Fast Algorithms for Capacitated Continuous Pricing with Discrete Choice Demand Models

Tom Haering

Robin Legault

Fabian Torres

Michel Bierlaire

STRC conference paper 2024 **September 19, 2024**

STRC | **24th Swiss Transport Research Conference**
Monte Verità / Ascona, May 15-17, 2024

Fast Algorithms for Capacitated Continuous Pricing with Discrete Choice Demand Models

Tom Haering
ENAC
EPFL
tom.haering@epfl.ch

Robin Legault
ORC
MIT
legault@mit.edu

Fabian Torres
ENAC
EPFL
fabian.torres@epfl.ch

Michel Bierlaire
ENAC
EPFL
michel.bierlaire@epfl.ch

September 19, 2024

Abstract

We introduce the Breakpoint Exact Algorithm with Capacities (BEAC), based on the state-of-the-art Breakpoint Exact Algorithm (BEA) to address the choice-based pricing problem (CPP) with capacity constraints, together with the Breakpoint Heuristic Algorithm (BHA) for both uncapacitated and capacitated instances. For capacity management, an approach based on an exogenous priority queue, as well as a supplier-controlled queuing strategy to generate maximal or minimal profit for robust optimization is developed. The BHA leverages a coordinate descent method, which we propose to extend with a dynamic line search (DLS) to escape local optima. Our results show that for the capacitated case, the BEAC reports runtimes up to 20 times faster than the state-of-the-art mixed-integer linear programming (MILP) approach, while the BHA performs from 100 to 5000 times faster than the MILP. For the uncapacitated case, the BHA outspeeds the BEA as well as the current state-of-the-art Branch & Benders Decomposition (B&BD) approach by multiple orders of magnitude, especially for high-dimensional instances. The average gap in optimality between the solution reported by the BHA and the global optimum was less than 0.2%, with the extension via DLS finding the global optimum in all tested instances, albeit at a significant computational cost.

Keywords

discrete choice; pricing; capacity constraints; heuristic, robust optimization

Suggested Citation

Haering, T., Legault, R., Torres, F., & Bierlaire, M. (2024). Fast Algorithms for Capacitated Continuous Pricing with Discrete Choice Demand Models. In: *Proceedings of the 24th Swiss Transport Research Conference (STRC)*

Contents

1 Introduction

Utilizing discrete choice models (DCMs) improves pricing strategies by detailing demand through individual preferences, aiding in understanding supply-demand dynamics (Sumida *et al.*, 2021). However, incorporating DCMs introduces computational issues due to their complex choice probabilities, which in general do not assume a closed-form expression Hanson and Martin (1996).

Innovative solutions address these: Gilbert *et al.* (2014) introduced a tractable approximation for revenue-maximizing pricing under mixed logit (ML) demand in congested networks, using a two-step method involving a mixed integer program and an ascent algorithm. Following this, Li *et al.* (2019) investigated a price optimization problem under discrete ML demand, proposing a pair of concave maximization problems as bounds for the revenue function. Marandi and Lurkin (2020) presented an iterative algorithm that converges to the optimal solution through linear optimization and convex approximation.

To provide a more general framework for integrating advanced choice models into optimization problems, Paneque *et al.* (2021) proposed the use of Monte Carlo simulation to generate a deterministic problem as a mixed-integer linear program (MILP). While increasing complexity due to the exponential scaling of the MILP solve time with the number of draws, this approach guarantees convergence to globally optimal solutions for sufficiently large numbers of draws. However, its practical applicability is limited to very small-scale instances, highlighting the need for more efficient computational strategies.

Recognizing this, Paneque *et al.* (2022) introduced a heuristic method leveraging the decomposable structure of scenarios, allowing for a more efficient and scalable approach to solving larger MILP formulations. Haering *et al.* (2023) introduce the Breakpoint Exact Algorithm (BEA) together with a Spatial Branch and Benders Decomposition (B&BD) approach to tackle uncapacitated pricing problem where demand is captured by any discrete choice model. They manage to outspeed the previous approaches on uncapacitated problems by several orders of magnitude when optimizing one or two prices and significantly reduce computational time for larger prices when using the B&BD algorithm. However, especially in high-dimensional instances, the problem remains difficult to solve, and the ability to add capacity constraints is highly beneficial in practice.

In this paper, we thus first extend the BEA to be able to handle capacity constraints and introduce an efficient heuristic for both the uncapacitated and capacitated case that succeeds at finding very high-quality solutions fast.

The paper is structured as follows. Section 2 describes the choice-based pricing problem and its key properties, as well as a brief description of the state-of-the-art Breakpoint Exact Algorithm (BEA). In Section 3, we introduce the new methods presented in this paper; the Breakpoint Exact Algorithm with Capacities (BEAC), the Breakpoint Heuristic Algorithm (BHA) and its extension with a dynamic line search (DLS). Section 4 presents the computational experiments, followed by Section 5, where we conclude the paper and present its essential takeaways.

2 Problem definition

Consider a competitive market with multiple 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 gather the competitors' alternatives, as well as the option to opt-out completely in alternative 0. We consider N customers choosing one product among all offered alternatives. Each individual n may furthermore have a different set of considered alternatives, denoted by C_n where $n \in \mathcal{N} = \{1, \ldots, N\}$. We require $0 \in C_n \forall n \in \mathcal{N}$, as otherwise the problem is unbounded. 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:

$$U_{in} \quad = \quad V_{in} + \beta_p^{in} p_i + \varepsilon_{in} \qquad \forall i \in C_n,$$

where $p_0 = 0$ and V_{in} represents 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, and ε_{in} is the unobserved error term (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. The probability $P_n(i)$ that individual n chooses alternative $i \in C_n$ can now be written as follows:

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

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 $[p_i^L, p_i^U]$. In general, $P_n(i)$ does not take on a closed-form expression and is thus difficult to integrate in an optimization framework.

2.1 Problem formulation and properties

To address the lack of closed-form expressions for the probability functions, 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}:

$$\begin{aligned} U_{inr} &= V_{in} + \beta_p^{in} p_i + \varepsilon_{inr} \qquad \forall i \in \{0, \ldots, J\}, n \in \mathcal{N}, r \in \mathcal{R}, \\ &= c_{inr} + \beta_p^{in} p_i \qquad \forall i \in \{0, \ldots, J\}, n \in \mathcal{N}, r \in \mathcal{R}. \end{aligned}$$

where $\mathcal{R} = \{1, \ldots, R\}$.

2.2 Breakpoints

In the CPP, the optimal price ensures that the utility of a product matches the utility of the next cheapest alternative for at least one customer and scenario, aiming to maximize profit without unnecessary customer loss. Specifically, for an optimal price p_i, there exists a customer n and scenario r such that any increase in p_i by $\varepsilon > 0$ would lower the utility U_{inr} below that of cheaper alternatives or the opt-out option, deterring that customer and decreasing overall profit. Hence, this price acts as a "breakpoint" or "indifference point" in customer decision-making, representing the maximum price before customer interest shifts to more affordable options.

This leads us to the derivation of the Breakpoint Exact Algorithm (BEA), first introduced in Haering *et al.* (2023). As we aim to develop the Breakpoint Exact Algorithm with Capacities (BEAC) based on BEA, we provide a brief overview of the latter. The BEA initially assumes a fixed order of prices and solves the original problem by evaluating all possible price permutations separately. The algorithm iteratively fixes the price of new alternatives in ascending order. Upon introducing a new alternative to the market, it calculates the breakpoint for each customer to switch from their current choice to the new alternative, sorts these breakpoints, and solves the restricted problem for each. This

process is recursively repeated until all prices are fixed. Ultimately, by exploring the entire space of breakpoints, the algorithm ensures the solution is optimal.

For a new alternative j, and given the set $\{0, 1, \ldots, j-1\}$ 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 $\{0, 1, \ldots, j-1\}$ for customer (n, r). This price is given by:

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

where $h_{nr}^j = \max_{i \in \{0,\ldots,j-1\}} \{c_{inr} + \beta_p^{in} p_i\}$. 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. Given the partial solution $p_1, \ldots, p_{j-1}$, the set of prices to test for alternative j is thus:

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

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

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

Result: optimal solution p^* and objective value o^* for the uncapacitated CPP.

$p_j^* \leftarrow 0 \quad \forall j \in \{1, \ldots, J\}$
$o^* \leftarrow 0$
for s **in** S **do**
  $p_{s_j} \leftarrow 0 \quad \forall j \in \{1, \ldots, J\}$
  $h_{nr}^{s_1} \leftarrow c_{0nr} \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}$
  $\eta_{nr} \leftarrow 0 \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}$
  $(\hat{p}, \hat{o}) \leftarrow \texttt{enumerate}(s, p, h^{s_1}, \eta, 1)$
  if $\hat{o} > o^*$ **then**
    $p^* \leftarrow \hat{p}$;
    $o^* \leftarrow \hat{o}$;
  end
end
return (p^*, o^*)

Algorithm 1 iterates over all possible orderings of prices $p_{s_1} \leq p_{s_2} \leq \cdots \leq p_{s_J}$, $s \in S$. Each restricted problem is addressed by the recursive `enumerate` function. This function accepts as arguments the current permutation $s \in S$ of alternatives, a partially filled vector of prices p, with components $p_{s_1} \leq \cdots \leq p_{s_{j-1}}$ already set, the utility $h_{nr}^{s_j}$ and the price η_{nr} selected by each simulated customer $(n, r) \in \mathcal{N} \times \mathcal{R}$, and the depth j of the current permutation's exploration.

`enumerate` sets the price of the current alternative s_j to each value in $\bar{\mathcal{P}}_{s_j}$ and solves the restricted problem recursively, updating generated profits to avoid separate computations for each price combination.

The BEA algorithm's complexity is $O(J!(NR)^J \log(NR))$, see Haering *et al.* (2023). This reflects exponential growth with J, the primary limitation addressed by the Breakpoint Heuristic Algorithm discussed later.

3 Methodology

3.1 Breakpoint exact algorithm with capacities (BEAC)

To incorporate capacity constraints into the BEA, we adopt a slightly simplified version, sequentially exploring each valid combination of breakpoints. For each combination, we invoke a function that calculates the objective value for these prices. Due to interdependent choices potentially leading to recursive substitution, continuously updating choices and profits becomes impractical. Additionally, when adding a new product, calculating breakpoints for each simulated customer from their previous preference to the new option is insufficient. Instead, breakpoints must be computed from any possible previous product to the new one, considering that capacity limits may force customers to choose an alternative other than their most preferred. This adjustment accounts for decision breakpoints involving switches from any introduced product to the new one. Despite these changes, the process of sequentially introducing prices remains, necessitating only a modification of the recursive function `enumerate` used in Algorithm 1 to a new function named "`enumerate_cap`". Algorithm 2 shows the recursive enumeration function within the BEAC.

Algorithm 2: Recursive enumeration function within the BEAC

Function `enumerate_cap`(s, p, j)**:**

$\bar{p}_{s_j}^{nrs_i} \leftarrow \frac{U_{nr}^{s_i} - c_{s_j nr}}{\beta_p^{s_j n}} \quad \forall (n, r) \in \mathcal{N} \times \mathcal{R}, i < j \in C \cup \{0\}$

$\mathcal{N}_2 \leftarrow \{(n, r, s_i) | p_{s_j}^L < \bar{p}_{s_j}^{nrs_i} < p_{s_j}^U\}$

$\mathcal{N}_2 \leftarrow \mathcal{N}_2 \cup \{p_{s_j}^L, p_{s_j}^U \ \forall i \in C\}$

Sort the elements of $\mathcal{N}_2$ from largest to smallest

if $j \leq J - 1$ **then**

  for $\bar{p}_{s_j}^{nrs_i} \in \mathcal{N}_2$ **do**

    $p_{s_j} \leftarrow \bar{p}_{s_j}^{nrs_i}$

    $(\hat{p}, \hat{o}) \leftarrow$ `enumerate_cap`$(s, p, j + 1)$

    if $\hat{o} > o^*$ **then**

      $o^* \leftarrow \hat{o}$

      $p^* \leftarrow \hat{p}$

    end

  end

end

else

  for $\bar{p}_{s_j}^{nrs_i} \in \mathcal{N}_2$ **do**

    $p_{s_j} \leftarrow \bar{p}_{s_j}^{nrs_i}$

    $o \leftarrow$ `compute_objective_value`(p)

    if $o > o^*$ **then**

      $o^* \leftarrow o$

      $p^* \leftarrow p$

    end

  end

  return (p^*, o^*)

end

end

Directly evaluating an objective function at each breakpoint combination enhances flexibility in profit computation methods. Algorithm 3 details evaluating profit using a priority queue, where individuals are assigned the highest utility alternative with available capacity. Algorithm 4 explores maximizing profit by allowing the supplier to choose customer order. This involves a precomputation step similar to the priority queue method: sorting alternatives for each customer by descending utility, then creating and lexicographically sorting an array of these preferences by price. Upon reaching capacity, alternatives are removed from all future choice sets, prioritizing more price-sensitive customers likely to switch to higher-priced options if their first choice is unavailable. For minimal profit calculation,

`compute_obj_value_with_forced_capacities` is invoked with $max =$ `false`, sorting preference orderings in descending order, thus optimizing for worst-case profit, presenting a method for robust optimization.

Algorithm 3: Compute Objective Value with Priority Queue

Function `compute_objective_value_with_priority_queue`$(p, c, prio\_queue)$`:`

- $\varsigma \leftarrow (0)_{i \in C}$
- **for** $idx \in prio\_queue$ **do**
  - $u \leftarrow [U^i_{idx}$ for $i \in C]$
  - $a \leftarrow \text{sort}(u, \text{descending})$
  - $\varphi \leftarrow \text{false}$
  - $j \leftarrow 1$
  - **while** $j \leq C - 1$ **and** $!\varphi$ **do**
    - **if** $\varsigma_{a_j} \leq c_{a_j} - 1$ **then**
      - ς_{a_j} += 1
      - $\varphi \leftarrow \text{true}$
    - **end**
    - **else**
      - j += 1
    - **end**
  - **end**
- **end**
- $o \leftarrow \sum_{i \in C} \varsigma_i \cdot p_i$
- **return** o

end

Algorithm 4: Compute Objective Value with Capacities (profit max/min)

Function `compute_objective_value_with_capacities`$(p, c; max)$**:**

$s \leftarrow \text{sortperm}(p)$
$\varsigma \leftarrow (0)_{i \in C}$
$A \leftarrow \{\}$
for $idx \in \mathcal{N} \times \mathcal{R}$ **do**
  $u \leftarrow [U^i_{idx} \text{ for } i \in C]$
  $a \leftarrow \text{sort}(u, \text{descending})$
  $A \leftarrow A \cup \{a\}$
end
if max **then**
  $A \leftarrow \text{sort}(A, \text{ascending})$
else
  $A \leftarrow \text{sort}(A, \text{descending})$
end
while $|A| \geq 1$ **do**
  $\pi \leftarrow A_{11}$
  $A \leftarrow A \setminus \{A_1\}$
  if $\pi \geq 1$ **then**
    $\varsigma_{s_{next\_pref}} \mathrel{+}= 1$
    if $\varsigma_{s_{next\_pref}} = c_{s_{next\_pref}}$ **then**
      Remove all entries π from A
      if max **then**
        $A \leftarrow \text{sort}(A, \text{ascending})$
      else
        $A \leftarrow \text{sort}(A, \text{descending})$
      end
    end
  end
end
$o \leftarrow \sum_{i \in C} \varsigma_i \cdot p_i$
return o

end

3.2 Breakpoint heuristic algorithm (BHA)

In this section, we introduce the Breakpoint Heuristic Algorithm (BHA) which can be applied to solve both the uncapacitated and capacitated version of the CPP. The BHA can be summarized as a coordinate descent, described in the following procedure:

1. Choose a starting point for the heuristic. As any combination of prices is feasible, the simplest choice here can be to choose the middle of the price bounds, $p^* = (\frac{p_i^L + p_i^U}{2})_{i \in C}$.
2. Evaluate the objective function (based on priority-queue or max/min profit) for price p, giving objective value o^*.
3. Set $j = 1$.
4. Solve the problem using the BEA (or the BEAC in the case of capacity constraints) but with modified bounds $\bar{p}^L, \bar{p}^U$, where, $\bar{p}_i^L = \bar{p}_i^U = p_i \; \forall i \neq j$ and $\bar{p}_j^L = p_j^L$, $\bar{p}_j^U = p_j^U$. Thus, all bounds except for alternative j are tight, drastically simplifying the problem. We thus iterate over all relevant breakpoints for all simulated customers, evaluating the objective value at each combination of breakpoints and updating the highest objective o^* and the best prices p^* whenever a better solution is found.
6. Set $j = j + 1$ and repeat from step 4. In the case of $j = D$, we reset it to $j = 1$.
7. Terminate once no change in the optimal solution is observed over D iterations.

The pseudocode for the BHA is provided in Algorithm 5.

Algorithm 5: Breakpoint Heuristic Algorithm (BHA)

Function `BHA`$(p_{start}; c, prio\_queue)$**:**

$o^* \leftarrow$ `compute_objective_value`(p_{start})
$p^* \leftarrow p_{start}$
$j \leftarrow 1$
$\sigma \leftarrow 0$
$i \leftarrow 0$
while $\sigma < D$ **do**
  $i \mathrel{+}= 1$
  $\lambda = \text{false}$
  $\hat{p}_j, \hat{o} \leftarrow$
   $\text{BEA}(\bar{p}^L(p, j, p^L), \bar{p}^U(p, j, p^U))$ **or** $\text{BEAC}(\bar{p}^L(p, j, p^L), \bar{p}^U(p, j, p^U), c, prio\_queue)$
  $p_j \leftarrow \hat{p}_j$
  if $\hat{o} > o^*$ **then**
    $o^* \leftarrow \hat{o}$
    $p^* \leftarrow p$
    $\sigma \leftarrow 0$
  else
    $\sigma \mathrel{+}= 1$
  end
  $j \mathrel{+}= 1$
  if $j > D$ **then**
    $j \leftarrow 1$
  end
end
return o^*, p^*

end

3.3 Dynamic line search heuristic (DLS)

The dynamic line search algorithm is an iterative enhancement to the BHA, aimed at escaping local optima through adaptive step size adjustments. Initiated with a set of initial prices p and an objective value o^*, the algorithm takes as additional inputs an initial step size δ, the number of steps to be taken in each direction (increase and decrease) k, a step increase factor γ, and a maximum step size $\Delta_{\max}$. Each iteration consists of the following steps:

- Exploring both increase and decrease directions for each price component $i \in C$, the algorithm tests k increment steps, each being a multiple of δ. At each step, a new candidate solution $\bar{p}$ is generated by adjusting the i-th component of p by $\pm k\delta$.
- For each candidate solution, the objective function is evaluated, and o^* is updated when a new best solution is found.
- If there was no improvement in objective value after completing the line search on all components of p, the step size δ is increased by the factor γ, enlarging the scope of the line search in the next iteration.

The algorithm continues until $\delta \geq \Delta_{\max}$. The pseudocode for the DLS can be found in Algorithm 6

Algorithm 6: Dynamic Line Search Algorithm

Function `dynamic_line_search_cap`$(p_{start}, \delta, k, \gamma, \Delta_{max}; caps, prio\_queue)$**:**

$o^* \leftarrow$ `compute_objective_value`(p_{start})
$p^* \leftarrow p_{start}$
$\varphi \leftarrow$ true
$\sigma \leftarrow 0$
while $\delta < \Delta_{max}$ **do**
  $\varphi \leftarrow$ false
  for $j \in 1 : D$ **do**
    for $d \in [-1, 1]$ **do**
      for $l \in 1 : k$ **do**
        $p^{new} \leftarrow p$
        $p_j^{new} \mathrel{+}= d \cdot l \cdot \delta$
        if $p_j^L \leq p_j^{new} \leq p_j^U$ **then**
          $o^{new}, p^{new}, \leftarrow$ `BHA`$(p_{new}; c, prio\_queue)$
          if $o^{new} > o^*$ **and** $p^{new} \in [p^L, p^U]$ **then**
            $o^* \leftarrow o^{new}$
            $p^* \leftarrow p^{new}$
            $\varphi \leftarrow$ true
            $\sigma \leftarrow 0$
          end
        end
      end
    end
  end
  if $!\varphi$ **then**
    $\sigma \mathrel{+}= 1$
    $\delta \mathrel{\cdot}= \gamma$
  end
  else
    $\sigma \leftarrow 0$
  end
  $p \leftarrow p^*$
  $o \leftarrow o^*$
end
return o^*, p^*
end

4 Results and discussion

4.1 Case study

To test the presented methodology we rely on the same case study as Paneque *et al.* (2021), Paneque *et al.* (2022), and Haering *et al.* (2023) 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 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), presenting the opt-out. We artificially add more PSP or PUP options by duplicating the respective alternative and increasing the access time from the parking space to the desired destination by one minute per duplicate. This corresponds to augmenting the parking space facilities in size.

4.2 Description of experiments

We will further refer to the different methods used as: Mixed-integer linear programming (MILP), Branch and Benders Decomposition (B&BD), Breakpoint Exact Algorithm (BEA), Breakpoint Exact Algorithm with Capacities (BEAC), BEAC with profit-maximizing (BEAC-M), BEAC with profit-minimizing / robust optimization (BEAC-R), Breakpoint Heuristic Algorithm (BHA) and Dynamic Line Search (DLS). The goal of our experiments is to answer the following questions:

1. How does the BEAC compare in runtime to the state-of-the art MILP approach of solving capacitated instances of the choice-based pricing problem (CPP) when an exogenous priority queue is set in place?
2. How do the BEAC-M and BEAC-R methods compare to the BEAC with an exogenous priority queue, both in terms of runtime and achieved revenue?
3. How do the BHA and DLS compare to the state-of-the-art B&BD approach on pricing instances without capacity constraints?
4. How do the BHA and DLS compare to the MILP and BEAC approaches on instances with capacity constraints and a priority queue in terms of runtime and achieved revenue?

To investigate these four issues we perform the tests described in Table 1, where N denotes the number of individuals considered, R the number of scenarios generated and $J = J_{PSP} + J_{PUP}$ equals the number of controlled prices.

Table 1 – Summary of Tests

	Test 1	**Test 2**	**Test 3**	**Test 4**
J	2	2, 4	4	2, 4
N	50	50	20	50
R	2, 5, 10, 25, 50, 100, 250	2, 5, 10, 25, 50, 100, 200, 250	100, 200, 300, 500, 1000	2, 5, 10, 25, 50, 100, 200, 250
Capacities	[20, 20]	[20, 20], [15, 15, 15, 15]	[∞, ∞, ∞, ∞]	[20, 20], [15, 15, 15, 15]
Methods	MILP, BEAC	BEAC, BEAC-M, BEAC-R	B&BD, BEA, BHA, DLS	MILP, BEAC, BHA, DLS

The bounds for all prices are taken directly from Paneque *et al.* (2021) and are defined to be [0.5, 0.7] for PSP alternatives and [0.65, 0.85] for PUP alternatives. For the DLS, we use the following hyperparameter inputs: $\delta = 0.005, k = 3, \gamma = 2, \Delta_{\max} = 0.05$. Both the MILP and B&BD experiments are performed using GUROBI 10.0.3 (Gurobi Optimization, LLC, 2021). All methods are run on a single thread in a computational cluster node with two 2.4 GHz Intel Xeon Platinum 8360Y processors, where we utilize 12 cores with a total of 16 GB of RAM.

4.3 Numerical results and analysis

Table 2 illustrates results from Test 1, showing that the BEAC is on average 20x faster than the MILP with equal profits for all completed instances. For unsolved instances ($R = 250$), the BEAC offers slightly better solutions. Table 3 compares BEAC, BEAC-M, and BEAC-R runtimes, noting that BEAC-M is about 1.2x slower on larger instances compared to BEAC, while BEAC-R finishes in approximately 1.25x the time of BEAC. Profit-wise, BEAC-M generates 5.5% more than BEAC with a random queue, and BEAC-R's robust profits are 0.1% lower than BEAC. Table 4 from Test 3 reveals that BHA and DLS significantly outperform B&BD and BEA in uncapacitated instances, with speedups

Table 2 – Test 1: MILP vs. BEAC in the capacitated case

			MILP		BEAC	
N	R	J	Time (s)	Profit	Time (s)	Profit
50	2	2	4.17	27.61	0.43	27.61
50	5	2	46.95	26.51	1.72	26.51
50	10	2	180.85	27.06	11.42	27.06
50	25	2	3119.66	27.08	169.08	27.08
50	50	2	>5 hours	≥25.15	1272.68	26.85
50	100	2	>25 hours	≥25.11	9928.57	26.85
50	250	2	>45 hours	≥23.45	>45 hours	≥25.00

Table 3 – Test 2: Priority queue vs. Max profit vs. Robust Optimization

			BEAC		BEAC-M		BEAC-R	
N	R	J	Time (s)	Profit	Time (s)	Profit	Time (s)	Profit
50	2	2	0.43	27.61	0.44	28.81	0.45	27.61
50	5	2	1.72	26.51	1.78	28.44	1.82	26.46
50	10	2	11.42	27.06	12.88	28.3	12.98	27.01
50	25	2	169.08	27.08	197.23	28.58	189.28	27.06
50	50	2	1272.68	26.85	1513.44	28.61	1523.89	26.85
50	100	2	9928.57	26.85	12093.8	28.57	12494.13	26.85
50	250	2	>45 hours	≥25.00	>45 hours	≥26.63	>45 hours	≥24.34

to $3 \cdot 10^6$. DLS, slower than BHA, does not improve objective values here. BHA and DLS achieve the global optimum for verifiable instances ($R = 100, 200$) and outperform the exact methods in unterminated cases. Table 5 indicates BHA is up to 200x faster than BEAC for two prices, and DLS is about 3x faster. For four prices, BHA and DLS achieve speed-ups to 5000x and 65x, respectively. DLS consistently finds global optima where verifiable but is 40x slower than BHA, whose solutions are within 0.2% of the optimal value. For instances with four prices and $R \geq 50$, BHA matches DLS's solutions.

Table 4 – Test 3: BHA and DLS vs. B&BD and BEA in the uncapacitated case

			B&BD		BEA		BHA		DLS	
N	R	J	Time (s)	Profit	Time (s)	Profit	Time (s)	Profit	Time (s)	Profit
20	100	4	12478	10.40	>24 hours	≥9.81	0.00	10.40	0.14	10.40
20	200	4	29213	10.40	>24 hours	≥10.40	0.01	10.40	0.41	10.40
20	300	4	>24 hours	≥10.38	>24 hours	≥10.13	0.02	10.24	0.64	10.24
20	400	4	>24 hours	≥9.81	>24 hours	≥9.42	0.05	10.26	0.78	10.26
20	500	4	>24 hours	≥10.01	>24 hours	≥9.67	0.13	10.24	1.37	10.24

Table 5 – Test 4: BHA and DLS vs. MILP and BEAC in the capacitated case

			MILP		BEAC		BHA		DLS	
N	R	J	Time (s)	Profit	Time (s)	Profit	Time (s)	Profit	Time (s)	Profit
50	2	2	4.17	27.61	0.43	27.61	0.22	27.61	1.03	27.61
50	5	2	46.95	26.51	1.72	26.51	0.32	26.46	5.91	26.51
50	10	2	180.85	27.06	11.42	27.06	0.58	27.05	20.34	27.06
50	25	2	3119.66	27.08	169.08	27.08	3.40	27.05	129.66	27.08
50	50	2	>5 hours	≥25.15	1272.68	26.85	8.31	26.53	559.04	26.85
50	100	2	>25 hours	≥25.11	9928.57	26.85	51.77	26.72	2791.28	26.85
50	250	2	>45 hours	≥23.45	>45 hours	≥25.00	455.37	26.66	15867.67	26.71
50	10	4	>10 hours	≥22.21	>10 hours	≥25.41	7.08	26.78	527.34	26.83
50	50	4	>20 hours	≥22.19	>20 hours	≥27.00	166.21	27.00	7234.88	27.00
50	100	4	>45 hours	≥20.50	>45 hours	≥24.86	866.97	26.67	34050.57	26.67
50	200	4	>72 hours	≥20.32	>72 hours	≥24.79	2762.39	26.70	106286.13	26.70

5 Conclusions

This research introduces the Breakpoint Exact Algorithm with Capacities (BEAC) and the Breakpoint Heuristic Algorithm (BHA), both of which offer substantial advancements in solving the choice-based pricing problem (CPP) with and without capacity constraints. The BEAC, enhancing the Breakpoint Exact Algorithm (BEA) with a capacity management strategy, outperforms the state-of-the-art mixed-integer linear programming (MILP) approach by 20 times in computational speed. The BHA, employing a coordinate descent method, excels in high-dimensional scenarios, showing remarkable efficiency in both capacitated and uncapacitated cases. Notably, it outpaces the MILP by a factor of 100 to 5000 for the capacitated case, and the state-of-the-art Branch and Benders Decomposition approach by several orders of magnitude for the uncapacitated case, while maintaining an

average optimality gap of less than 0.2%. The dynamic line search extension of the BHA succeeds in identifying the global optimum in all tested instances, albeit with a significant speed reduction. For future research, other extensions of the BHA to escape local optima should be considered.

6 References

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

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

Haering, T., R. Legault, F. Torres, I. Ljubic and M. Bierlaire (2023) Exact algorithms for continuous pricing with advanced discrete choice demand models, *Technical Report*, **TRANSP-OR 231211**, Transport and Mobility Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

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

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

Li, H., S. Webster, N. Mason and K. Kempf (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.

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

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

Paneque, M. P., B. Gendron, S. S. Azadeh and M. Bierlaire (2022) A lagrangian decomposition scheme for choice-based optimization, *Computers & Operations Research*, **148**, 105985.

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