Exact Algorithms for Continuous Pricing with Advanced Discrete Choice Demand Models

Tom Haering [*] Robin Legault [†] Fabian Torres [*]
Ivana Ljubić [‡] Michel Bierlaire [*]

December 11, 2023

Report TRANSP-OR 231211
Transport and Mobility Laboratory
School of Architecture, Civil and Environmental Engineering
Ecole Polytechnique Fédérale de Lausanne
`transp-or.epfl.ch`

[*]École Polytechnique Fédérale de Lausanne (EPFL), School of Architecture, Civil and Environmental Engineering (ENAC), Transport and Mobility Laboratory, Switzerland, {tom.haering, fabian.torres, michel.bierlaire }@epfl.ch

[†]Massachusetts Institute of Technology (MIT), Operations Research Center, USA, legault@mit.edu

[‡]ESSEC Business School, Department of Information Systems, Decision Sciences and Statistics, France, ljubic@essec.edu

Abstract

We present the Breakpoint Exact Algorithm (BEA) together with a Spatial Branch and Bound (B&B), and Spatial Branch and Benders Decomposition (B&BD) approach to tackle the uncapacitated choice-based pricing problem (CPP) where demand is captured by a discrete choice model (DCM). Integrating advanced DCMs into optimization problems is challenging, due to the lack of closed-form probability expressions. We leverage problem characteristics to reformulate the state-of-the-art simulation-based formulation of the CPP as a mixed integer linear program (MILP) into a non-convex quadratically constrained quadratic program (QCQP), and then into a non-convex QCQP with linear objective (QCQP-L). We exploit utility breakpoints to develop the BEA, which scales polynomially in the number of customers and draws, along with an efficient Spatial Branch and Bound algorithm utilizing the McCormick envelope for relaxations, which are then solved using Benders decomposition. Our methods are evaluated against solving the MILP, QCQP, or QCQP-L with GUROBI, on a mixed logit parking space operator case study. We outspeed the state of the art by several orders of magnitude when optimizing one or two prices and reduce computational time drastically for larger numbers of prices. This methodology suits all choice-based optimization problems with linear-in-price utilities, given any random utility model.

Keywords: discrete choice models, optimal pricing, exact algorithms, Spatial Branch and Bound, Benders Decomposition

1 Introduction

Price optimization is essential when pricing decisions need to be made for one or multiple products, particularly when there are cross-effects between their demands (Talluri & Van Ryzin, 2004). This problem can arise in various areas, including revenue management for airlines, railways, and hotels, assortment pricing in retail, or product line pricing in consumer goods industries.
Disaggregate demand representations, such as discrete choice models, allow for a better reflection of the heterogeneity of customers' tastes and preferences and account for supply-demand interactions (Sumida et al., 2021). However, including such models increases computational complexity, as the resulting choice probabilities are nonlinear. As such, customary nonlinear algorithms may terminate at a local optimum due to the highly nonlinear expected revenue function. Hanson & Martin (1996) were among the first to demonstrate that the projected revenue function does not exhibit concavity in prices, even in the case of a basic logit model. Later works have demonstrated that, under uniform price sensitivities across all products, the expected revenue function is concave in the choice probability vector

(D. Zhang & Lu, 2013; Song et al., 2021). This concavity result also holds under asymmetric price sensitivities, not only for the logit, but also for the nested logit (NL) model (Li & Huh, 2011). Unique price solutions exist for some logit models under restrictive conditions on the degree of asymmetry in the price sensitivity parameters, as shown for the logit (Akçay et al., 2010), NL (Gallego & Wang, 2014), paired combinatorial logit (PCL) (Li & Webster, 2017), and generalized extreme value (GEV) models (H. Zhang et al., 2018).

In some studies, pricing decisions are optimized jointly with other decisions, such as assortment or scheduling decisions (Bertsimas et al., 2020), and decisions of multiple firms are studied mainly from a game theory perspective, both cooperative (Bortolomiol et al., 2021) and non-cooperative (Schlicher & Lurkin, 2022).

The logit model exhibits many limitations, see Train (2009): it does not account for random taste variation, i.e. differences in tastes that cannot be linked to observed characteristics. It implies proportional substitution across alternatives, which can lead to unrealistic behavior, like the Red Bus / Blue Bus paradox. Furthermore, logit cannot handle situations where unobserved factors are correlated over time.

To address these issues, more involved choice models have been explored in recent literature: Increasingly many authors consider pricing problems under a mixed logit (ML) model that accommodates customer heterogeneity by allowing some parameters to vary across customers (Li et al., 2019; van de Geer & den Boer, 2022). The ML model can, under mild regularity conditions, approximate choice probabilities of any discrete choice model derived from the random utility maximization (RUM) assumption (McFadden & Train, 2000), making it a popular choice model. However, the expected revenue function under the mixed logit model is not well-behaved, and the concavity property with respect to the choice probabilities breaks down, even for entirely symmetric price sensitivities across products and segments.

Gilbert et al. (2014) proposed a tractable approximation of the ML pricing problem for a revenue-maximizing network pricing problem, which involves selecting appropriate tolls for a congested network. They solve it in a two-step approach: First, they solve a mixed integer program that simplifies the original problem by assuming a simpler distribution for the price sensitivity parameter. The optimal solution of this program is then utilized as the starting solution for an ascent algorithm that solves a differentiable optimization problem, which approximates the original ML pricing problem more accurately.

Li et al. (2019) study a price optimization problem with discrete ML demand. They propose two concave maximization problems as lower and upper bounds for the revenue function and develop an algorithm that converges to a local optimum.

Marandi & Lurkin (2020) propose an iterative optimization algorithm that asymptotically converges to the optimal solution, by formulating a linear optimization

problem based on the trust-region approach to find a feasible solution, and designing a convex optimization problem using a convexification technique to approximate the optimization problem from above. They then use a branching method to tighten the optimality gap.
van de Geer & den Boer (2022) consider a finite mixture of logit in a market segmented by products and intrinsic product valuations of customers. However, the customers' price sensitivity parameters are product-dependent only, allowing for a scalable algorithm that quickly converges to an optimal product price.
A general implementation approach for integrating any advanced choice model into an optimization problem has been proposed in Paneque et al. (2021), where Monte Carlo simulation is used to generate a deterministic problem at the cost of an increase in complexity since the resulting mixed integer linear problem (MILP) involves finding the best price over a large number of scenarios, generated by taking draws from the stochastic components of the formulation. With a sufficiently large number of draws, the MILP formulation guarantees convergence to globally optimal solutions. However, since the complexity of the MILP scales exponentially with the number of draws, the approach can currently only be applied to solving small-scale instances, i.e., with few individuals, draws, and alternatives.

We summarize that for pricing with disaggregate demand modeling in the form of a DCM, which we will refer to as the choice-based pricing problem (CPP), there is no exact solution approach in the literature that is both general and capable of solving realistic instances in a reasonable amount of time. Our goal is to fill this gap by leveraging on the MILP approach in Paneque et al. (2021).
To this extent, we first present some key properties of the problem that allow us to reformulate it more efficiently, leading from the MILP to a non-convex quadratically constrained quadratic program (QCQP), and non-convex quadratically constrained linear program (QCLP) formulation. Exploiting additional features allows us to develop the breakpoint exact algorithm (BEA), whose time complexity scales polynomially in the number of customers and draws, but exponentially in the number of prices to be optimized. We then introduce a Spatial Branch and Bound (B&B) algorithm that makes use of the McCormick envelope together with custom branching and enumeration rules to efficiently solve the problem for complex instances. This B&B algorithm is further accelerated by the use of a Benders decomposition to solve the McCormick envelope in each node, an approach that we refer to as Spatial Branch and Benders Decomposition. Finally, we compare the MILP, the QCQP, and QCLP formulations (all solved using the state-of-the-art solver GUROBI) to our custom algorithms by application to a mixed logit parking choice case study by Ibeas et al. (2014).
The paper is structured as follows. Section 2 describes the choice-based pricing problem in its probabilistic form. In Section 3, we explore its key characteristics

and the different mathematical formulations. Section 4 describes the breakpoint exact algorithm, followed by the Spatial Branch and Bound procedure in Section 5 and the extension with a Benders decomposition in Section 6. Section 7 presents the computational experiments. Section 8 concludes the paper and presents the essential takeaways of this study.

2 Problem definition

Consider a competitive market with $J + K$ products, of which J products are controlled by a supplier that wants to identify the set of prices that maximizes their revenue. We number the controlled alternatives from 1 to J, and the competitors' alternatives using non-positive numbers, from $1 - K$ to 0. The supply for every product is assumed to be uncapacitated. We then consider N customers choosing exactly one product among all offered alternatives (as the K competing products do not generate any revenue for the supplier, choosing not to buy any product can be seen as part of the competition). Each individual may furthermore have a different set of considered alternatives, as some products might inherently be rejected or otherwise unavailable to them. We thus denote C_n the choice set of individual n, where $n \in \mathcal{N} = \{1, \ldots, N\}$. The behavior of the customers is captured by a random utility model: each alternative i is associated with a stochastic utility U_{in}, which depends on socioeconomic characteristics of individual n, as well as alternative-specific attributes, and can be defined as follows:

$$
\begin{aligned}
U_{jn} &= V_{jn} + \varepsilon_{jn} && \forall j \in \{1-K, \ldots, 0\} \cap C_n,\\
U_{in} &= V_{in} + \beta_p^{in} p_i + \varepsilon_{in} && \forall i \in \{1, \ldots, J\} \cap C_n,
\end{aligned}
$$

where V_{in} is the deterministic part of the utility that is observed by the analyst, which can take any form and be non-linear in the explanatory variables (attributes of the customer or the product), and ε_{in} is the error term that is unobserved (and thus a random variable). The only assumption we make on the utilities is for them to be linear in the prices p_i, which are multiplied by a pricing coefficient $\beta_p^{in} < 0$, that can vary across n and i. This variation can be explicit, using a function of the socio-economic characteristics, or implicit, using a random variable with an assumed distribution. In all cases, the coefficient is assumed to be negative. The probability $P_n(i)$ that individual n chooses alternative $i \in C_n$ can now be written as follows:

$$P_n(i) = \mathbb{P}(U_{in} \geq U_{jn} \; \forall j \in C_n)$$

We can now formulate the probabilistic version of the uncapacitated choice-based pricing problem (CPP), see Formulation 1.

$$
\begin{aligned}
&\max_{p,U} \sum_{n\in\mathcal{N}} \sum_{i\in\{1,\ldots,J\}\cap C_n} P_n(i)p_i \\
&\text{s.t.} \\
&P_n(i) = \mathbb{P}(U_{in} \geq U_{jn}\ \forall j \in C_n) && \forall n \in \mathcal{N}, i \in C_n \\
&U_{jn} = V_{jn} + \varepsilon_{jn} && \forall n \in \mathcal{N}, j \in \{1-K,\ldots,0\} \cap C_n \\
&U_{in} = V_{in} + \beta_p^{in} p_i + \varepsilon_{in} && \forall n \in \mathcal{N}, i \in \{1,\ldots,J\} \cap C_n \\
&p \in [p_1^L, p_1^U] \times \ldots \times [p_J^L, p_J^U] \\
&U \in \mathbb{R}^{(J+K)N}
\end{aligned} \tag{1}
$$

The controlled prices $p_i, i \in \{1,\ldots,J\}$ are decision variables that need to be optimized in order to maximize the expected profit, expressed as each product's price times the probability the product is bought by an individual, summed up over all individuals. We assume each price p_i to be bounded within a continuous domain $[p_i^L, p_i^U]$. It is crucial that customers are able to buy non-controlled products, as otherwise, the problem could become unbounded.
For some well-known choice models, like the logit, nested logit, or ordered logit, the probability $P_n(i)$ has a closed-form expression, allowing for the probabilistic CPP to be solved using conventional optimization methods. However, as mentioned in the introduction, such models have various drawbacks. Aiming to alleviate the unrealistic assumptions inherent in the logit model, advanced discrete choice models like mixed logit or probit have demonstrated improved predictive power, at the cost of the choice probability $P_n(i)$ in general no longer taking on a closed-form expression and thus being difficult to integrate in an optimization problem.

3 Problem formulations and properties

To address the lack of closed-form expressions for the probability functions in the stochastic version of the CPP, we employ the simulation approach of Paneque et al. (2021): We take R draws ε_{inr} from the distribution of the error terms to generate R scenarios with deterministic utilities U_{inr}:

$$
\begin{aligned}
U_{jnr} &= V_{jn} + \varepsilon_{jnr} && \forall j \in \{1-K,\ldots,0\}, n \in \mathcal{N}, r \in \mathcal{R}, \\
U_{inr} &= V_{in} + \beta_p^{in} p_i + \varepsilon_{inr} && \forall i \in \{1,\ldots,J\}, n \in \mathcal{N}, r \in \mathcal{R},
\end{aligned}
$$

where $\mathcal{R} = \{1, \ldots, R\}$. As the V_{in} are constant as well, we can define $c_{inr} := V_{in} + \varepsilon_{inr}$ and further simplify the utilities as:

$$\begin{aligned} U_{jnr} &= c_{jnr} && \forall j \in \{1-K, \ldots, 0\}, n \in \mathcal{N}, r \in \mathcal{R}, \\ U_{inr} &= c_{inr} + \beta_p^{in} p_i && \forall i \in \{1, \ldots, J\}, n \in \mathcal{N}, r \in \mathcal{R}. \end{aligned}$$

In every scenario, each customer wants to maximize their utility across all alternatives available to them. This can be written as a binary knapsack problem, see Formulation 2, where p is a given vector of prices and ω_{inr} are called choice variables. We have that:

$$\omega_{inr} = \begin{cases} 1 & \text{if } U_{inr}(p) = \max\limits_{j \in C_n} U_{jnr}(p), \\ 0 & \text{otherwise,} \end{cases} \qquad \forall n \in \mathcal{N}, i \in C_n, r \in \mathcal{R}.$$

It is well established that, if all utilities $U_{inr}(p)$ are distinct, the knapsack problem admits a constraint matrix that is totally unimodular, see for example Wolsey (2020). This implies that the restrictions on ω_{inr} being binary can be relaxed to $\omega_{inr} \geq 0$ without altering the optimal solution, allowing us to consider the dual (Formulation 3) of this continuous relaxation, and invoke strong duality.

$$\begin{aligned} &\max_{\omega} \sum_{i \in C_n} \omega_{inr} U_{inr}(p) \\ &\text{s.t.} \\ &\sum_{i \in C_n} \omega_{inr} = 1 \\ &\omega_{inr} \in \{0, 1\} \quad \forall i \in C_n \end{aligned} \tag{2}$$

$$\begin{aligned} &\min_{h_{nr}} h_{nr} \\ &\text{s.t.} \\ &h_{nr} \geq U_{inr}(p) \quad \forall i \in C_n \end{aligned} \tag{3}$$

Both the primal and dual are always feasible and bounded, hence we can impose the following sufficient optimality conditions:

$$\begin{aligned} &(\text{Strong duality}) & h_{nr} &= \textstyle\sum_{i \in C_n} \omega_{inr} U_{inr}(p), \\ &(\text{Dual feasibility}) & h_{nr} &\geq U_{inr}(p) && \forall i \in C_n, \\ &(\text{Primal feasibility I}) & \textstyle\sum_{i \in C_n} \omega_{inr} &= 1, \\ &(\text{Primal feasibility II}) & \omega_{inr} &\geq 0 && \forall i \in C_n, \end{aligned}$$

which will allow us to integrate the utility maximization problem for each customer and scenario into a larger optimization framework.
Given the above characterization of the choice variables ω_{inr}, we can approximate the choice probabilities as follows:

$$P_n(i) \approx \widehat{P}_n(i) = \frac{1}{R}\sum_r \omega_{inr} \qquad \forall n \in \mathcal{N}, i \in C_n, r \in \mathcal{R}.$$

The probability of individual n choosing alternative i is thus approximated by the average number of times that customer made that choice over all scenarios. This estimator is unbiased, see for example Train (2009).
Before we present the full model, we can simplify the problem by gathering all products that are not in our control, i.e. that are offered by potential competitors, into one single opt-out alternative. This is possible because, given a scenario, all of these utilities become constants, thus we only have to keep the option that has the highest constant utility as it will always be preferred over all other non-controlled alternatives. We thus define:

$$U_{0nr} = \max_{j \in \{1-K,\ldots,0\} \cap C_n} c_{jnr} \qquad \forall n \in \mathcal{N}, r \in \mathcal{R}.$$

As this utility is again constant we refer to it as c_{0nr}. We can simplify the notation in the following sections by redefining $C_n = \{1, \ldots, J\} \cap C_n$ and assuming the opt-out to be available to everyone.
It is worth noting that the presented simulation approach can be extended to also deal with randomly distributed parameters, as would be necessary in the case of for example a mixed logit model. To this extent, we likewise take draws from the distributions of the stochastic parameters, and, given a scenario r, denote a draw of a distributed parameter β_d^{in} by β_{dr}^{in}.

3.1 MILP formulation

The full MINLP formulation of the uncapacitated choice-based pricing problem (CPP) is given in Formulation 4.

$$
\begin{aligned}
&\max_{p,\omega,U,h} \frac{1}{R}\sum_{r\in\mathcal{R}}\sum_{n\in\mathcal{N}}\sum_{i\in C_n} p_i\omega_{inr} && && (o)\\
&\text{s.t.} && &&\\
&\sum_{i\in C_n\cup\{0\}} \omega_{inr} = 1 && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\mu_{nr})\\
&h_{nr} = c_{0nr}\omega_{0nr} + \sum_{i\in C_n} U_{inr}\omega_{inr} && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\zeta_{nr})\\
&h_{nr} \geq c_{0nr} && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\alpha_{0nr})\\
&h_{nr} \geq U_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\alpha_{inr})\\
&U_{inr} = c_{inr} + \beta_p^{in} p_i && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\kappa_{inr})\\
&\omega \in \{0,1\}^{(J+1)NR} && &&\\
&p \in [p_1^L, p_1^U]\times\ldots\times[p_J^L, p_J^U] && &&\\
&U, h \in \mathbb{R}^{JNR}, \mathbb{R}^{NR} && &&
\end{aligned}
\tag{4}
$$

The objective function (o) is equal to the profit and is thus defined as the approximated choice probability of individual n selecting alternative $i \in C_n$ multiplied by the alternative's price p_i, summed over all individuals. The constraints define the individual choices: Constraints (μ_{nr}) guarantee that exactly one alternative is chosen per individual and scenario. Constraints (κ_{inr}) model the utility U_{inr} of each alternative i for individual n in scenario r. Constraints (ζ_{nr}) and constraints (α_{inr}) enforce the optimality conditions for the customer utility maximization problem. Note that this is a non-convex MINLP formulation, as both the objective (o) and the constraints (ζ_{nr}) contain the products $p_i\omega_{inr}$. These non-linear terms are handled automatically by modern solvers like GUROBI or CPLEX, but can be linearized explicitly using a big-M approach, as proposed by Paneque et al. (2021). The big-M value is determined by the highest possible value the variable p_i can achieve, which here corresponds to an upper bound p_i^U, set by the supplier. The products $p_i\omega_{inr}$ is then replaced by auxiliary variables η_{inr}, which is defined using the following constraints:

$$
\begin{aligned}
0 &\leq \eta_{inr} \leq p_i^U\omega_{inr}\\
p_i - p_i^U(1-\omega_{inr}) &\leq \eta_{inr} \leq p_i.
\end{aligned}
$$

Note that by linearizing the products, the knapsack property is lost, i.e., it is no longer possible to relax the integrality constraints on ω. This is a natural drawback of the formulation as an MILP and the reason for which we will later extend the framework of Paneque et al. (2021) to a non-convex formulation.

Finally, the price variables may also depend on the individuals (or groups of individuals), thus allowing for segmented targeting of the population. Let us now explore some key properties of the problem.

3.2 Restriction to fixed prices

If the prices are all fixed to constant values, it is trivial to find the optimal values of all variables in Formulation 4. To see this, it is enough to observe that with fixed prices p, the problem reduces to solving the utility maximization problem (Formulation 2) for each customer n and scenario r. As the prices are the only connecting variables, this can be done separately for every tuple (n, r).
This property will be used in various ways within our algorithms, for example in order to quickly generate feasible solutions from just a vector of prices, to compute the resulting profit, or in the context of a Benders decomposition approach, where in each iteration prices are fixed to a candidate solution, making the subproblem easy to solve.

3.3 Continuous reformulation

In the MILP formulation of the problem introduced by Paneque et al. (2021), the integrality property of the knapsack problem for every customer is not exploited. This is due to the fact, that in an MILP context, the product $p_i \omega_{inr}$ has to be linearized with big-M constraints, which invalidate the knapsack property. However, if we choose to not linearize the product, we can instead define the problem as a non-convex quadratically constrained quadratic program (QCQP), see Formulation 5. The formulation is the same as Formulation 4, except that the variables ω_{inr} are no longer constrained to be binary and instead are relaxed to be in the interval $[0, 1]$.

$$
\begin{aligned}
&\max_{p,\omega,U,h} \frac{1}{R} \sum_{r\in\mathcal{R}} \sum_{n\in\mathcal{N}} \sum_{i\in C_n} p_i \omega_{inr} \\
&\text{s.t.} \\
&\sum_{i\in C_n\cup\{0\}} \omega_{inr} = 1 && \forall n\in\mathcal{N}, r\in\mathcal{R} \quad (\mu_{nr}) \\
&h_{nr} = c_{0nr}\omega_{0nr} + \sum_{i\in C_n} U_{inr}\omega_{inr} && \forall n\in\mathcal{N}, r\in\mathcal{R} \quad (\zeta_{nr}) \\
&h_{nr} \geq c_{0nr} && \forall n\in\mathcal{N}, r\in\mathcal{R} \quad (\alpha_{0nr}) \\
&h_{nr} \geq U_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} \quad (\alpha_{inr}) \\
&U_{inr} = c_{inr} + \beta_p^{in} p_i && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} \quad (\kappa_{inr}) \\
&\omega \in [0,1]^{(J+1)NR} \\
&p \in [p_1^L, p_1^U] \times \ldots \times [p_J^L, p_J^U] \\
&U, h \in \mathbb{R}^{JNR}, \mathbb{R}^{NR}
\end{aligned}
\tag{5}
$$

We can further simplify the problem by isolating all non-convexity into a set of bilinear constraints (λ_{inr}) which define the product $p_i\omega_{inr}$, turning the problem into a non-convex QCQP with linear objective (QCQP-L), described in Formulation 6. As this reformulation is purely algebraic, all properties of the QCQP-L formulation still hold, most notably, under the assumption of distinct utilities, the program is again integral.

$$
\begin{aligned}
&\max_{p,\omega,\eta,U,h} \frac{1}{R}\sum_{r\in\mathcal{R}}\sum_{n\in\mathcal{N}}\sum_{i\in C_n}\eta_{inr} && && \\
&\text{s.t.} && && \\
&\sum_{i\in C_n\cup\{0\}}\omega_{inr} = 1 && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\mu_{nr}) \\
&h_{nr} = c_{0nr}\omega_{0nr} && && \\
&\qquad + \sum_{i\in C_n}[c_{inr}\omega_{inr} + \beta_p^{in}\eta_{inr}] && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\zeta_{nr}) \\
&h_{nr} \geq c_{0nr} && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\alpha_{0nr}) \\
&h_{nr} \geq U_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\alpha_{inr}) \\
&U_{inr} = c_{inr} + \beta_p^{in}p_i && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\kappa_{inr}) \\
&\eta_{inr} = p_i\omega_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\lambda_{inr}) \\
&\omega \in [0,1]^{(J+1)NR} && && \\
&p \in [p_1^L, p_1^U]\times\ldots\times[p_J^L, p_J^U] && && \\
&U, h \in \mathbb{R}^{JNR}, \mathbb{R}^{NR} && &&
\end{aligned} \tag{6}
$$

3.4 Breakpoints

A very useful insight into the CPP is that, even though the price variables live in a continuous domain, we can a priori enumerate all relevant solution values, to which we later refer to as breakpoints. The key is the following observation: for any of the controlled prices, its optimal value will necessarily be such that the utility of its corresponding alternative is equal to the utility of the next cheapest alternative (for some customer and scenario). This is due to the fact that we want to maximize the profit, thus increasing the price as much as possible without losing any more customers than necessary. For an optimal price p_i, there will thus inadvertently exist a customer n and scenario r such that if we increase p_i by any $\varepsilon > 0$, the utility U_{inr} would fall under that of cheaper alternatives or the opt-out, thus deterring that customer and lowering the total profit. Otherwise, this increase would improve the current solution, which would consequently be suboptimal. This optimal value for p_i can thus be considered a "breakpoint" in the customer's decision.
To illustrate this idea we consider the case of one price and one customer and scenario. This means the only relevant breakpoint is the price $\overline{p}_1$ for which the

utility of the product to be sold is equal to the utility of the opt-out alternative:

$$c_{1nr} + \beta_p^1 \overline{p}_1 = c_{0nr} \implies \overline{p}_1 = \frac{c_{0nr} - c_{1nr}}{\beta_p^1}.$$

Figure 1 shows the revenue of the supplier as a function of the price p_1 of the one controlled alternative. We observe that the optimal profit is achieved at $p_1 = \overline{p}_1$.

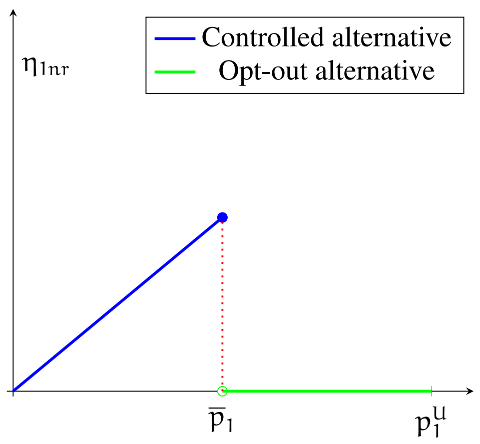

Figure 1 – Revenue for one price and one simulated customer

This is because we assume the so-called "optimistic setting", where at a utility tie, we assume customers to choose the higher-priced alternative, see for example Labbé et al. (1998). As we are maximizing a sample average estimate of the true objective function (i.e. we rely on simulation and use a finite sample), this objective function is non-concave. This can be illustrated further by adding more customers, see Figure 2, resulting in the typical shape of the objective functions of (bi)linear pricing problems, see for example Violin (2016). Here the optimal revenue is achieved at $p = \overline{p}_1^2$. Above each line, we indicate the number of captured customers for that price range.

If we knew the breakpoints in advance, it would be sufficient to evaluate the revenue for each of them and take the best solution. The question thus arises of how to identify all breakpoints. For this we can apply the following procedure: First, we need to assume an ordering of the prices, i.e. a permutation s such that $p_{s(1)} \leq \ldots \leq p_{s(J)}$. We then compute all breakpoints for $p_{s(1)}$ compared to the opt-out option. Subsequently we iterate over all of these values, fixing $p_{s(1)}$ to each breakpoint, which allows us to compute a new intermediary opt-out option for each tuple (n, r), which will have utility $\overline{U}_{0nr} = \max(U_{0nr}, U_{s(1)nr}(\overline{p}_{s(1)}))$. This allows us to compute the breakpoints for the next highest price $\overline{p}_{s(2)}$ in the

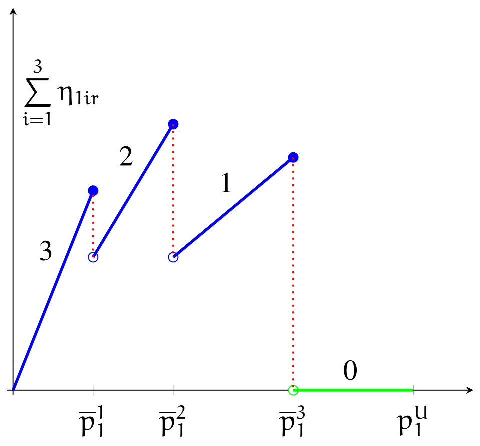

Figure 2 – Revenue for one price and three simulated customers

same way. This procedure continues all the way to the highest price until all breakpoints for this ordering have been explored. Repeating this enumerative process for all possible permutations results in the complete set of breakpoints.

In the next section, we describe how we can transform this insight into an efficient algorithm, whose complexity grows polynomially in the number of customers and scenarios.

4 Breakpoint exact algorithm (BEA)

In this section, we present the breakpoint exact algorithm (BEA) for the CPP that expands and exploits the notion of breakpoints discussed in Section 3.4. This algorithm first assumes the order of prices to be fixed. The original problem is then solved by testing all possible permutations of prices separately. The BEA can easily be adapted to a heuristic by considering only a subset of the possible orderings of prices, where the solution will depend strongly on how close the optimal ordering is to the ones explored. Throughout this section, to lighten the notation, we assume that $p_0 \leq p_1 \leq \ldots \leq p_J$, with $p_0 = 0$ being the price of the opt-out alternative.

The algorithm proceeds by iteratively introducing and fixing the price of new alternatives, from the cheapest to the most expensive one. When a new alternative is introduced in the market, we calculate, for each customer, the price for which its utility would be equal to that of the alternative that was previously selected. We then sort these breakpoints and solve the restricted problem associated with each of them, as well as the one obtained by fixing the price of the current alternative to its upper bound. This procedure is repeated recursively until all the prices have

been fixed. At the end of the algorithm, the space of breakpoints has been fully explored, and the incumbent solution is optimal.
For a new alternative j, and given the set $\{0, 1, \dots, j-1\}$ of cheaper alternatives whose prices have been previously fixed, the breakpoint $\bar{p}_j^{nr}$ is defined as the price at which the utility of alternative j matches the maximum utility over alternatives $\{0, 1, \dots, j-1\}$ for customer (n, r). This price is given by:

$$\bar{p}_j^{nr} = \frac{h_{nr}^j - c_{jnr}}{\beta_p^{jn}},$$

where $h_{nr}^j = \max_{i \in \{0,\dots,j-1\}}\{c_{inr} + \beta_p^{in} p_i\}$. From there, the set of relevant breakpoints for alternative j corresponds to all the breakpoints $\bar{p}_j^{nr}$ that lie in the feasible interval $[p_j^L, p_j^U]$. The upper bound p_j^U also has to be considered in the BEA, as increasing the price of j to its maximum feasible value can lead to an optimal solution in some cases. For example, this systematically occurs in instances where all the customers select alternative j in any feasible solution. Finally, we can omit all the prices in the interval $[p_j^L, p_{j-1}]$, as we assumed that the prices respect the order $p_0 \leq \dots \leq p_{j-1} \leq p_j \leq \dots \leq p_J$. Given the partial solution $p_1, \dots, p_{j-1}$, the set of prices to test for alternative j is thus:

$$\bar{\mathcal{P}}_j = \left\{\{\bar{p}_j^{nr} : n \in \mathcal{N}, r \in \mathcal{R}\} \cup \{p_j^U\}\right\} \cap \left[\max\{p_j^L, p_{j-1}\}, p_j^U\right]$$

In Proposition 4.1, we show that the BEA, by recursively exploring all the prices of set $\bar{\mathcal{P}}_j$ at each step, yields an optimal solution.

Proposition 4.1. *There exists an optimal solution p^* to the CPP that respects $p_j^* \in \bar{\mathcal{P}}_j \ \forall j \in \{1, \dots, J\}$.*

Proof. Let p^1 be a feasible solution such that $p_1^1 \leq \dots \leq p_J^1$ and $p_j^1 \notin \bar{\mathcal{P}}_j$ for a given alternative $j \in \{1, \dots, J\}$. We define the solution p^2 as follows:

$$p_i^2 = \begin{cases} p_i^1 & \text{if } i \neq j \\ \min\{q \in \bar{\mathcal{P}}_j : q > p_j^1\} & \text{if } i = j \end{cases}$$

Since $p_j^U = \max \bar{\mathcal{P}}_j$, the price p_j^2 is always feasible and well-defined. Solution p^2 thus preserves the feasibility of solution p^1. Since the pricing coefficients β_p^{in} are assumed to be negative for each alternative i and each customer n, the consequence of increasing the price of alternative j can only impact the behavior of the simulated customers who select alternative j in solution p^1, i.e., such that $\omega_{jnr}^1 = 1$. Since p_i^2 is defined as the smallest element of $\bar{\mathcal{P}}_j$ larger than p_j^1, and since the customers are assumed to select the most expensive alternative among the alternatives of maximum utility in case of equality, no customer will switch from j to a cheaper alternative $i \in \{0, \dots, j-1\}$ when increasing the price of alternative j from p_j^1 to p_j^2. Consequently, the revenue $\eta_{jnr}^1 = p_j^1$ that was obtained from a simulated customer (n, r) such that $\omega_{jnr}^1 = 1$ can either increase to $\eta_{jnr}^2 = p_j^2$ if j remains the maximum utility choice for (n, r) in solution p^2, or it can reach $\eta_{jnr}^2 = p_i^2$ for some alternative $i \in \{j+1, \dots, J\}$ if increasing the price of alternative j causes its utility to fall below that of a more expensive alternative. In both cases, we have that $\eta_{jnr}^2 \geq \eta_{jnr}^1$. This means that the objective value of solution p^2 is at least that of p^1. In the case of inequality, we directly conclude that p^1 is suboptimal. Otherwise, a feasible solution that respects the assumption of the proposition's statement and that leads to a revenue at least equal to that of solution p^1 can be constructed by repeating the same procedure until $p_i \in \bar{\mathcal{P}}_i \ \forall i \in \{1, \dots, J\}$. □

The pseudocode of the BEA algorithm is provided in Algorithm 1. We denote by S the set of all possible permutations s of $\{1, \dots, J\}$. For a given permutation $s \in S$, the jth element of the ordered list s in denoted by s_j.

Algorithm 1: Breakpoint exact algorithm (BEA) to solve the CPP

```
Result: optimal solution $p^*$ and value $o^*$ for Formulation 6.
$p_j^* \leftarrow 0 \quad \forall j \in \{1, \dots, J\}$
$o^* \leftarrow 0$
for $s$ in $S$ do
    $p_{s_j} \leftarrow 0 \quad \forall j \in \{1, \dots, J\}$
    $h_{nr}^{s_1} \leftarrow c_{0nr} \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}$
    $\eta_{nr} \leftarrow 0 \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}$
    $(\hat{p}, \hat{o}) \leftarrow$ enumerate$(s, p, h^{s_1}, \eta, 1)$
    if $\hat{o} > o^*$ then
        $p^* \leftarrow \hat{p}$;
        $o^* \leftarrow \hat{o}$;
    end
end
return $(p^*, o^*)$
```

Algorithm 1 iterates over all the possible orderings of prices $p_{s_1} \leq p_{s_2} \leq \dots, \leq p_{s_J}$, $s \in S$. Each restricted problem is solved by the recursive `enumerate` function, whose pseudocode is provided in Algorithm 2. This function takes as input the current permutation $s \in S$ of alternatives, a partially populated vector of prices p, whose components $p_{s_1} \leq \dots \leq p_{s_{j-1}}$ have been previously fixed, the utility $h_{nr}^{s_j}$ and the price η_{nr} of the alternative currently selected by each simulated customer $(n, r) \in \mathcal{N} \times \mathcal{R}$, and the depth j of exploration in the current ordering.

The function `enumerate` iteratively sets the price of the current alternative s_j to each element of $\bar{\mathcal{P}}_{s_j}$ and recursively solves the resulting restricted problem. At

level j of the recursion, $\mathcal{N}_1$ is the set of simulated customers who select alternative s_j at any feasible price. Set $\mathcal{N}_2$ refers to the set of simulated customers who may or may not select alternative s_j depending on its price.
At level j of the recursion, the alternative s_j, which cannot be cheaper than any of the previously introduced ones, is added to the product line. Its price p_{s_j} is initialized to its upper bound $p^u_{s_j}$. The customers' behavior is adjusted accordingly, and the incumbent solution p^* and its objective value o^* of the current restricted problem are initialized. If $j \leq J - 1$, the price of the current alternative is iteratively fixed to each value $p_{s_j} \in \bar{\mathcal{P}}_j$, the maximum utility alternative is updated for each customer, and the resulting subproblem is solved recursively. We denote its optimal solution by $\hat{p}$ and update the incumbent solution p^* if the objective value $\hat{o}$ exceeds the incumbent value o^*. If $j = J$, the current alternative is the last and most expensive one. We denote by $\tilde{o}$ the revenue generated by the customers selecting alternatives other than s_J. Each time a breakpoint $\bar{p}^{n_i r_i}_{s_J}$ is visited, the corresponding customer (n_i, r_i) switches from their previously selected alternative to s_J, which reduces by $\eta_{n_1 r_1}$ the revenue $\tilde{o}$ generated by alternatives s_1 to s_{J-1}. At this point, $|\mathcal{N}_1| + i$ customers select alternative J, for a total objective value of $\tilde{o} + (|\mathcal{N}_1| + i)\bar{p}^{n_i r_i}_{s_J}$. The incumbent solution is updated.
The solution $\hat{p}$ with value $\hat{o}$ that is returned to Algorithm 1 for each is the optimal solution to the initial problem, only restricted by the ordering $s \in S$ of the prices. At the end of the execution of BEA, all the solutions p^* to the CPP that respects $p^*_j \in \bar{\mathcal{P}}_j \; \forall \, j \in \{1, \ldots, J\}$ have been explored and the solution p^* with value o^* returned by Algorithm 1 is optimal, by Proporition 4.1.

Algorithm 2: Recursive enumeration function within BEA

```
Function enumerate($(s, p, h^{s_j}, \eta, j)$):
    $p_{s_j} \leftarrow p^U_{s_j}$
    $\bar{p}^{nr}_{s_j} \leftarrow \frac{h^{s_j}_{nr} - c_{s_j nr}}{\beta^{s_j n}_p} \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}$
    $\mathcal{N}_1 \leftarrow \{(n, r) | \bar{p}^{nr}_{s_j} \geq p^U_{s_j}\}$
    $\mathcal{N}_2 \leftarrow \{(n, r) | \max\{p^L_{s_j}, p_{s_{j-1}}\} < \bar{p}^{nr}_{s_j} < p^U_{s_j}\}$
    $h^{s_{j+1}}_{nr} \leftarrow h^{s_j}_{nr} \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R} \setminus \mathcal{N}_1$
    $h^{s_{j+1}}_{nr} \leftarrow c_{s_j nr} + \beta^{s_j n}_p p^U_{s_j} \quad \forall (n, r) \in \mathcal{N}_1$
    $\eta_{nr} \leftarrow p^U_{s_j} \quad \forall (n, r) \in \mathcal{N}_1$
    $o^* \leftarrow \sum_{n \in \mathcal{N}} \sum_{r \in \mathcal{R}} \eta_{nr}$
    $p^* \leftarrow p$
    Sort the elements of $\mathcal{N}_2$ so that $\bar{p}^{n_1 r_1}_{s_j} \geq \bar{p}^{n_2 r_2}_{s_j} \geq \cdots \geq \bar{p}^{n_{|\mathcal{N}_2|} r_{|\mathcal{N}_2|}}_{s_j}$
    if $j \leq J - 1$ then
        for $i \in \{1, \ldots, |\mathcal{N}_2|\}$ do
            $p_{s_j} \leftarrow \bar{p}^{n_i r_i}_{s_j}$
            $h^{s_{j+1}}_{nr} \leftarrow c_{s_j nr} + \beta^{s_j n}_p p_{s_j} \quad \forall (n, r) \in \{(n_1, r_1), \ldots, (n_i, r_i)\} \cup \mathcal{N}_1$
            $\eta_{nr} \leftarrow p_{s_j} \quad \forall (n, r) \in \{(n_1, r_1), \ldots, (n_i, r_i)\} \cup \mathcal{N}_1$
            $(\hat{p}, \hat{o}) \leftarrow$ enumerate($(s, p, h^{s_{j+1}}, \eta, j + 1)$)
            if $\hat{o} > o^*$ then
                $o^* \leftarrow \hat{o}$
                $p^* \leftarrow \hat{p}$
            end
        end
    end
    else
        $\tilde{o} \leftarrow o^* - \sum_{(n,r) \in \mathcal{N}_1} \eta_{nr}$
        for $i \in \{1, \ldots, |\mathcal{N}_2|\}$ do
            $\tilde{o} \leftarrow \tilde{o} - \eta_{n_i r_i}$
            $p_{s_j} \leftarrow \bar{p}^{n_i r_i}_{s_j}$
            $o \leftarrow \tilde{o} + (|\mathcal{N}_1| + i) p_{s_j}$
            if $o > o^*$ then
                $o^* \leftarrow o$
                $p^* \leftarrow p$
            end
        end
        return ($p^*$, $o^*$)
    end
end
```

Proposition 4.2. *The time complexity of the BEA algorithm is* $O(J!(NR)^J \log(NR))$.

Proof. Until level $J-1$, the algorithm keeps track of the utility $h_{nr}^{s_j}$ of the alternative currently selected by each simulated customer and its price η_{nr}. These updates lead to operations with time complexity $\mathcal{O}(NR)$ at each passage in the for loop from level 1 to $J-1$. When setting the last price s_J, however, only the objective value o of the current solution is computed, by updating at each breakpoint the revenue generated by the number of captured customers times the current breakpoints. This allows the operations in the innermost loops of the algorithm to be executed in constant time. The dominant operation at level J is thus the sorting step, with time complexity $\mathcal{O}(NR\log(NR))$. Since $\mathcal{O}(NR)$ recursive calls are made at each level, the total time complexity of the BEA algorithm for a given permutation s is $\mathcal{O}((NR)^J\log(NR))$. There are J! possible orderings of the prices. We conclude that the time complexity is $O(J!(NR)^J\log(NR))$ in worst case. $\square$

This complexity exhibits exponential growth with J. However, for a fixed number of prices, it grows polynomially in the number NR of simulated customers. The BEA algorithm can thus be expected to perform well for instances of the CPP with few prices and a high number of simulated customers.

5 Spatial Branch and Bound algorithm

The key property of the CPP, that we are trying to exploit in the following algorithm, is that for a fixed price, the problem becomes trivial to solve. Indeed, it reduces to assigning the value 1 to the choice variable ω_{inr} if for individual n, in scenario r, the highest utility is the one of alternative i and 0 otherwise. Furthermore, with the QCQP-L formulation (Formulation 6), all non-convexity of the problem is contained in a set of bilinear equality constraints. We want to make use of both of these properties, together with additional features, to design an efficient solution method for the CPP.
Starting from the QCQP-L formulation and given a set of bounds $p_i \in [p_i^L, p_i^U]\ \forall i \in J$, we relax the constraints $\eta_{inr} = p_i\omega_{inr}\ \forall i \in C_n$ by the use of a McCormick envelope, see McCormick (1976):

$$\begin{aligned}
\eta_{inr} &\geq p_i^L\omega_{inr}\\
\eta_{inr} &\geq p_i^U\omega_{inr} + p_i - p_i^U\\
\eta_{inr} &\leq p_i^L\omega_{inr} + p_i - p_i^L\\
\eta_{inr} &\leq p_i^U\omega_{inr}
\end{aligned}$$

This yields the McCormick relaxation of the QCQP-L shown in Formulation 7. We then employ a Spatial Branch and Bound algorithm, see for example Liberti

(2008), to find the best bounds for all the prices. A conceptual outline of the method is given below:

1. Solve the McCormick relaxation (Formulation 7) using the initial bounds.
2. From the solution value of the prices, compute the corresponding choices and construct a feasible solution.
3. Choose a price to branch on (see Section 5.1), then split the search interval for that price, i.e., its bounds, into two, while all other price bounds remain the same. Add two new nodes to the Branch and Bound tree, each corresponding to one set of the new bounds.
4. Choose the next node from the tree (see Section 5.3) based on the achieved objective value in its parent node (*best-first-search*), and solve the relaxation with the bounds corresponding to that node.
5. Continue until the relative gap between the objective value of the tightest relaxation is close enough (up to a predefined relative optimality gap) to the best feasible solution found.

The choice of using *best-first-search* is supported by the non-convex search space and the fact that we can easily generate feasible solutions, and thus lower bounds, removing the need to go deep into the tree early. The great benefit of such a custom Branch and Bound approach is that instead of branching on the JNR variables η_{inr}, as a conventional solver would, we only branch on the J price variables, which drastically reduces the computational effort.

$$
\begin{aligned}
&\max_{p,\omega,\eta,U,h} \frac{1}{R}\sum_{r\in\mathcal{R}}\sum_{n\in\mathcal{N}}\sum_{i\in C_n\cup\{0\}} \eta_{inr} \\
&\text{s.t.} \\
&\sum_{i\in C_n\cup\{0\}} \omega_{inr} = 1 && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\mu_{nr}) \\
&h_{nr} = c_{0nr}\omega_{0nr} \\
&\qquad + \sum_{i\in C_n} [c_{inr}\omega_{inr} + \beta_p^{in}\eta_{inr}] && \forall n\in\mathcal{N}, r\in\mathcal{R} && (\zeta_{nr}) \\
&h_{nr} \geq U_{inr} && \forall i, n\in\mathcal{N}, r\in\mathcal{R} && (\alpha_{inr}) \\
&U_{inr} = c_{inr} + \beta_p^{in} p_i && \forall i, n\in\mathcal{N}, r\in\mathcal{R} && (\kappa_{inr}) \\
&\eta_{inr} \geq p_i^L \omega_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\lambda^1_{inr}) \\
&\eta_{inr} \geq p_i^U \omega_{inr} + p_i - p_i^U && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\lambda^2_{inr}) \\
&\eta_{inr} \leq p_i^L \omega_{inr} + p_i - p_i^L && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\lambda^3_{inr}) \\
&\eta_{inr} \leq p_i^U \omega_{inr} && \forall n\in\mathcal{N}, i\in C_n, r\in\mathcal{R} && (\lambda^4_{inr}) \\
&\omega \in [0,1]^{(J+1)NR} \\
&p \in [p_1^L, p_1^U] \times \ldots \times [p_J^L, p_J^U] \\
&\eta, U, h \in \mathbb{R}^{JNR}, \mathbb{R}^{JNR}, \mathbb{R}^{NR}
\end{aligned}
\tag{7}
$$

5.1 Branching strategy

If we deal with more than one price, an important decision we have to make at each branching is which alternative's price to branch on. The traditional approach of Spatial Branch and Bound is to always select the alternative i where the interval $[p_i^L, p_i^U]$ is the largest *(longest-edge-branching)*. This is comparable to a strategic exhaustive search. In our case, we utilize a custom branching rule instead, where we branch along the asset which displays the largest average absolute violation of the constraints $\eta_{inr} = p_i\omega_{inr}$ over all $i \in \{1, \ldots, J\}$:

$$\texttt{branching\_index} \quad = \quad \arg\max_{i\in\{1,\ldots,J\}} \left\{ \textstyle\sum_{nr} |\overline{\eta}_{inr} - \overline{p}_i\overline{\omega}_{inr}| \right\}$$

where $\overline{p}, \overline{\omega}, \overline{\eta}$ are the solution values in the current relaxation. The motivation behind this rule is that we know that for an optimal solution, the violation of this constraint needs to be zero, thus exploring the space where there is the most potential to reduce this violation promises faster convergence, which is also the case empirically (especially for instances with higher number of prices). Furthermore, after each branching, the bounds improve and the McCormick relaxation

gets tighter. Thus the largest average absolute constraint violation decreases, implying convergence over time.
Other branching strategies like probabilistic branching based on market shares or profits, as well as a fully random approach, were computationally explored in preliminary experiments, all performing significantly worse than this deterministic strategy.

5.2 Dominating and dominated alternatives

The further down the Branch and Bound tree we progress, the tighter the bounds on the prices. This indicates that for certain tuples of individuals and scenarios, it can happen that one alternative is either dominating or dominated over the entire set of bounds in the current node, meaning that for any set of prices found within these bounds, the alternative either always has the highest utility amongst all alternatives (dominating) or there exists at least one alternative that always has higher utility than that alternative (dominated). Formally we define an alternative i to be *dominating* for an individual n and scenario r, if given a set of bounds $[p_i^L, p_i^U]_{i \in \{1,\dots,J\}}$, the following holds:

$$U_{inr}(p_i^U) \;\; \geq \;\; U_{jnr}(p_j^L) \quad \forall j \in \{1, \dots, J\} \setminus \{i\}$$

i.e. its utility, given the least attractive (highest possible) price within the bounds, is higher than all other utilities given their best (lowest) prices, making it guaranteed to be chosen. In the same way, an alternative i is described to be *dominated* if:

$$U_{inr}(p_i^L) \;\; \leq \;\; U_{jnr}(p_j^U) \quad \forall j \in \{1, \dots, J\} \setminus \{i\}$$

i.e. its utility, given the most attractive (lowest possible) price within the bounds, is lower than all other utilities given their worst (highest) prices, making it guaranteed to never be chosen. If for a tuple there exists an alternative that is dominating, we can remove that tuple entirely from the current node and simply add the price of the dominating alternative to the total profit after the optimization (except if the dominating alternative is the opt-out). If for a tuple (n, r) there exists an alternative j that is dominated, we can add the constraint that $\omega_{jnr} = \eta_{jnr} = 0$, simplifying the problem. The closer we get to the optimal solution, i.e. to a breakpoint (as discussed in section 3) the more likely it is that two utilities take very similar values, to the point where comparisons become a question of numerical precision. If that is the case, we define the alternative that brings more profit, i.e. has a higher price, to be the dominant one (this is also what mathematical solvers do in such cases, as the goal is to maximize the objective).
Applying this approach leads to large speed-ups in convergence, especially in the later parts of the algorithm, which is notorious for being the most time-consuming.

5.3 Enumeration strategy

The enumeration strategy concerns the choice of which node to select next when exploring a given branch in the Branch and Bound tree. As mentioned above, the first criterion is the upper bound on the objective from its parent node, i.e., the best possible objective value that can be achieved when exploring that node. However, we will generally have multiple nodes with the same upper bound, each exploring a disjoint search space, raising the question of which space to explore first. Here we opt to always select the node where the number of market captures is maximized, i.e., the node with the fewest people that, given the bounds on the prices in that node, are guaranteed to choose the opt-out option (the opt-out is the dominating alternative). We can formally define the number of guaranteed opt-outs given a node j with its lower price bounds $(p_j^L)_{i\in\{1,\dots,J\}}$ as follows:

$$\sigma(j) \;:=\; \#\{(n,r) \mid U_{0nr} > U_{inr}((p_j^L)_i)\ \forall i \in \{1,\dots,J\}\}$$

i.e., the number of (simulated) customers, for which the highest achievable utilities of the priced options are all strictly smaller than the utility of the opt-out alternative.

As the penalty coefficients in the utility are negative, lower price ranges will always capture more customers, yielding a lower number of guaranteed opt-outs. Thus this strategy coincides with preferring the node that explores lower prices over the one exploring higher prices after branching.

Let Ω denote the set of all currently open nodes in the tree, where a node j consists of a set of bounds Δ^j, its upper bound on the objective value $\hat{o}_j$ and the number of guaranteed opt-outs $\sigma(j)$. The next node to be selected from the tree can now be written as:

$$j^* \;:=\; \arg\min_{j\in\Omega}\{\sigma(j) \mid \hat{o}_j = \max\{\hat{o}_i \mid \{\Delta^i,\ \hat{o}_i\} \in \Omega\}\}$$

implying that, among the nodes that reach the highest upper bound on their potential objective value, we choose the one that has the lowest number of guaranteed opt-outs.

5.4 Finding an initial solution

The Branch and Bound algorithm is initiated with a starting solution, i.e., a set of prices for which we compute the integer objective value. This objective value serves as the initial lower bound for the optimal objective value of the solution. It reduces the search tree size, since if we explore a branch and the relaxation in this branch already yields an objective value that is worse than the current best lower bound, there is no need to continue along that branch. It is thus beneficial to start with a good initial guess for the price, for which we use a simple sampling heuristic: We sample 5 customers, one single scenario, and then solve this reduced problem using the QCQP-L formulation with a mathematical solver. This heuristic generally provides decent initial solutions in a short amount of time. Other, more sophisticated heuristics might lead to further improvements in the overall performance of the algorithm.

5.5 Utilizing breakpoints

If one can a priori assume which one of the products will always have the lowest price out of all of them, for example when offering different amounts of the same product, we can make use of the insights about breakpoints from Section 3.4. By computing all the breakpoints for the cheapest controlled alternative in advance, after each branching, we can cut off the regions where no breakpoints are contained and thus reduce the search space.

Algorithm 3 provides the pseudo-code for the Branch and Bound procedure described in this section.

Algorithm 3: A Spatial Branch & Bound algorithm to solve the CPP

```
     Result: $\texttt{perc}_{\texttt{tol}}$-optimal solution $(p^*, \omega^*, \eta^*)$ for Formulation 6.
(1)  Initialization: Set $j := 0$, $\Delta^j := [p_1^L, p_1^U] \times \cdots \times [p_J^L, p_J^U]$, $o^* := o^{\text{heuristic}}$,
       $\hat{o}_j := \infty$, $\Omega := \{\{\Delta^j,\ \hat{o}_j\}\}$
(2)  while $\frac{\max_j\{\hat{o}_j\} - o^*}{o^*} \cdot 100 \le \texttt{perc}_{\texttt{tol}}$ do
(3)      let $j := \text{argmin}\{\sigma(j) \mid \hat{o}_j = \max\{\hat{o}_i \mid \{\Delta^j,\ \hat{o}_j\} \in \Omega\}\}$. Remove $\{\Delta^j,\ \hat{o}_j\}$
           from $\Omega$ and solve Formulation 7 with bounds $\Delta^j$. Denote its optimal
           solution by $(p_j, \omega_j, \eta_j)$ and its optimal objective value by $o_j$ as well
           as its integer optimal value $\bar{o}_j$.
(4)      if $\bar{o}_j > o^*$ then
(5)          compute $\bar{\omega}_j, \bar{\eta}_j$ from $p_j$ and set $o^* = o_j$,
               $(p^*, \omega^*, \eta^*) := (p_j, \bar{\omega}_j, \bar{\eta}_j)$, delete from $\Omega$ all instances $\{\Delta^j,\ \hat{o}_j\}$
               where $\hat{o}_j \le o^*$.
(6)      end
(7)      if $o_j > o^*$ then
(8)          let $i = \text{argmax}\left\{\frac{\sum_{nr} |\bar{\eta}_{inr} - \bar{p}_i \bar{\omega}_{inr}|}{NR}, i \in C_n\right\}$ and divide the interval
               $[p_i^L, p_i^U]$ into two new intervals $[p_i^L, \frac{p_i^L + p_i^U}{2}]$ and $[\frac{p_i^L + p_i^U}{2}, p_i^U]$.
               Construct the two new subpolyhedra $\Delta'$ and $\Delta''$. Define
               $\hat{o}' = \hat{o}'' := o_j$ and augment $\Omega = \Omega \cup \{\Delta', \hat{o}'\} \cup \{\Delta'', \hat{o}''\}$.
(9)      end
(10) end
```

The algorithm is initialized with the full set of bounds for each price and the lower bound on the objective value generated by the starting heuristic. We select the node with the highest upper bound and solve the related relaxation. Next we fix the prices to their solution values to compute the corresponding integer solution (line 3). If it is better than the current best integer solution, we update the latter (lines 4-5). If the objective value of the relaxation is larger than the current best integer objective value, we select a price to branch on, create the two new nodes and add them to the tree (lines 7-9). The algorithm terminates when the relative gap between the best lower bound and upper bound is higher than a predefined tolerance $\texttt{perc}_{\texttt{tol}}$.

6 Benders decomposition

In every node of the Branch and Bound tree, we need to solve Formulation 7, which may be time-consuming due to a large number of variables η and ω. However, Formulation 7 is highly separable: indeed, if all variables p_i are fixed to a certain value, the utility maximization problem can be solved for every individual and scenario independently. This is why we consider a Benders decomposition approach to speed up the solution of the McCormick relaxation in each node of the Branch and Bound tree, see for example Rahmaniani et al. (2017). It is worth noting that Benders is most commonly applied in the context of MILP optimization, however, here we use it to speed up the computation of a separable LP. The idea is to derive a valid problem reformulation by projecting out the ω, η, h and U variables from Formulation 7, and replacing them with auxiliary variables $\mathcal{P}_{nr}$ whose value will represent the revenue of customer n in scenario r. A conceptual outline of the decomposition method is given below:

1. Choose an initial set of values for the prices. Compute the integer objective value to get a lower bound on the optimal objective value.

2. Solve the Benders subproblem (Formulation 9) for all individuals n and scenarios r separately, with the price variables fixed to the initial prices.

3. Extract the dual values of the constraints fixing the prices and add the following Benders cuts to the master problem (Formulation 8):

$$\mathcal{P}_{nr} \;\leq\; \mathcal{P}^c_{nr} - \sum_i \varphi^c_{inr}(p_i - p^c_i)$$

   where p_i and $\mathcal{P}_{nr}$ are variables in the master problem representing the price of alternative i and the obtained revenue from tuple (n, r) respectively. For a cut c, p^c are the values the prices are fixed to, $\mathcal{P}^c_{nr}$ is the revenue generated from tuple (n, r) with these prices and φ^c_{nr} is the dual vector corresponding to constraints $p_i = p^c_i, i \in C_n$.

4. Solve the master problem with these NR optimality cuts and retrieve the optimal solution values for the prices. Compute the integer objective value and update the best lower bound if necessary.

5. Solve the subproblem with the price variables fixed to the new set of prices to obtain the dual values for the next optimality cuts.

6. We iterate the process until the objective value of the master problem is close enough to the one of the best feasible solution found.

Benders decomposition works by decomposing the original problem into a master problem and a subproblem, where the master problem is a relaxation of the original problem that iteratively is improved by the addition of optimality and feasibility cuts.
As the subproblem can be solved independently for each individual and scenario, we can similarly add a Benders cut for each of them separately (called *multi-cut* or *disaggregate* Benders). Adding all NR cuts at once is not a common practice in Benders decomposition, however, our preliminary tests have revealed that the number of iterations and the overall time needed to solve the LP relaxation are drastically reduced when compared to the traditional implementation of adding only violated Benders cuts in each iteration. More detailed information on the implementation of the cuts is given in Appendix B.1.
Formulation 8 shows the Benders master problem, where the variables consist of the revenue $\mathcal{P}_{nr}$ for each individual n and scenario r and the complicating variables, i.e. the prices, p_i. Each Benders cut $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the set of all currently added cuts, consists of the candidate price p_i^c used in the fixing constraint of the subproblem, the resulting revenue $\mathcal{P}_{nr}^c$ for each tuple (n, r) and the corresponding dual variable values φ_{inr}^c. Constraints (a_{cnr}) represent the Benders cuts, constraint (b) is included to ensure that the initial LP solved for $\mathcal{C} = \emptyset$ is not unbounded, where $\mathcal{P}^{UB}$ is a reasonable upper bound on the total revenue.

$$
\begin{aligned}
&\max_{\mathcal{P},p} \sum_{n\in\mathcal{N}} \sum_{r\in\mathcal{R}} \mathcal{P}_{nr} \\
&\text{s.t.} \\
&\mathcal{P}_{nr} \leq \mathcal{P}_{nr}^c - \sum_i \varphi_{inr}^c (p_i - p_i^c) \quad \forall c \in \mathcal{C}, n \in \mathcal{N}, r \in \mathcal{R} \quad (a_{cnr}) \\
&\sum_{nr} \mathcal{P}_{nr} \leq \mathcal{P}^{UB} \qquad\qquad\qquad\qquad\qquad\qquad\qquad (b) \\
&p \in [p_1^L, p_1^U] \times \ldots \times [p_J^L, p_J^U] \\
&\mathcal{P} \in \mathbb{R}^{NR}
\end{aligned}
\tag{8}
$$

Formulation 9 shows the subproblem for an individual n and scenario r. It corresponds to Formulation 7 (reduced to one tuple) with the only addition being constraints (φ_{inr}), fixing p_i to p_i^c. We notice that for any price vector p^c, the resulting Benders subproblem is feasible and an optimal solution is obtained. Therefore, no Benders feasibility cuts are needed. To speed up the generation of optimality cuts, we set a time limit for the computation of the master problem. This means we do not solve it to optimality, i.e., we generate suboptimal candidate solutions for the subproblem, potentially weakening the resulting optimality cuts. However, we observed that a large amount of time in Benders was spent on proving optimality of the solution for the master problem, which is only of key importance in the last iteration of Benders, to prove convergence.

$$
\begin{aligned}
\mathcal{P}^c_{nr} = \max_{p,\omega,\eta,U,h} \; & \frac{1}{R}\sum_{i\in C_n} \eta_i \\
\text{s.t.} \quad & \\
& \sum_{i\in C_n\cup\{0\}} \omega_i = 1 && && (\mu) \\
& h = c_{0nr}\omega_0 + \sum_{i\in C_n} [c_{inr}\omega_i + \beta_p^{in}\eta_i] && && (\zeta) \\
& h \geq c_{0nr} && && (\alpha_0) \\
& h \geq U_i && \forall i \in C_n && (\alpha_i) \\
& U_i = c_{inr} + \beta_p p_i && \forall i \in C_n && (\kappa_i) \\
& \eta_i \geq p_i^L \omega_i && \forall i \in C_n && (\lambda_i^1) \\
& \eta_i \geq p_i^U \omega_i + p_i - p_i^U && \forall i \in C_n && (\lambda_i^2) \\
& \eta_i \leq p_i^L \omega_i + p_i - p_i^L && \forall i \in C_n && (\lambda_i^3) \\
& \eta_i \leq p_i^U \omega_i && \forall i \in C_n && (\lambda_i^4) \\
& p_i = p_i^c && \forall i \in C_n && (\varphi_{inr}^c) \\
& \omega \in [0,1]^{|C_n|+1} \\
& p \in \bigtimes_{i\in C_n} [p_i^L, p_i^U] \\
& \eta, U, h \in \mathbb{R}^{|C_n|}, \mathbb{R}^{|C_n|}, \mathbb{R}
\end{aligned}
\tag{9}
$$

7 Numerical results

In this Section, we present the parking space operator case study to evaluate the procedures presented in the previous sections. We describe the performed experiments, comment on the numerical results, and give a summary on our findings.

7.1 Case study

To test the presented methodology we rely on the same case study as Paneque et al. (2021), as it uses a mixed logit model published in the literature, illustrating the fact that there is no need for any assumption about the choice model to apply the methodology. The case study concerns a parking services operator, motivated by the published disaggregate demand model for parking choice by Ibeas et al. (2014). The exact specification of the model with all estimated parameters can be found in Appendix A. The choice set consists of three services: paid on-street parking (PSP), paid parking in an underground car park (PUP), and free on-street parking (FSP). The prices of PSP and PUP are controlled by a single operator trying to maximize profit, whereas the FSP alternative does not provide any revenue and thus represents the opt-out option. For this case study, we assume that all customers must pay the same price for the same service. Furthermore, no capacity constraints are imposed on the parking facilities.

7.2 Extension to more than two prices

In order to test the performance of our methods on instances with a larger number of prices, we artificially extend the choice set as follows. First, we define:

$$\begin{aligned} \mathrm{J}_{\mathrm{PSP}} &:= \text{ Number of paid on-street parking alternatives.} \\ \mathrm{J}_{\mathrm{PUP}} &:= \text{ Number of underground parking alternatives.} \end{aligned}$$

where for the non-extended choice set we have $\mathrm{J}_{\mathrm{PSP}} = \mathrm{J}_{\mathrm{PUP}} = 1$ and the total number of prices to be optimized is $\mathrm{J} = \mathrm{J}_{\mathrm{PSP}} + \mathrm{J}_{\mathrm{PUP}}$. To add more PSP or PUP options we duplicate the respective alternative and increase the access time from the parking space to desired destination by one minute per duplicate. This corresponds to augmenting the parking space facilities in size.

7.3 Description of experiments

The goal of our experiments is to answer the following questions:

1. How does relaxing the integer choice variables to continuous ones, i.e. going from the MILP to the QCQP formulation, affect runtime, and is there a difference when isolating the non-convex constraints, i.e. going from the QCQP to the QCQP-L formulation? How does solving these different Formulations using GUROBI compare to our custom algorithms on small to medium-sized instances?

2. How do our custom algorithms (Spatial Branch and Bound, Spatial Branch and Benders, and breakpoint exact algorithm) compare to solving the QCQP-L formulation directly with GUROBI on instances with an increased number of draws?

3. What is the highest number of draws we can solve instances for (up to an optimality gap of up to 5%), within a time limit of 72 hours, using the Spatial Branch and Benders or breakpoint exact algorithm? Will the solution values converge at a certain number of draws?

4. How do the different solution methods fare when the number of prices is increased?

To investigate these four issues we perform the tests described in Table 1, where N denotes the number of individuals considered and R the number of scenarios generated. For the rest of this section, we will use "MILP", "QCQP-L", and "QCQP" to express when we use a mathematical solver to directly solve the problems given in Formulation 4, 5 and 6 respectively. Furthermore, "BEA" refers to the breakpoint exact algorithm presented in Section 4, "B&B" to the Spatial Branch and Bound algorithm presented in Section 5, and lastly, "B&BD" refers to the Spatial Branch and Bound with the addition of solving each node using a Benders decomposition as presented in Section 6.

Table 1 – Summary of Tests

	Test 1	**Test 2**	**Test 3**	**Test 4**
J_{PSP}	1	0, 1	0, 1	0, 1, 2, 3
J_{PUP}	1	1	1	1, 2, 3
N	50	50	50	50, 20
R	200, 300, 400, 500	1,000, 2,000, . . . , 7,000	2,000, 4,000, . . . , 30,000, 100k, 500k, 1m	200
Time limit	72 hours	24 hours	72 hours	24 hours
Methods	MILP, QCQP, QCQP-L, B&B, B&BD, BEA	QCQP-L, B&B, B&BD, BEA	B&BD, BEA	QCQP-L, B&B, B&BD, BEA
Optimality tolerance	0.01%	0.01%	5%	0.01%

The bounds for all prices are defined to be 0€ for the lower bound and 2€ for the upper bound. When optimizing one price only, we fix the price of PSP to be 0.6€. The accelerated cut generation discussed in Section 6 is applied with a time limit for the Benders Master problem that is dependent on the number of scenarios and set to $R/10$. All experiments are performed using GUROBI 10.0.3 (Gurobi Optimization, LLC, 2021) with one thread, on a computational cluster node with two 2.4 GHz Intel Xeon Platinum 8360Y processors, where we utilize 16 cores with a total of 64 GB of RAM. We use GUROBI's hyperparameter tuning tool to maximize the performance of all methods (see Appendix B.2). All runtimes are reported in seconds. Whenever the time limit is reached, we report the achieved relative optimality gap instead.

7.4 Results and analysis

For each of the four test series, we present the runtimes, optimal prices p_i, $i \in \{1, \ldots, J\}$, and objective values of the solution (i.e., the attained profit).
Table 2 shows the runtime and results respectively for Test 1, where we optimize two prices with the integer formulation (MILP) vs. the ones with continuous choice variables. Compared to the MILP, we get a speed-up factor of up to 7x for the QCQP, 9x for the QCQP-L, 30x for the B&B, and 20x for the B&BD. Here we see that for lower numbers of draws, the Benders decomposition, due to its sizable overhead, does not bring further improvement over the Branch and Bound approach. The breaking point exact algorithm (BEA) outperforms all methods with a speed-up of factor up to 3519x over the MILP. In general, we see that the relative increase in runtime for the MILP is much greater than for the other methods, implying that the larger the instance, the bigger the improvements in performance. The optimal prices and profit are identical for all methods.
For Test 2, Table 3 depicts the results for one-price optimization with large numbers of draws, and Tables 4 and 5 show the results when optimizing both prices simultaneously. When optimizing the price of PUP only, we achieve a speed-up factor over GUROBI (using the QCQP-L formulation) of up to 9.8x with the B&B and up to 12.7x with the B&BD method. For the BEA, it is hard to measure the speed-up factor because it is of several magnitudes. The largest measurable factor is for 5000 draws, where the QCQP-L takes 83651 seconds and the BEA finds the optimal solution in 0.05 seconds, giving a speed-up of factor $1.6 * 10^6$. For two-price optimization, the B&BD method solves instances to $\sim 1\%$ optimality for up to 4000 draws, and from there, it manages to get a significantly smaller optimality gap as well as superior feasible solutions within the time limit when compared to the B&B, and drastically smaller gaps when compared to the GUROBI methods. Indeed, for 4000 draws and above, GUROBI does not find a first feasible solution within the time limit. The BEA manages to solve all instances to optimality in

less than a quarter of the time limit.
Tables 6 and 7 show the results for one-price and two-price optimization respectively in Test 3 (exploring the highest possible number of draws we can solve instances with). With the B&BD method, we are able to solve one-price instances with up to 20000 draws to 5% optimality in less than 3.5 hours, and instances with two prices and up to 10000 draws within the time limit of 72 hours. The BEA solves instances with only one price within a fraction of a second, which is why additional instances were added with $R = 100k, 500k, 1m$ to show the way the runtime scales. What we see is that we can solve instances with one million draws in 77 seconds, again outperforming the state-of-the-art and any other proposed method by several orders of magnitude. For two prices, the BEA solves all instances to optimality significantly faster than the B&BD method, allowing for an increase of the number of scenarios of up to 22k (for which we no longer invoked the B&BD method).
Comparing the prices and objective values directly we could not observe convergence for either the one-price or two-price case. However, when using the highest number of draws as a baseline to evaluate the other prices, i.e. compute the profit that each price would have generated for the instance with the highest number of draws, we see a convergence in objective value for the one-price case starting from 6000 draws. For two prices, although the objective value is very consistent, we could not observe convergence to three significant digits.
Finally, Table 8 depicts the results for Test 4, when the number of prices is increased. The prices reported in the table are in each case taken from the method that reported the best lower bound on the profit. It is evident that for one and two prices, the BEA drastically outperforms both the B&B and B&BD methods. However, for three prices, the B&B algorithm performs slightly better, and at four prices and above, the BEA is not capable of solving even very small instances within the 24-hour time limit. To further investigate the runtime difference between the B&B methods and the BEA we additionally consider two very small instances ($N = 20, R = 100$ and $N = 20, R = 50$) with a large time limit of 72 hours, in the hopes of getting a termination from the BEA, however, that did not occur, proving that Branch and Bound methods both strongly outperform the BEA for higher numbers of prices, with observed speed-up factors of up to 14x for the B&B and 20x for the B&BD approach when optimizing four prices simultaneously.
A last additional series of experiments conducted within Test 4 (see Table 9), comparing the B&B and B&BD approaches on instances with four prices, low numbers of customers and draws, and a 24h time limit, shows that the addition of a Benders decomposition to the B&B leads to improved results starting already from 500 draws, compared to 1000 draws for two prices and 3000 draws for one price. A possible explanation for this behavior is that we are projecting out η_{inr}

Table 2 – Test 1: Results for two-price optimization (small-scale instances)

N	R	MILP	QCQP	QCQP-L	B&B	B&BD	BEA	p_1	p_2	Profit
50	200	9,042	4,195	3,403	1,386	3,015	11	0.546	0.679	26.992
50	300	42,804	9,503	7,951	2,998	5,418	24	0.565	0.670	27.119
50	400	112,335	22,567	19,283	6,378	11,858	41	0.556	0.665	27.172
50	500	242,877	32,886	25,030	7,963	12,064	69	0.568	0.678	27.158

Table 3 – Test 2: Results for one-price optimization (large-scale instances)

N	R	QCQP-L	B&B	B&BD	BEA	p_1	Profit
50	1,000	2,435	287	412	0	0.545	23.931
50	2,000	11,018	1,247	1,301	0	0.550	23.946
50	3,000	25,986	2,859	2,521	0	0.555	23.976
50	4,000	53,072	5,234	4,711	0	0.556	23.960
50	5,000	83,651	8,516	6,550	0	0.554	23.945
50	6,000	-	10,785	7,984	0	0.555	23.923
50	7,000	-	14,782	10,556	0	0.551	23.939

and ω_{inr} variables, whose number grows polynomially with the number of prices, N and R, and replace them with NR variables of type $\mathcal{P}$. Hence, the larger the number of prices, the more variables are removed in the solving of the LPs in each node of the spatial branching.

We summarize our findings as follows: Rewriting the problem in a continuous way (QCQP and QCQP-L) already leads to a substantial speed-up over the integer formulation (MILP). When optimizing instances with two prices or fewer, the BEA is by far the best choice, outspeeding other methods by several magnitudes. Conversely, when dealing with three or more prices, the Branch and Bound methods exhibit superior performance. Furthermore, the higher the number of draws, the larger the additional benefit of employing a Benders decomposition within the Branch and Bound approach. Lastly, the larger the number of prices, the lower the number of draws needed for the Spatial Branch and Benders Decomposition method to surpass the pure Spatial Branch and Bound algorithm.

Table 4 – Test 2: Runtime results for two-price optimization (large-scale instances)

		QCQP-L		B&B		B&BD		BEA	
N	R	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)
50	1,000	58,132	0.00	32,963	0.00	21,960	0.00	294	0.00
50	2,000	-	3.60	-	0.57	-	0.05	1,244	0.00
50	3,000	-	20.68	-	10.57	-	0.24	3,005	0.00
50	4,000	-	-	-	33.65	-	1.04	5,834	0.00
50	5,000	-	-	-	40.60	-	3.42	8,931	0.00
50	6,000	-	-	-	55.06	-	5.97	13,568	0.00
50	7,000	-	-	-	67.94	-	3.85	21,133	0.00

Table 5 – Test 2: Numerical results for two-price optimization (large-scale instances)

		QCQP-L			B&B			B&BD			BEA		
N	R	p_1	p_2	Profit	p_1	p_2	Profit	p_1	p_2	Profit	p_1	p_2	Profit
50	1,000	0.573	0.670	27.031	0.573	0.670	27.031	0.573	0.670	27.031	0.573	0.670	27.031
50	2,000	0.584	0.797	26.208	0.592	0.682	26.918	0.564	0.664	27.100	0.564	0.664	27.100
50	3,000	0.532	0.783	22.553	0.584	0.736	26.893	0.571	0.677	27.144	0.564	0.672	27.144
50	4,000	-	-	-	1.242	0.766	25.707	0.567	0.677	27.139	0.567	0.677	27.139
50	5,000	-	-	-	1.041	0.939	21.988	0.554	0.715	27.012	0.572	0.670	27.182
50	6,000	-	-	-	1.004	1.732	18.915	0.590	0.726	26.929	0.571	0.670	27.131
50	7,000	-	-	-	1.339	1.986	18.997	0.572	0.696	27.081	0.569	0.667	27.112

Table 6 – Test 3: Convergence results for one-price optimization *(within 5% optimality gap)

N	R	B&BD	BEA	Price	p_1	Profit based on $R = 1m$
50	2,000	608*	0	0.550	23.946	23.916
50	4,000	1,759*	0	0.556	23.960	23.918
50	6,000	3,248*	0	0.555	23.923	23.920
50	8,000	4,368*	0	0.555	23.910	23.920
50	10,000	5,883*	0	0.555	23.922	23.920
50	12,000	6,412*	0	0.550	23.924	23.920
50	14,000	7,871*	0	0.555	23.912	23.920
50	16,000	9,300*	0	0.555	23.911	23.920
50	18,000	10,733*	0	0.554	23.921	23.920
50	20,000	11,994*	0	0.553	23.926	23.920
50	100,000		5	0.554	23.926	23.920
50	500,000		32	0.554	23.920	23.920
50	1,000,000		77	0.553	23.920	23.920

Table 7 – Test 3: Convergence results for two-price optimization *(within 5% optimality gap)

N	R	B&BD	BEA	p_1	p_2	Profit	Profit based on $R = 22k$
50	2,000	10,699*	1,244	0.565	0.664	27.100	27.107
50	4,000	42,612*	5,834	0.567	0.677	27.139	27.100
50	6,000	113,907*	13,568	0.571	0.670	27.131	27.104
50	8,000	170,053*	27,749	0.565	0.671	27.112	27.109
50	10,000	206,411*	45,096	0.569	0.667	27.115	27.104
50	12,000		57,760	0.566	0.668	27.080	27.108
50	14,000		82,679	0.567	0.669	27.100	27.105
50	16,000		108,539	0.567	0.671	27.114	27.106
50	18,000		163,748	0.568	0.665	27.098	27.103
50	20,000		184,679	0.567	0.672	27.119	27.106
50	22,000		235,602	0.564	0.668	27.110	27.108

Table 8 – Test 4: Results for increasing the number of prices

				QCQP-L		B&B		B&BD		BEA								
N	R	J_{PSP}	J_{PUP}	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	p_1	p_2	p_3	p_4	p_5	p_6	Profit
50	200	0	1	82	-	19	-	47	-	0	-	0.550						23.959
50	200	1	1	3,574	-	1,413	-	3,077	-	12	-	0.546	0.679					26.991
50	200	1	2	46,389	-	34,340	-	54,748	-	39,636	-	0.555	0.648	0.640				26.535
50	200	2	2	-	6.91%	-	1.27%	-	1.13%	-	-	0.559	0.569	0.647	0.643			26.611
50	200	2	3	-	48.80%	-	7.29%	-	4.97%	-	-	0.559	0.569	0.648	0.644	0.635		26.699
50	200	3	3	-	120.20%	-	22.31%	-	15.45%	-	-	0.559	0.570	0.576	0.660	0.644	0.644	26.778
20	100	2	2	-	5.00%	12,478	-	37,395	-	-	-	0.550	0.526	0.587	0.577			10.308
20	50	2	3	-	48.00%	119,946	-	65,359	-	-	-	0.573	0.572	0.629	0.629	0.626		10.588

Table 9 – Test 4: Results for four prices and $N = 20$ customers

				B&B		B&BD		BEA						
N	R	J_{PSP}	J_{PUP}	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	p_1	p_2	p_3	p_4	Profit
20	100	2	2	12478	0.00	37395	0.00	-	-	0.550	0.526	0.587	0.577	10.308
20	200	2	2	29213	0.00	57229	0.00	-	-	0.565	0.578	0.618	0.622	10.496
20	300	2	2	-	0.19	-	0.41	-	-	0.571	0.548	0.632	0.635	10.511
20	500	2	2	-	1.42	-	0.88	-	-	0.559	0.561	0.620	0.625	10.352
20	1000	2	2	-	5.48	-	2.14	-	-	0.549	0.557	0.621	0.619	10.414

8 Conclusions

We proposed the breakpoint exact algorithm (BEA) as well as a Spatial Branch and Bound (B&B) and Spatial Branch and Benders Decomposition (B&BD) approach to tackle the uncapacitated choice-based pricing problem, where demand is captured by an advanced discrete choice model. The procedure can be applied to any choice-based optimization problem and any random utility model, as long as the utilities are linear in the price. The stochasticity in the demand is dealt with using simulation, which leads to a large MILP formulation that is difficult to solve. We show that some properties of the model can be exploited to derive more efficiently solvable versions. The BEA proves to be highly efficient for two prices or fewer, where it drastically outperforms the state-of-the-art as well as the other proposed methods, but is surpassed by the B&B approaches when increasing the number of prices to three or more. Furthermore, the larger the number of simulation scenarios, the bigger the benefit of employing a Benders decomposition within the Spatial Branch and Bound algorithm, with the necessary number of draws for the benefits to show decreasing as the number of prices increases.
For future research, it would be an interesting avenue to expand both the B&B algorithms and the BEA to tackle more complex problems. Capacity constraints, for example, would have to be formulated in a way that does not add new products to the formulation and, crucially, maintains integrality. The investigation of column generation methods could prove to be fruitful in that regard. Currently under development is an efficient heuristic based on the BEA, that is able to deal with large numbers of prices as well as draws and customers, with preliminary results showing great potential. In addition to that, it would be interesting to tackle other choice-based optimization problems, like facility location, revenue management, or equilibrium problems.

References

Akçay, Y., Natarajan, H. P., & Xu, S. H. (2010). Joint dynamic pricing of multiple perishable products under consumer choice. *Management Science*, *56*(8), 1345–1361.

Bertsimas, D., Sian Ng, Y., & Yan, J. (2020). Joint frequency-setting and pricing optimization on multimodal transit networks at scale. *Transportation Science*, *54*(3), 839–853.

Bortolomiol, S., Lurkin, V., & Bierlaire, M. (2021). A simulation-based heuristic to find approximate equilibria with disaggregate demand models. *Transportation Science*, *55*(5), 1025–1045.

Gallego, G., & Wang, R. (2014). Multiproduct price optimization and competition under the nested logit model with product-differentiated price sensitivities. *Operations Research*, *62*(2), 450–461.

Gilbert, F., Marcotte, P., & Savard, G. (2014). Mixed-logit network pricing. *Computational Optimization and Applications*, *57*, 105–127.

Gurobi Optimization, LLC. (2021). *Gurobi Optimizer Reference Manual.* Retrieved from `https://www.gurobi.com`

Hanson, W., & Martin, K. (1996). Optimizing multinomial logit profit functions. *Management Science*, *42*(7), 992–1003.

Ibeas, A., Dell'Olio, L., Bordagaray, M., & Ortúzar, J. d. D. (2014). Modelling parking choices considering user heterogeneity. *Transportation Research Part A: Policy and Practice*, *70*, 41–49.

Labbé, M., Marcotte, P., & Savard, G. (1998). A bilevel model of taxation and its application to optimal highway pricing. *Management science*, *44*(12-part-1), 1608–1622.

Li, H., & Huh, W. T. (2011). Pricing multiple products with the multinomial logit and nested logit models: Concavity and implications. *Manufacturing & Service Operations Management*, *13*(4), 549–563.

Li, H., & Webster, S. (2017). Optimal pricing of correlated product options under the paired combinatorial logit model. *Operations Research*, *65*(5), 1215–1230.

Li, H., Webster, S., Mason, N., & Kempf, K. (2019). Product-line pricing under discrete mixed multinomial logit demand: winner—2017 msom practice-based research competition. *Manufacturing & Service Operations Management*, *21*(1), 14–28.

Liberti, L. (2008). Introduction to global optimization. *Ecole Polytechnique*.

Marandi, A., & Lurkin, V. (2020). An exact algorithm for the static pricing problem under discrete mixed logit demand. *arXiv preprint arXiv:2005.07482*.

McCormick, G. P. (1976). Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, *10*(1), 147–175.

McFadden, D., & Train, K. (2000). Mixed MNL models for discrete response. *Journal of applied Econometrics*, *15*(5), 447–470.

Paneque, M. P., Bierlaire, M., Gendron, B., & Azadeh, S. S. (2021). Integrating advanced discrete choice models in mixed integer linear optimization. *Transportation Research Part B: Methodological*, *146*, 26–49.

Rahmaniani, R., Crainic, T. G., Gendreau, M., & Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, *259*(3), 801–817.

Schlicher, L., & Lurkin, V. (2022). Stable allocations for choice-based collaborative price setting. *European Journal of Operational Research*, *302*(3), 1242–1254.

Song, J.-S. J., Song, Z. X., & Shen, X. (2021). Demand management and inventory control for substitutable products. *Available at SSRN 3866775*.

Sumida, M., Gallego, G., Rusmevichientong, P., Topaloglu, H., & Davis, J. (2021). Revenue-utility tradeoff in assortment optimization under the multinomial logit model with totally unimodular constraints. *Management Science*, *67*(5), 2845–2869.

Talluri, K., & Van Ryzin, G. (2004). Revenue management under a general discrete choice model of consumer behavior. *Management Science*, *50*(1), 15–33.

Train, K. E. (2009). *Discrete choice methods with simulation*. Cambridge university press.

van de Geer, R., & den Boer, A. V. (2022). Price optimization under the finite-mixture logit model. *Management Science*, *68*(10), 7480–7496.

Violin, A. (2016). *Mathematical programming approaches to pricing problems*. Springer.

Wolsey, L. A. (2020). *Integer programming*. John Wiley & Sons.

Zhang, D., & Lu, Z. (2013). Assessing the value of dynamic pricing in network revenue management. *INFORMS Journal on Computing*, *25*(1), 102–115.

Zhang, H., Rusmevichientong, P., & Topaloglu, H. (2018). Multiproduct pricing under the generalized extreme value models with homogeneous price sensitivity parameters. *Operations Research*, *66*(6), 1559–1570.

A Discrete choice model in Ibeas et al. (2014)

In this section we present the discrete choice model specification of the parking choice case study taken from Ibeas et al. (2014) as well as a table with estimated coefficients. The choice set consists of three services: paid on-street parking (PSP), paid parking in an underground car park (PUP), and free on-street parking (FSP). The utilities of the three alternatives are defined as follows:

$$\begin{aligned}
U_{\mathsf{PSP},n} = {} & \mathsf{ASC}_{\mathsf{PSP}} + \beta_{\mathsf{TD}} \cdot \mathsf{TD}_{\mathsf{PSP}} + \gamma_{\mathsf{AT}} \cdot \mathsf{AT}_{\mathsf{PSP}} + \gamma_{\mathsf{FEE}} \cdot \mathsf{FEE}_{\mathsf{PSP}} \\
& + \beta^{\mathsf{PSP}}_{\mathsf{FEE,LI}} \cdot \mathsf{LI}_n \cdot \mathsf{FEE}_{\mathsf{PSP}} + \beta^{\mathsf{PSP}}_{\mathsf{FEE,RES}} \cdot \mathsf{RES}_n \cdot \mathsf{FEE}_{\mathsf{PSP}} + \varepsilon_{\mathsf{PSP},n},
\end{aligned}$$

$$\begin{aligned}
U_{\mathsf{PUP},n} = {} & \mathsf{ASC}_{\mathsf{PUP}} + \beta_{\mathsf{AGEveh3}} \cdot \mathsf{AGEveh3}_n + \beta_{\mathsf{TD}} \cdot \mathsf{TD}_{\mathsf{PUP}} + \gamma_{\mathsf{AT}} \cdot \mathsf{AT}_{\mathsf{PUP}} \\
& + \gamma_{\mathsf{FEE}} \cdot \mathsf{FEE}_{\mathsf{PUP}} + \beta^{\mathsf{PUP}}_{\mathsf{FEE,LI}} \cdot \mathsf{LI}_n \cdot \mathsf{FEE}_{\mathsf{PUP}} + \beta^{\mathsf{PUP}}_{\mathsf{FEE,RES}} \cdot \mathsf{RES}_n \cdot \mathsf{FEE}_{\mathsf{PUP}} \\
& + \varepsilon_{\mathsf{PUP},n},
\end{aligned}$$

$$U_{\mathsf{FSP},n} = \beta_{\mathsf{TD}} \cdot \mathsf{TD}_{\mathsf{FSP}} + \beta_{\mathsf{OINT}} \cdot \mathsf{OINT}_n + \gamma_{\mathsf{AT}} \cdot \mathsf{AT}_{\mathsf{FSP}} + \varepsilon_{\mathsf{FSP},n},$$

where $\varepsilon_{\mathsf{PSP},n}, \varepsilon_{\mathsf{PUP},n}, \varepsilon_{\mathsf{FSP},n} \overset{\text{iid}}{\sim} \mathrm{Gumbel}(0, 1)$, and γ_{AT} and γ_{FEE} are normally distributed random variables, making the choice model a mixed logit (ML), while the coefficients β are constants.

The explanatory variables include the following socioeconomic characteristics: trip origin (if outside town, it affects the utility of free street parking), age of the vehicle (if less than three years old, it affects the utility of paid underground parking), the income of the driver (if low, it affects the utility of paid alternatives), area of residency of the driver (if in town, it affects the utility of paid alternatives). Additionally, the following attributes of the alternatives are considered: access time to destination (the time it takes from the parking space to a user's real destination), access time to parking (the time it takes a user to find an empty space in the parking area and park) and parking fee. The estimated parameters are shown in Table 10.

To extend the choice set to more than two prices, we duplicate the existing PSP and PUP alternatives but add one more minute of travel time to the desired destination per duplicate. This corresponds to increasing the size of the parking facilities. The attributes of the extended PSP and PUP alternatives are defined as:

$$\begin{aligned}
x_{\mathrm{PSP}_i n k_{\mathrm{AT}}} &:= x_{\mathrm{PSP} n k_{\mathrm{AT}}} + (i-1) \qquad \forall i \in \{1, \dots, J_{\mathrm{PSP}}\} \\
x_{\mathrm{PUP}_i n k_{\mathrm{AT}}} &:= x_{\mathrm{PUP} n k_{\mathrm{AT}}} + (i-1) \qquad \forall i \in \{1, \dots, J_{\mathrm{PUP}}\}
\end{aligned}$$

where k_{AT} defines the index of the arrival time attribute, which is encoded in minutes. The rest of the attributes remain untouched, i.e.:

$$\begin{aligned}
x_{\mathrm{PSP}_i n k} &:= x_{\mathrm{PSP} n k} \qquad \forall i \in \{1, \dots, J_{\mathrm{PSP}}\} \qquad \forall k \neq k_{\mathrm{AT}} \\
x_{\mathrm{PUP}_i n k} &:= x_{\mathrm{PUP} n k} \qquad \forall i \in \{1, \dots, J_{\mathrm{PUP}}\} \qquad \forall k \neq k_{\mathrm{AT}}
\end{aligned}$$

Table 10 – Utility parameters reported in Ibeas et al. (2014)

Parameter	Value
ASC_{FSP}	0.0
ASC_{PSP}	32.0
ASC_{PUP}	34.0
Fee (€) [γ_{FEE}]	$\sim \mathcal{N}(-32.328, 14.168)$
Fee PSP - low income (€) [$\beta^{\text{PSP}}_{\text{FEE,LI}}$]	-10.995
Fee PUP - low income (€) [$\beta^{\text{PUP}}_{\text{FEE,LI}}$]	-13.729
Fee PSP - resident (€) [$\beta^{\text{PSP}}_{\text{FEE,RES}}$]	-11.440
Fee PUP - resident (€) [$\beta^{\text{PUP}}_{\text{FEE,RES}}$]	-10.668
Access time to parking (min) [γ_{AT}]	$\sim \mathcal{N}(-0.788, 1.06)$
Access time to destination (min) [β_{TD}]	-0.612
Age of vehicle (1/0) [β_{AGEveh3}]	4.037
Origin (1/0) [β_{OINT}]	-5.762

B Implementation with GUROBI

In this section, we describe relevant details when implementing the Branch and Benders algorithm using the mathematical solver GUROBI.

B.1 Efficiently adding Benders cuts

In each iteration of the Benders decomposition algorithm, we are adding NR cuts to the master problem, where N is the number of customers and R the number of draws in the Monte Carlo simulation procedure. GUROBI's latest version 10 introduced a concise matrix notation, allowing us to add many constraints by only accessing the model once, thus minimizing the computational expense. In our coding example, we are using GUROBI with the `gurobipy` python package.
All variables in the master problem (see Formulation 8) are defined as matrix variables:

```python
import gurobipy as gp
m = gp.Model()
p_vars = m.addMVar(shape=J, lb=-inf, ub=inf)
obj_nr_var = m.addMVar(shape=N * R, lb=-inf, ub=0)
obj_var = m.addMVar(shape=1, lb=-inf, ub=0)
```

We then add the relevant constraints:

```python
m.addConstr(p_vars[i] >= lowerbound[i])
```

```
m.addConstr(p_vars[i] <= upperbound[i])
m.addConstr(obj_var == np.ones(N * R) @ obj_nr_var)
m.addConstr(obj_var >= obj_lowerbound)
m.addConstr(obj_var <= 0)
```

It is important to mention that for the Benders decomposition, we transform the problem from a maximization to a minimization problem, resulting in the objective variables being bounded by 0 from above. Finally, assuming we have computed the dual variables of a subproblem in an array `phi` of shape $J \times NR$, as well as an array `profit` of shape $1 \times NR$ with the corresponding achieved profits for each customer and scenario, we can add all NR Benders cuts with one matrix constraint:

```
m.addConstr(obj_nr_var - phi @ p_vars >= profit - phi @ p)
```

We do not need to invoke callbacks for this procedure, as we solve a continuous problem relaxation at each node of the Spatial Branch and Bound method.

B.2 Hyperparameter optimization

We make use of GUROBI's automatic hyperparameter tuning tool to optimize the performance of all solution methods that invoke it. We ran the tuning tool on each formulation with an instance of three prices ($J_{PSP} = 1$, $J_{PUP} = 2$) and $N = R = 50$ with 10 random tune trials per parameter set and a tune time limit of 12 hours. The reported optimal parameters that were used in the final experiments are the following:

- MILP (Formulation 4)
  - `ScaleFlag 1`
  - `SimplexPricing 0`
  - `NormAdjust 1`
  - `Heuristics 0`
  - `MIPFocus 2`
  - `Cuts 0`
  - `AggFill 1000`
  - `Threads 1`
- QCQP and QCLP (Formulations 5 and 6)
  - `ScaleFlag 0`
  - `SimplexPricing 2`

  - `NormAdjust 0`
  - `MIPFocus 3`
  - `Cuts 0`
  - `NonConvex 2`
  - `Threads 1`

- McCormick Relaxation and Benders Subproblem (Formulations 7 and 9)
  - `ScaleFlag 1`
  - `PrePasses 2`
  - `Presolve 1`
  - `NormAdjust 1`
  - `SimplexPricing 2`
  - `AggFill 1000`
  - `PreDepRow 1`
  - `PreDual 0`
  - `NumericFocus 1`
  - `Threads 1`
- Benders Master Problem (Formulation 8)
  - `ScaleFlag 1`
  - `SimplexPricing 3`
  - `NormAdjust 1`
  - `PreDual 1`
  - `NumericFocus 1`
  - `PreDepRow 0`
  - `Presolve 1`
  - `Threads 1`

GUROBI reported being unable to improve on the baseline parameter set for the Benders subproblem (Formulation 9). This is likely due to its small size, making the effects of different hyperparameters numerically insignificant. Applying the same hyperparameters as for the McCormick relaxation led to small improvements in computational speed nonetheless, which is why we kept them for the experiments.