The multi-period competitive location problem of temporary retail facilities

Warley Almeida Silva

1 Description

The retail industry has been experiencing the surge of a brand-new mode of location decisions - temporary retail facilities, also popularly known as pop-up stores (Rosenbaum et al., 2021). On one hand, property owners are prone to negotiating shorter leases due to the high vacancy rates as many brick-and-mortar stores go online. On the other hand, retail companies are realizing the benefits of exploring markets for a restricted amount of time, profiting from the sense of urgency for being temporary and strategically engaging unfamiliar customers with their brand. Brick-and-mortar and online retail companies alike have been profiting from the installation of temporary retail facilities. However, there is no framework in the literature tailored to the strategic location of temporary retail facilities, helping decision makers exploit the market for restricted amounts of time and optimizing the profit over a planning horizon.

Two characteristics of the retail industry add complexity to the problem. First, retail managers do not have direct control over the estimated revenue of an installed facility unlike other industries. Retail companies cannot control how different customer profiles will decide to patronize their stores, but they can often model the patronizing behavior of the customers in a certain reasonable way, which is often a discrete choice model. Second, the company making decisions is often not the only company in the market offering a (potentially brand-new) product or a service. In other words, the company should be aware of competitors acting and reacting in this market. However, knowing for sure how competitors are going to react is quite difficult, and there is some intrinsic uncertainty involved in the process. These additional difficulties are often considered in the so-called competitive facility location problems.

This report proposes a dynamic programming model for the multi-period competitive location problem of temporary retail facilities. More specifically, let us consider an online retail company that would like to exploit the market of a metropolitan region. The retail company would like to have only one temporary retail facility opened per time period (e.g., months, quarters, or years), which must be moved to a different location at the end of each time period. Managers have selected a set of candidate locations fit for the structure of temporary facilities and with owners prone to short-term leasing, and each location has a maintenance cost (e.g., rental and staff costs). The desired location at the next

time period must be chosen at the beginning of the current time period, before knowing what competitors will do in the next time period, to allow the company to proceed with rental procedures and organize everything accordingly. These candidate locations are also desirable for the competitors, but the company does not know in advance where competitors will install temporary facilities. In this sense, the company must take into account rational reactions of the competitors, which may not be fully rational throughout the planning horizon, and assume that if it decides to install a temporary facility in the same location as a competitor they will not be successful (i.e., a pessimistic rental turnout).

The retail company has built a rank-based discrete choice model to emulate the patronizing behavior of the customers. More specifically, managers have established a set of customer profiles, where each customer profile has a personal ranking over the candidate locations. This personal ranking may depend implicitly on multiple aspects of the locations (e.g., ease of parking, neighbourhood style, driving distance, or additional amenities). Each customer profile patronizes the temporary facility at the location with the highest rank, which has been opened either by the company or by the competitors, and brings in an estimated revenue. In this context, the retail company would like to know which location to plan moving the temporary facility to at the beginning of each time period to maximize its expected profit over the planning horizon.

2 Model

Stages, states, and actions. Each time period represents a stage k, where stages $\bar{k} = 0$ and $\bar{k} = N$ are the first and last time periods. Let $\mathcal{L}$ be the set of candidate locations. Each candidate location of the temporary facility at stage k represents a state x_k. Due to the competition, managers may fail to secure the desired next location for the temporary facility. Let $\bar{x}_k = 0$ be the state to represent the situation where the company did not succeed in installing a temporary facility due to the competitors at stage k. Let also $\{\bar{x} = 0\}$ be equal to the empty set $\emptyset$ for the sake of simplicity. The set of feasible states at stage k is $X_k = X = \mathcal{L} \cup \{0\}$. Each desired location of the temporary facility in time period $k+1$ represents an action u_k. Recall that the company must move the temporary facility to a different location at the end of the planning horizon, so the current location of the temporary facility is not considered as a feasible action $\bar{u}_k$. The set of feasible actions at stage k for state x_k is $U_k(x_k) = U(x_k) = \mathcal{L} \backslash \{x_k\}$.

Uncertainty. The uncertainty at stage k is the rearrangement of competitor facilities at time period $k + 1$ (i.e., how competitors are going to react to the temporary facility at location x_k in stage k). Let ω_k be the random variable that describes the competitor configuration in time period $k+1$. Here, configuration means simply a subset of candidate locations. This random variable is drawn from the support $\Omega_k(x_k) = \Omega(x_k) = \{\omega \,|\, \omega \subseteq \mathcal{L} \backslash \{x_k\}\}$ (i.e., set of candidate locations perceived to be available in time period $k + 1$ by competitors). Let us assume that the random variables ω_k are independent between time periods, which is a reasonable assumption in the dynamic context of temporary retail.

The probability distribution of the random variable ω_k can be estimated based on how lucrative the competitor configuration ω_k would be for the competitors. Let $r(A, B)$, with $A \subseteq \mathcal{L}$ and $B \subseteq \mathcal{L}$, be the revenue function defined later, which gives the revenue obtained by configuration A when competing against configuration B. The probability of the random variable ω_k being equal to ω at state x_k in stage k can be estimated with the help of the softmax function as $P_k(\omega_k = \omega \,|\, x_k) = \frac{e^{\beta_k(r(\omega,\{x_k\})-m(\omega))}}{\sum_{\bar{\omega}\in\Omega_k(x_k)} e^{\beta_k(r(\bar{\omega},\{x_k\})-m(\bar{\omega}))}}$. Constant β_k is key for modelling a decaying rational reaction of the competitors throughout the planning horizon. Intuitively, competitors may be fierce at the beginning of the competition, but may not be willing to spend time and resources for sequentially optimal reactions as time progresses. This behavior can be modeled by having $\beta_{k+1} < \beta_k \, \forall k \in \{0, ..., N-2\}$, which causes competitors to have progressively more equal probabilities over the support $\Omega_k(x_k)$ as time progresses. Let us set $\beta_k = \frac{1}{d^k}$, where $d > 1$ is the rationality decay parameter. Large values of d describe scenarios where competitors quickly become less rational.

Transition function. If the company decides to move the temporary facility from the current location x_k to a desired location u_k, the actual location x_{k+1} in time period $k+1$ depends on what competitors have chosen as their configuration ω_k in time period $k+1$. Since the company assumes the worse turnout of rental negotiations, the state x_{k+1} is going to be the desired location u_k only if location u_k is not part of the competitor configuration ω_k. Therefore, the transition function can be written as $f(x_k, u_k, \omega_k) = \begin{cases} u_k, & \text{if } u_k \notin \omega_k \\ 0, & \text{otherwise} \end{cases}$.

Cost function. First, the company must pay the maintenance cost $m(x_k)$ for having a temporary facility at location x_k. Assume that no maintenance is paid for the temporary facility in stage 0. Second, the company must collect the revenue $r(\{f(x_k, u_k, \omega_k)\}, \omega_k)$ that depends on the desired location u_k and the competitor configuration ω_k. Therefore, the cost functions can be written as $g_k(x_k, u_k, \omega_k) = m(x_k) - r(\{f(x_k, u_k, \omega_k)\}, \omega_k)$, and $g_N(x_k, u_k, \omega_k) = m(x_k)$.

Revenue function. The revenue function $r(A, B)$, with $A \subseteq \mathcal{L}$ and $B \subseteq \mathcal{L}$, relies on the rank-based discrete choice model that describes the patronizing behavior of the customers (e.g., Beresnev and Melnikov (2019) use this choice model). Let $\mathcal{C}$ be the set of customer profiles, where each customer profile j has a ranking over the candidate locations $\mathcal{L}$, a function $h_j(C)$, with $C \subseteq \mathcal{L}$, that returns the location in subset C with the highest rank for customer j, and an estimated revenue p_{ij} for a temporary facility at location i. Therefore, the revenue function $r(A, B)$ can be written as $r(A, B) = \sum_{i\in A} \sum_{j\in\mathcal{C} \,|\, i=h_j(A\cup B)} p_{ij}$.

Bellman recurrence equations. Recall that $J_k(x)$ is the optimal expected cost at stage k from state x, so $-J_k(x)$ is the optimal expected profit. In this sense, the Bellman recurrence equations can be written as follows:

$$J_N(x) = m(x) \qquad \forall x \in X_N \tag{1}$$

$$J_k(x) = \min_{u\in U_k(x_k)} Q_k(x, u) \qquad \forall x \in X_k, \forall k \in \{0, ...N-1\} \tag{2}$$

$$\text{where } Q_k(x, u) = m(x) + \mathbb{E}_{\omega_k}\left[-r(\{f(x, u, \omega_k)\}, \omega_k) + J_{k+1}(f(x, u, \omega_k))\right] \tag{3}$$

3 Algorithms

The main challenge in the implementation of exact and approximate algorithms to solve this problem is handling the support $\Omega_k(x_k)$, which grows exponentially with the number of candidate locations $|\mathcal{L}|$. This challenge can be partially overcome by considering a randomly sampled set of realizations $\bar{\Omega}_k(x_k) \subset \Omega_k(x_k)$ of size s. The size s can be chosen according to the trade-off between the quality of the optimal policy and computational time. Explicitly, the expectation term has been computed as $\mathbb{E}_{\omega_k}[F(\omega_k)] = \sum_{\omega \in \bar{\Omega}_k(x_k)} P_k(\omega_k = \omega) \cdot F(\omega_k)$.

Exact approach. The stochastic backward chaining algorithm has been implemented to obtain exact policies in the computational experiments, as there is no known closed form of the optimal policy for this problem. The stochastic backward chaining algorithm also gives somewhat approximate policies if the support of the random variable ω_k has been sampled instead of enumerated. Algorithm 1 presents a pseudocode of the implementation. In summary, the implemented algorithm applies the version of the Bellman recurrence equations based on the $Q_k(x, u)$ values to compute the optimal cost-to-go values $J_k(x)$ and the optimal policy $\pi^* = (\mu_0^*, ..., \mu_{N-1}^*)$. Let us refer to this algorithm with the full support $\Omega_k(x_k)$ as the backward solver, and to the version relying on a sample of size s of the support $\Omega_k(x_k)$ as the sampled backward solver.

Algorithm 1 Stochastic backward chaining algorithm

Require: instance I
function RUN SOLVER
  for $k \in \{N, ..., 0\}$ **do**
    for $x \in X = \mathcal{L} \cup \{0\}$ **do**
      if k = N **then**
        $J_N(x) \leftarrow g_N(x)$ ▷ Equation (1)
      else
        $\bar{U} \leftarrow U(x)$
        $u^* \leftarrow first(\bar{U})$ ▷ First action
        for $u \in \bar{U}$ **do**
          if $Q_k(x, u) < Q_k(x, u^*)$ **then** ▷ Equation (3)
            $u^* \leftarrow u$
          end if
        end for
        $J_k(x) \leftarrow Q_k(x, u^*)$ ▷ Equation (2)
        $\mu_k^*(x) \leftarrow u^*$ ▷ Equation (2)
      end if
    end for
  end for
  return $J_k(x), \mu_k^*(x)$
end function

Approximate approach. The fitted value iteration algorithm with a linear architecture has been developed to obtain approximate policies in the computational experiments. Since the main challenge is to compute the expectation term, let us approximate the $Q_k(x, u)$ values instead of the $J_k(x)$ values. Let us work with a linear architecture for this approximation $\tilde{Q}_k(x, u) = r_k^T \phi(x, u)$, where r_k^T is a coefficient vector learned from one-step lookahead observations

and $\phi(x, u)$ is a feature vector for state x and action u. Once the $\tilde{Q}_k(x, u)$ functions have been learned, the approximate cost-to-go values $\tilde{J}_k(x)$ and the approximate policy $\tilde{\pi} = (\tilde{\mu}_0, ..., \tilde{\mu}_{N-1})$ can be computed using the same loop shown in algorithm 1, but switching the true $Q_k(x, u)$ values by the $\tilde{Q}_k(x, u)$ values. Let us refer to this algorithm as the parametric solver, and to the version relying on a sample of the support $\Omega_k(x_k)$ as the sampled parametric solver.

There are two ingredients for the parametric solver. First, it is important to define how to train coefficients $r_{N-1}, r_{N-2}, ..., r_0$. Let us employ the training method seen on class, which uses the approximation $\tilde{Q}_{k+1}(x, u)$ from stage $k + 1$ (or true values for stage N) to compute the label $\hat{Q}_k(x, u)$ for a point $\phi(x, u)$. In other words, function $\hat{Q}_k(x, u)$ can be written as $\hat{Q}_k(x, u) = m(x) + \mathbb{E}_{\omega_k}\left[-r(\{f(x, u, \omega_k)\}, \omega_k) + \min_{\bar{u} \in U_{k+1}(f(x,u,\omega_k))} \tilde{Q}_{k+1}(f(x, u, \omega_k), \bar{u})\right]$. If the stage $k+1$ is the last stage N, the approximate function $\tilde{Q}_N(x, u)$ returns $g_N(x)\, \forall u \in U_N(x)$. The set of points and labels generated by the combinations of states $x \in X_k$ and $u \in U_k(x)$ for stage k, which can be enumerated, have been used for the training coefficient r_k. Algorithm 2 presents a pseudocode of the offline training procedure, which uses the mean of squared errors from the `scipy` package (Virtanen et al., 2020) as the loss function.

Algorithm 2 Training procedure for the parametric solver

Require: instance I
function TRAIN SOLVER
  for $k \in \{N - 1, ..., 0\}$ **do**
    $m \leftarrow 0$
    for $x \in X = \mathcal{L} \cup \{0\}$ **do**
      for $u \in U(x)$ **do**
        $points[m] \leftarrow \phi(x, u)$
        $labels[m] \leftarrow \hat{Q}_k(x, u)$
        $m \leftarrow m + 1$
      end for
    end for
    $r_k \leftarrow minimizeLeastSquaredError(points, labels)$
  end for
  return $\tilde{Q}_k(x, u) \leftarrow r_k^T \phi(x, u)$
end function

Second, it is important to define the feature vector $\phi(x, u)$. Three sets of features have been used: (1) the distance between locations x and u in the ranking of each customer $j \in \mathcal{C}$; (2) the maintenance costs $m(x)$ and $m(u)$; and (3) the estimated revenues $r(\{x\}, \{\ell\})$ and $r(\{u\}, \{\ell\})$ $\forall \ell \in \mathcal{L}$. The feature set (1) give some intuition of how taking action u at state x increases or decreases the attraction of customer profile j. The feature set (2) provides information about the maintenance costs of state x and action u, and the feature set (3) emulates the competition of location x and location u with a competitor configuration of size one to provide some insight about the behavior of the revenue function.

4 Experiments

Instances. There are three instances: a small instance I^S, a medium instance I^M, and a large instance I^L. Instance I^S has three candidate locations, two customer profiles, and a planning horizon of two time periods. Instance I^M has four candidate locations, eight customer profiles, and a planning horizon of six time periods. Instance I^G has ten candidate locations, thirteen customer profiles, and a planning horizon of six time periods.

Experiments. Let us answer three questions through the experiments. First, are the exact and approximate approaches correctly implemented? Second, how does the rationality decay parameter d affect the expected profit? Third, how well can the parametric solver approximate the backward solver?

Simulation. In the computational experiments, it is important to compare the optimal policy π^* (obtained by the backward solver for instances I^S and I^M, and by the sampled backward solver with $s = 512$ for instance I^L) with a potentially sub-optimal policy $\tilde{\pi}$ (obtained by the parametric solver for instances I^S and I^M, and by the sampled parametric solver with $s = 512$ for instance I^L). In this sense, when the text below mentions the $J_k(x)$ values found by applying an sub-optimal policy $\tilde{\pi} = (\tilde{\mu}_0, ..., \tilde{\mu}_{N-1})$, they refer to the $J_k(x)$ values calculated using Algorithm 1 where lines $8-13$ concerned with finding the action that minimizes the $Q_k(x, u)$ values are changed to enforce $u^* \leftarrow \tilde{\mu}_k(x)$. The comparison of the $J_k(x)$ values induced by the optimal policy π^* and the ones induced by the sub-optimal policy $\tilde{\pi}$ provides a strategy to compare these two policies, and to judge whether the sub-optimal policy $\tilde{\pi}$ is of good quality.

First question. Let us use instance I^S to answer this question. The rationality decay parameter d has been set to 1 to simplify calculations, and the supports $\Omega_k(x_k)$ are fully enumerated. Table 1 presents the $J_k(x)$ values computed manually in the first row, the $J_k(x)$ values outputted by the backward solver along with the optimal policy π^* in the second row, the $J_k(x)$ values obtained when applying the policy $\tilde{\pi}$ outputted by the parametric solver in the third row, and the $\tilde{J}_k(x)$ values estimated by the parametric solver when computing the policy $\tilde{\pi}$. Manual calculations have been omitted due to space.

(k, x)	$(0,0)$	$(0,1)$	$(0,2)$	$(0,3)$	$(1,0)$	$(1,1)$	$(1,2)$	$(1,3)$
$J_k(x)$ manual	0	-450	0	-450	0	-800	100	-800
$J_k(x)$ from π^*	0	-450	0	-450	0	-800	100	-800
$J_k(x)$ from $\tilde{\pi}$	50	-450	0	-450	100	-800	100	-800
$\tilde{J}_k(x)$ from $\tilde{\pi}$	-200	-400	-300	-400	-300	-600	100	-600

Tab. 1: Values of $J_k(x)$ and $\tilde{J}_k(x)$ obtained manually and by policies π^* and $\tilde{\pi}$.

First, the $J_k(x)$ values and the optimal policy π^* outputted by the backward solver match those calculated manually. Therefore, the implementation of the backward solver is correct. Second, the $J_k(x)$ values computed when applying the potentially sub-optimal policy $\tilde{\pi}$ outputted by the parametric solver gives, in

fact, a sub-optimal policy. Nevertheless, this sub-optimal policy $\tilde{\pi}$ does not give $J_k(x)$ values that far from the optimal policy π^*, which is the desirable outcome of using a parametric solver. Third, the approximate $\tilde{J}_k(x)$ values are not very close to the true $J_k(x)$ values, even though there are some parallels between these values (e.g., for states 0 and 2 in stage 0 it holds that $J_0(0) = J_0(2)$ and $\tilde{J}_0(0) = \tilde{J}_0(2)$). The arguably bad approximation $\tilde{J}_k(x)$ is the reason behind the sub-optimality of policy $\tilde{\pi}$ for instance I^S. Nevertheless, the results for the backward and parametric solvers show that the implementations are correct (i.e., they match manually computed values and have expected behaviors).

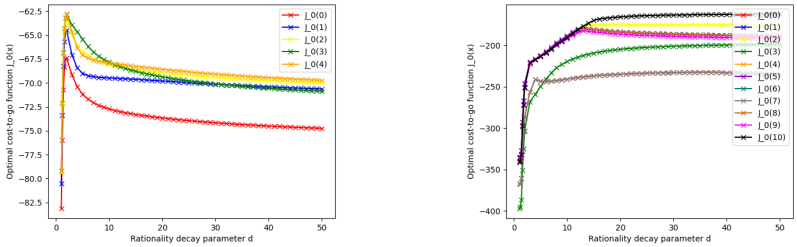
Second question. Let us use instances I^M and I^L to answer this question. Figure 1 present the $J_0(x)$ values for instance I^M (found by the backward solver) and for instance I^L (found by the sampled backward solver with $s = 512$).

The rationality decay parameter $d > 1$ models the decaying effort of competitors when competing for multiple time periods, which allow the company to be strategic over time. One could think intuitively that larger values of d would be better for the expected profit of the company, as it means that competitors will become less rational faster in the planning horizon. However, that is not necessarily the case due to the non-linear and disconnected nature of the revenue function $r(A, B)$, which depends on the discrete choice model.

For the sake of intuition, consider the following analysis. Fully rational competitors may place a higher probability on a configuration $\bar{\omega_k}$ that captures a customer profile $\bar{j}$ through the second (or lower) highest ranked location $\bar{i}$ because the estimated revenue $p_{\bar{i}\bar{j}}$ is the most attractive. In this context, the company has the opportunity to capture this customer profile (among others) by moving the temporary facility to the highest ranked location $\bar{k}$ for customer profile j, which guarantees the capture of this customer profile and some estimated revenue. However, with a decaying rationality, the competitor may have a higher probability of selecting a configuration that has location $\bar{k}$, which prevent the company from capturing this customer profile and accessing this estimated revenue. Therefore, it makes sense that the rationality decay parameter d does not necessarily increase or decrease the expected profit. For the rest of the experiments, let us consider the rationality parameter $d = 2$, which can be interpreted as the competitors becoming halfly as rational per time period.

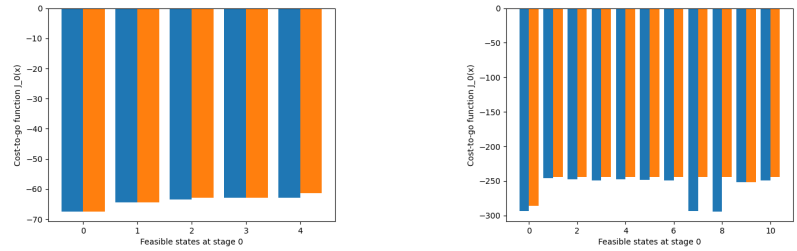
Third question. Let us use instances I^M and I^L to answer question #3. Figure 2 presents the $J_0(x)$ values for the feasible states x at stage 0 when applying the optimal policy π^* in blue and when applying approximate policy $\tilde{\pi}$ in orange for instances I^M and I^L. Recall that the optimal policy π^* and the approximate policy $\tilde{\pi}$ for instance I^L have been computed by the sampled backward solver and the sampled parametric solver, respectively, with $s = 512$.

In terms of computational time, the backward solver takes on average 0.21 seconds to find the optimal policy π^* for instance I^M, and the sampled backward solver with $s = 512$ takes 443.23 seconds on average to find the optimal policy π^* for instance I^L. The parametric solver spends on average 0.32 seconds with the offline training, and takes 0.05 seconds to find the approximate policy $\tilde{\pi}$ for instance I^M. The sampled parametric solver with $s = 512$ spends on average 527.62 seconds with the offline training, and takes 0.12 seconds to find the



(a) Instance I^M. (b) Instance I^L.

Fig. 1: $J_0(x)$ values with varying values of the rationality decay parameter d.



(a) Instance I^M. (b) Instance I^L.

Fig. 2: $J_0(x)$ values when applying optimal policy π^* in blue and when applying approximate policy $\tilde{\pi}$ in orange.

approximate policy $\tilde{\pi}$ for instance I^M. In this sense, the (sampled) parametric solver is considerably faster to find an online policy, specially for instance I^L.

In terms of policy quality, for instance I^M, the approximate policy $\tilde{\pi}$ induces $J_0(x)$ values quite close to the ones obtained by the optimal policy π^*. In fact, three out of five feasible states at stage 0 have $J_0(x)$ values equal to the optimal ones. For instance I^L, the same can be partially observed, even though there are larger differences between $J_0(x)$ values obtained by the approximate policy $\tilde{\pi}$ for states 7 and 8 at stage 0. Overall, the parametric solver with a linear architecture seems to perform better than expected in providing an approximate policy $\tilde{\pi}$ close in quality to the optimal policy π^*. However, as expected due to the non-linear and discontinuous nature of the revenue function $r(A, B)$, the estimated $\tilde{J}_0(x)$ values outputted by the (sampled) parametric solver are not close to the real ones. Since the $J_0(x)$ values of the optimal policy π^* and the approximate policy $\tilde{\pi}$ are sufficiently close for most of the feasible states at stage 0, the parametric solver could be used as the main solver in practice. However, there is some risk involved in this choice, as if the initial state x_0 was 7 and 8 for instance I^L, for example, the approximate policy $\tilde{\pi}$ would guide the company towards a considerably sub-optimal path throughout the planning horizon.

5 Conclusion

The multi-period competitive location problem of temporary retail facilities arises from recent developments in the retail industry. The problem encodes many interesting challenges, such as estimating how customers behave, optimizing location decisions in a competitive environment, modelling the rationality decay of the competitors throughout the planning horizon, enforcing constant change to keep the urgency and novelty factor of the temporary facilities, and being strategic under uncertainty to maximize the profit of the company.

This project has proposed a stochastic dynamic programming formulation for a constrained version of the problem, where the retail company can only have one temporary facility per time period. The stochastic backward chaining algorithm, referred to as the backward solver, has been used as the exact approach, and the fitted value iteration algorithm with a linear architecture, referred to as the parametric solver, has been developed as an approximate approach. The results show that being the backward solver can only solve quickly instances of smaller sizes due to the computationally costly expectation term, but the parametric solver yields sub-optimal policies of considerably high quality faster after a somewhat costly offline training procedure for practical applications. The results also show that the proposed rationality decay parameter d may not be easy to set as the increase of this parameter does not necessarily mean that the company will have larger expected profits at the end of the planning horizon.

Some of the concepts seen on class would not be applicable to the problem under consideration. For example, it would not be reasonable to approximate the $J_k(x)$ value in the fitted value iteration algorithm, as it would require the parametric solver to still compute the expectation term to find the approximate policy $\tilde{\pi}$. In addition, due to the non-linear and discontinuous nature of the revenue function $r(A, B)$, approximating the $Q_k(x, u)$ values seems more suitable because it allows the parametric solver to connect state x with action u through the feature vector $\phi(x, u)$, which would not be possible otherwise.

It would also not be reasonable to consider an infinite planning horizon, as the concept of temporary retail is to exploit a market for a restricted amount of time. If one were to consider an infinite planning horizon and still set the parameter $\beta_k = \frac{1}{d^k}$, the probability $P_k(\omega_k = \omega \mid x_k)$ computed using the softmax function would give progressively less and less weight to the profitability of the competitor configuration ω_k and converge towards an uniform distribution among the realizations of the random variable ω_k. In this sense, one could potentially find an analytical policy based on this property for an infinite horizon.

Nevertheless, some of the concepts seen on class would be applicable to the problem under consideration. For example, one could propose a rollout variant to find an approximate policy $\tilde{\pi}$ that consisted of selecting the location with the smallest maintenance cost until the end of the planning horizon. In addition, instead of using a linear architecture for the parametric solver, one could apply machine learning techniques and use, for example, a neural network to approximate the $Q_k(x, u)$ values inside the fitted value iteration framework.

Future works on this problem could study, for example, other ways of mod-

elling the rationality decay of the competitors throughout the planning horizon. It is also possible to add more complexity to the costs functions (e.g., consider transportation costs from location x_k to location x_{k+1}) and more constraints to the decisions (e.g., assume that the company has a budget for each stage k and not all locations can be funded). Lastly, it is also possible to study the problem without the restriction of having only one temporary facility per time period. If the company can have a subset of locations with temporary facilities at each time period, the set of feasible states increase exponentially according to the number of candidate locations, which would add complexity to the problem.

6 Instructions

First, let us start by describing the folders of the project. Folder `instances` has folders `small`, `medium`, and `large` for instances I^S, I^M, and I^L, respectively. Each instance has been described in four files. File `metadata.txt` contains sets $\mathcal{L}$ and $\mathcal{C}$ and the number of time periods N of the planning horizon; file `locations.csv` contains the maintenance cost $m(i)\,\forall i \in \mathcal{L}$; file `customers.csv` contains the rankings of customer profile $\forall j \in \mathcal{C}$; and file `revenues.csv` contains the estimated revenue $p_{ij}\,\forall i \in \mathcal{L}, \forall j \in \mathcal{C}$. Folder `graphs` contains graphs generated by the code, folder `policies` contains policies exported by the code, and folder `training` contains training data used by the parametric solver.

Second, let us explain the scripts. Script `instance.py` contains the definition of the Instance class. This class has as attributes the instance information and as methods the maintenance cost $m(x)$ and the revenue function $r(A, B)$, among others. Script `backward.py` contains the definition of the Backward class, which contains the implementation of the backward solver. This class has as attributes the built policy π and the values $J_k(x)$ and as methods the $Q_k(x, u)$ function and the $J_k(x)$ function, among others. Script `parametric.py` contains the definition of the Parametric class, which contains the implementation of the parametric solver. This class has as attributes the built policy π and the trained coefficients r_k and as methods the $\tilde{Q}_k(x, u)$ function and the training procedure, among others. Script `main.py` creates an object of the Instance class based on the instance folder provided as an argument and creates objects of the Backward and Parametric solvers to run those solvers according to the arguments. Script `generator.py` creates a random instance according to some parameters (e.g., number of locations and number of customers profiles) and export it to the folder `instances/random`. Lastly, scripts `bar_graph.py` and `decay_graph.py` draw the graphs seen in Section 4 based on the provided arguments and the policies it can find in the `policies` folder.

Third, let us provide some instructions on how to run the code. Start by checking if the required packages listed on `requirements.txt` are installed in your computer. Command `python main.py instances/<name> --backward --parametric -d <d> -s <s>` runs the backward and parametric solvers for instance `<name>` with the rationality decay factor $d =$ `<d>` and the number of samples $s =$ `<s>`. Adding the `--verbose` flag makes the code output the com-

putations inside the solver (e.g., $Q_k(x, u)$ values), and adding the `--policies` flag makes the code output the policy found by the solvers in textual format. Command `python decay_graph.py instances/<name> <max-decay> -s 512` draws the decay graph discussed in the second question of Section 4 for instance `<name>` with $s = 512$ and the rationality decay parameter d varying from 1 to `max-decay`. Command `python bar_graph.py instances/<name> -d 2 -s 512` draws the bar graph discussed in the third question of Section 4 for instance `<name>` with $s = 512$ and $d = 2$. Most of the scripts have been implemented with the `argparse` package, so it is possible to type `python <name>.py --help` to obtain more information about the script. Lastly, there is a list of commands in `experiments.sh` ran for generating the results seen in Section 4.

7 Self-assessment

3.1 Problem description: A+. The problem studied in the project has been well described from a practical point of view and has interesting properties from a theoretical point of view. The problem is original, and has been briefly contextualized inside the literature about the competitive facility location problem and temporary retail. This section complies with the instructions.

3.2 Modelling: A+. The model presented in the project is well described and has all the ingredients from a stochastic dynamic programming formulation. The assumptions about the structure of the problem (e.g., random variables) are clearly stated and defended. This section complies with the instructions.

3.3 Algorithms: A+. The algorithms implemented in the project have been well explained with the help of pseudocodes and mathematical equations. There is also a strong parallel between the explanation of the algorithms and the way they have been implemented. This section complies with the instructions.

3.4 Results and analysis: A-. The analysis of the results obtained for this project has shown that the algorithms have been correctly implemented, discussed one of the main characteristics of the problem, and defended the parametric solver with linear architecture as viable options to the standard backward solver. However, more analysis about the results of the parametric solver could have been done (e.g., how do the results vary for the parametric solver if the sample size changes?). It was difficult to provide intuition about the instances within space constraints. This section complies with the instructions.

3.5 Conclusion: A+. The conclusion written in the project have correctly summarized the motivation of the problem and the findings through computational experiments. In addition, the conclusion has properly contextualized how other concepts seen on this class could be applied to the problem, as well as proposed future expansions. This section complies with the instructions.

3.6 Other parts (instructions, references, self assessment): A+. The instructions on how to run the code have been clearly written with examples, the references have been properly cited throughout the report, and the self-assessment has been properly at the end of the report.

3.7 Code: A+. The code submitted for this project has been well

documented, with a straightforward connection to the mathematical notation used in the report. The code has been separated in multiple modules for the ease of comprehension, and each script has specific instructions on how to run it. The code complies with instructions.

3.8 Presentation: A-. The presentation made for this project has tried to convey as much information as possible about the model and the results, but some details have been explained without much depth or examples due to time constraints. The presentation complies with instructions.

References

Beresnev, V. and Melnikov, A. (2019). Approximation of the competitive facility location problem with mips. *Computers & Operations Research*, 104:139–148.

Rosenbaum, M. S., Edwards, K., and Ramirez, G. C. (2021). The benefits and pitfalls of contemporary pop-up shops. *Business Horizons*, 64(1):93–106.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.