.* **1987**, *12*, 441–450. [[CrossRef](#)]
39. Parcel Capacity Requirements of Chinese Express Firms. Available online: <http://www.chinawutong.com/baike/81061.html> (accessed on 28 March 2019).
40. Ulmer, M.W.; Goodson, J.C.; Mattfeld, D.C.; Thomas, B.W. Dynamic Vehicle Routing: Literature Review and Modeling Framework. 2017. Available online: https://www.researchgate.net/publication/313421699_Dynamic_Vehicle_Routing_Literature_Review_and_Modeling_Framework (accessed on 4 April 2019)
41. Kreher, D.L.; Stinson, D.R. *Combinatorial Algorithms: Generation, Enumeration, and Search*; CRC Press: Boca Raton, FL, USA, 1998; Volume 7.

 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).