Electronic Companion
"A Continuous-Knapsack-Based Framework for Constrained Static Assortment Optimization"

Appendix A: Pseudocode for the Algorithms

For completeness, this section provides implementation-level pseudocode for the algorithms.

Algorithm 1 MNL-Threshold-Based Constrained Assortment Optimization (`MNL-THR`)

Input: Downward-closed family $\mathcal{L}$, $\epsilon \in (0,1)$ and an α-approximation algorithm $\mathcal{A}_{\mathcal{L}}$

Output: Approximate optimal assortment $\hat{S}$.

1: Initialize candidate solution set $\mathcal{S}_{\text{cand}} \leftarrow \emptyset$. Define grid size $M_\epsilon \leftarrow \lceil \log_{1+\epsilon}(1/\epsilon) \rceil$.
2: **for** $t = 1$ **to** M_ϵ **do**
3: Set guess $\psi_t \leftarrow (1+\epsilon)^{t-1}\epsilon$. Let $S_t \leftarrow \mathcal{A}_{\mathcal{L}}\left(\tilde{\mathscr{R}}^{\text{MNL}}(\cdot, \psi_t)\right)$.
4: Add S_t to $\mathcal{S}_{\text{cand}}$.
5: **end for**
6: **return** $\hat{S} = \arg\max_{S \in \mathcal{S}_{\text{cand}}} \mathscr{R}^{\text{MNL}}(S)$.

Algorithm 2 CAP-Threshold Constrained Assortment Optimization (`CAP-THR`)

Input: Downward-closed family $\mathcal{L}$, $\epsilon \in (0,1)$ and an α-approximation algorithm $\mathcal{A}_{\mathcal{L}}$

Output: Approximate optimal assortment $\hat{S}$.

1: $\mathcal{S}_{\text{cand}} \leftarrow \emptyset$. $M_\epsilon \leftarrow \left\lceil \log_{1+\epsilon}(1/\epsilon) \right\rceil$. $\Psi \leftarrow \left\{ \epsilon(1+\epsilon)^{t-1} : t = 1, \ldots, M_\epsilon \right\}$.
2: **for** each combination $\boldsymbol{\psi} = (\psi_1, \ldots, \psi_m) \in \Psi^m$ **do**
3: $S_{\boldsymbol{\psi}} \leftarrow \mathcal{A}_{\mathcal{L}}(F_{\boldsymbol{\psi}})$ where $F_{\boldsymbol{\psi}}(S) = \sum_{j=1}^{m} \rho_j \tilde{\mathscr{R}}_j(S, \psi_j)$
4: $\mathcal{S}_{\text{cand}} \leftarrow \mathcal{S}_{\text{cand}} \cup \{S_{\boldsymbol{\psi}}\}$
5: **end for**
6: **return** $\hat{S} = \arg\max_{S \in \mathcal{S}_{\text{cand}}} \sum_{j=1}^{m} \rho_j \mathscr{R}_j(S)$

Algorithm 3 Joint Pricing Threshold-Based Optimization (`JOINT-THR`)

Input: Downward-closed family $\mathcal{L}$, $\epsilon \in (0,1)$ and an α-approximation algorithm $\mathcal{A}_{\mathcal{L}}$

Output: Approximate optimal assortment $\hat{S}$ and pricing $\hat{\boldsymbol{r}}$.

1: Initialize candidate solution set $\mathcal{S}_{\text{cand}} \leftarrow \emptyset$. Define grid size $M_\epsilon \leftarrow \lceil \log_{1+\epsilon}(1/\epsilon) \rceil$.
2: **for** $t = 1$ **to** M_ϵ **do**
3: Set guess $\psi_t \leftarrow (1+\epsilon)^{t-1}\epsilon$.
4: Define set function $F_t(S) = \tilde{\mathscr{R}}^{\text{Joint}}(S, \psi_t)$ as per Eq. (13).
5: Let $S_t \leftarrow \mathcal{A}_{\mathcal{L}}\left(\tilde{\mathscr{R}}^{\text{Joint}}(\cdot, \psi_t)\right)$
6: Add S_t to $\mathcal{S}_{\text{cand}}$.
7: **end for**
8: **return** $\hat{S} = \arg\max_{S \in \mathcal{S}_{\text{cand}}} \mathscr{R}^{\text{Joint}}(S)$ and corresponding optimal prices.

Appendix B: Computational Study

We conduct a computational study to complement the worst-case approximation analysis and to evaluate the practical quality–runtime trade-offs of the proposed framework. We focus on three questions. First, how close are the proposed algorithms to an exact benchmark on typical instances? Second, how do they compare empirically with the corresponding algorithms of Udwani (2025)? Third, how much solution quality is lost when the geometric search over the no-purchase probability is coarsened or replaced by the single-slice choice $\psi = 1/2$? We report results for cardinality and budget constraints, for which exact benchmarks can be computed at the tested scales.

B.1. Experimental Design

We consider instances with $n \in \{50, 100, 200, 300, 400, 500, 600\}$. For cardinality constraints, we set $K = n/10$. For budget constraints, each item cost b_i is drawn independently and uniformly from $\{1, \dots, 10\}$, and the budget is set to $B = K\bar{b}$, where $\bar{b} = n^{-1} \sum_{i=1}^{n} b_i$ is the realized mean item cost. Thus, K determines the maximum assortment size under the cardinality constraint and controls the scale of the budget constraint. For each value of n, we generate 30 independent product instances. In our baseline experiments, product revenues are uniformly distributed over $[1, 10]$, while the attraction weights satisfy $\log v_i \sim N(0, 1)$, so that v_i follows a lognormal distribution with unit log-scale dispersion.

For `MNL-THR`, we vary the geometric-grid parameter over $\epsilon \in \{0.01, 0.02, \dots, 0.10\}$. A smaller ϵ produces a denser grid and therefore requires more calls to the underlying constrained submodular maximization routine. Unless otherwise stated, the main comparisons use $\epsilon = 0.05$ as a representative intermediate choice. For cardinality constraints, we compare `MNL-THR` with `MNL-fast`, the fast approximation that fixes $\psi = 1/2$, and with Udwani-Alg1. The exact benchmark is computed using the linear-program formulation of the cardinality-constrained MNL problem. For budget constraints, both `MNL-THR` and `MNL-fast` use a modified density-greedy knapsack subroutine adapted from Algorithm 2 of DeValve et al. (2023). The budget-constrained theoretical guarantees in Section 2.5 instead rely on the subroutine of Sviridenko (2004). We compare these implementations with Udwani-Alg3, the faster Algorithm 3 of Udwani (2025). The budget-constrained $0.5 - \varepsilon$ guarantee summarized in Table 1 pertains to Algorithm 5. The exact budget benchmark is computed using the MNL mixed-integer linear formulation and solved to zero relative optimality gap.

All reported values are averages over 30 instances, and numbers in parentheses are standard errors. Runtimes are wall-clock seconds measured under the same implementation environment. Because $K = n/10$ scales proportionally with n in both settings, these experiments primarily assess scalability and the quality–runtime trade-off as the number of products grows.

B.2. Cardinality Constraints

Table EC.1 reports the cardinality results at $\epsilon = 0.05$, and Figure EC.1 plots the corresponding revenue ratios relative to the exact LP benchmark. `MNL-THR` remains consistently close to optimal: its mean benchmark ratio ranges from 0.9935 to 0.9991 across the seven tested sizes, with an overall average of 0.9963. In contrast, `MNL-fast`, which eliminates the geometric search by fixing $\psi = 1/2$, attains ratios between 0.7270 and 0.8197. Thus, on these instances, the grid search increases the benchmark ratio by approximately 17.7–27.0 percentage points relative to the single-slice method.

Table EC.1 Cardinality-constrained results. Values are mean reduced revenues with standard errors in parentheses; times are mean seconds. `MNL-THR` uses $\epsilon = 0.05$.

n	Exact Value	Exact Time	`MNL-THR` Value	`MNL-THR` Time	`MNL-fast` Value	`MNL-fast` Time	Udwani-Alg1 Value	Udwani-Alg1 Time
50	8.5342 (0.0551)	0.025	8.5034 (0.0581)	0.061	6.9920 (0.1456)	0.001	7.8229 (0.0894)	0.003
100	8.9619 (0.0386)	0.025	8.9416 (0.0412)	0.188	6.9342 (0.1921)	0.001	8.4275 (0.0752)	0.010
200	9.2721 (0.0212)	0.029	9.2635 (0.0212)	0.495	7.0343 (0.2047)	0.002	8.8666 (0.0436)	0.026
300	9.4002 (0.0135)	0.029	9.3709 (0.0118)	0.835	6.8379 (0.2188)	0.004	9.0552 (0.0281)	0.049
400	9.4646 (0.0125)	0.063	9.4238 (0.0099)	1.113	7.3159 (0.1841)	0.005	9.1924 (0.0240)	0.073
500	9.5265 (0.0105)	0.103	9.4813 (0.0098)	1.354	7.3540 (0.1679)	0.006	9.2777 (0.0274)	0.100
600	9.5727 (0.0111)	0.340	9.5102 (0.0080)	1.694	7.2534 (0.1728)	0.007	9.3411 (0.0222)	0.128

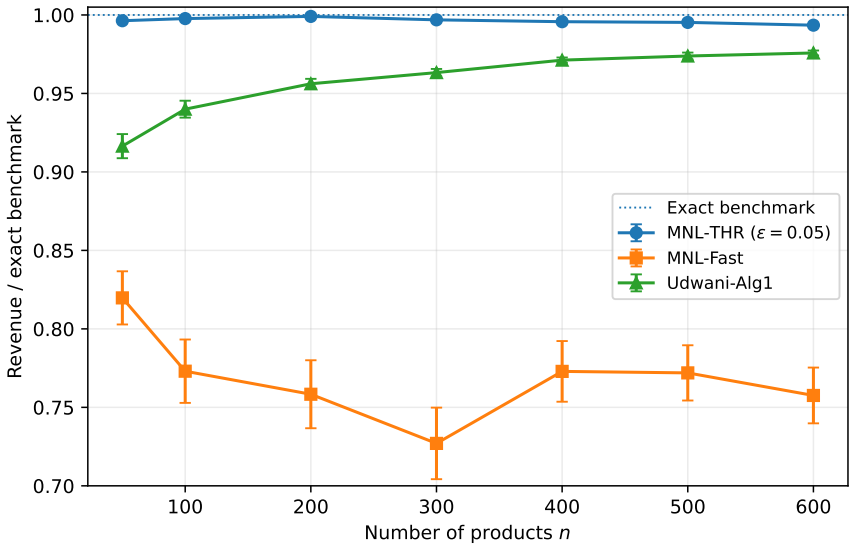


Figure EC.1 Cardinality constraints: mean revenue relative to the exact LP benchmark as a function of the number of products. Error bars indicate one standard error. `MNL-THR` uses $\epsilon = 0.05$.

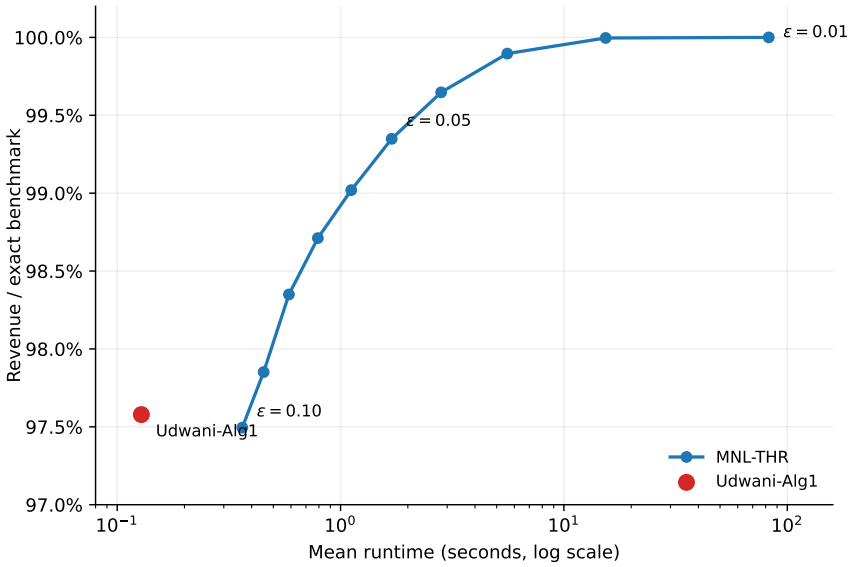


Figure EC.2 Cardinality constraints: quality–runtime frontier of `MNL-THR` at $n = 600$ as the grid parameter varies over $\epsilon \in \{0.01, 0.02, \ldots, 0.10\}$. The runtime axis is logarithmic.

MNL-THR also dominates Udwani-Alg1 in mean revenue at every tested size. At $n = 50$, the mean benchmark ratios are 0.9963 for MNL-THR and 0.9164 for Udwani-Alg1. At $n = 600$, the corresponding values are 0.9935 and 0.9758. Averaged across the seven values of n, the improvement is approximately 4.0 percentage points. The performance gap between MNL-THR and Udwani-Alg1 becomes smaller as n increases, but MNL-THR has higher revenue throughout the experiment.

The main computational cost of MNL-THR arises from the geometric search over the no-purchase probability. Figure EC.2 illustrates the resulting quality–runtime trade-off for $n = 600$ by plotting mean revenue relative to the exact benchmark against mean runtime. The ten points along the MNL-THR curve correspond to $\epsilon \in \{0.01, 0.02, \ldots, 0.10\}$, while Udwani-Alg1 is represented by a single point because it does not use this grid parameter. As ϵ increases from 0.01 to 0.05, the mean runtime of MNL-THR decreases from 82.6 seconds to 1.69 seconds, a reduction by a factor of approximately 49, while the benchmark ratio declines only from 1.0000 to 0.9935. At $\epsilon = 0.10$, MNL-THR requires only 0.363 seconds and achieves a benchmark ratio of 0.9749. For comparison, Udwani-Alg1 runs in 0.128 seconds and achieves a ratio of 0.9758. Thus, at the coarsest grid, MNL-THR and Udwani-Alg1 exhibit similar solution quality, with Udwani-Alg1 being faster; using a moderately denser grid, however, allows MNL-THR to obtain substantially higher revenue while retaining a relatively modest runtime. In particular, $\epsilon = 0.05$ achieves 0.9935 of the exact benchmark, illustrating how the grid parameter can be used to trade additional computation for improved solution quality.

B.3. Budget Constraints

Table EC.2 summarizes the budget results for the scalable modified-greedy implementation, while Figure EC.3 reports the corresponding benchmark ratios for the three principal comparison methods. At $\epsilon = 0.05$, MNL-THR attains between 0.9939 and 0.9990 of the exact MILP benchmark, with an overall average of 0.9964. In contrast, MNL-fast attains ratios between 0.8410 and 0.9121. The geometric search therefore improves the benchmark ratio by roughly 8.2–15.8 percentage points relative to the single-slice implementation.

MNL-THR also obtains higher mean revenue than Udwani-Alg3 at every tested size. The average improvement in benchmark ratio is approximately 3.18 percentage points, ranging from 5.38 points at $n = 50$ to 1.54 points at $n = 600$. Udwani-Alg3 is faster, so these methods occupy different points on the empirical quality–runtime frontier.

Figure EC.4 shows a similar quality–runtime trade-off under the budget constraint. For $n = 600$, the MNL-THR curve corresponds to the ten grid choices $\epsilon \in \{0.01, 0.02, \ldots, 0.10\}$, while Udwani-Alg3 appears as a single comparison point because it does not depend on this grid parameter. With the densest grid, $\epsilon = 0.01$, MNL-THR attains the exact benchmark to the reported precision but requires 106.2 seconds on average. Increasing ϵ to 0.05 reduces the mean runtime to 2.41 seconds, a factor of approximately 44, while maintaining a benchmark ratio of 0.9939. At $\epsilon = 0.10$, MNL-THR requires only 0.550 seconds and achieves a ratio of 0.9775. For comparison, Udwani-Alg3 requires 0.598 seconds and achieves a ratio of 0.9785, indicating that the coarsest MNL-THR grid and Udwani-Alg3 have very similar empirical quality and runtime at this problem size. A moderately denser grid, however, enables MNL-THR to achieve substantially higher revenue at the cost of additional computation. This pattern is consistent across problem sizes: averaging over all seven values of n, the benchmark ratios of MNL-THR are 0.9990, 0.9964, and 0.9854 for $\epsilon = 0.01$, 0.05, and 0.10, respectively.

Table EC.2 Budget-constrained results. Values are mean reduced revenues with standard errors in parentheses; times are mean seconds. `MNL-THR` uses $\epsilon = 0.05$.

n	Exact Value	Exact Time	`MNL-THR` Value	`MNL-THR` Time	`MNL-fast` Value	`MNL-fast` Time	Udwani-Alg3 Value	Udwani-Alg3 Time
50	8.5226 (0.0542)	0.203	8.4748 (0.0579)	0.136	7.7752 (0.1246)	0.001	8.0171 (0.0819)	0.009
100	8.9597 (0.0384)	0.307	8.9486 (0.0410)	0.324	7.5332 (0.1637)	0.002	8.4775 (0.0650)	0.022
200	9.2721 (0.0212)	0.234	9.2632 (0.0212)	0.805	8.0077 (0.1391)	0.004	8.9289 (0.0485)	0.057
300	9.4002 (0.0135)	0.463	9.3753 (0.0118)	1.158	8.1139 (0.1407)	0.006	9.1268 (0.0213)	0.093
400	9.4646 (0.0125)	0.745	9.4293 (0.0112)	1.664	8.3814 (0.1183)	0.009	9.2199 (0.0227)	0.135
500	9.5265 (0.0105)	1.024	9.4839 (0.0106)	1.988	8.4465 (0.1146)	0.011	9.3366 (0.0200)	0.180
600	9.5727 (0.0111)	1.556	9.5138 (0.0088)	2.408	8.2967 (0.1234)	0.012	9.3667 (0.0197)	0.598

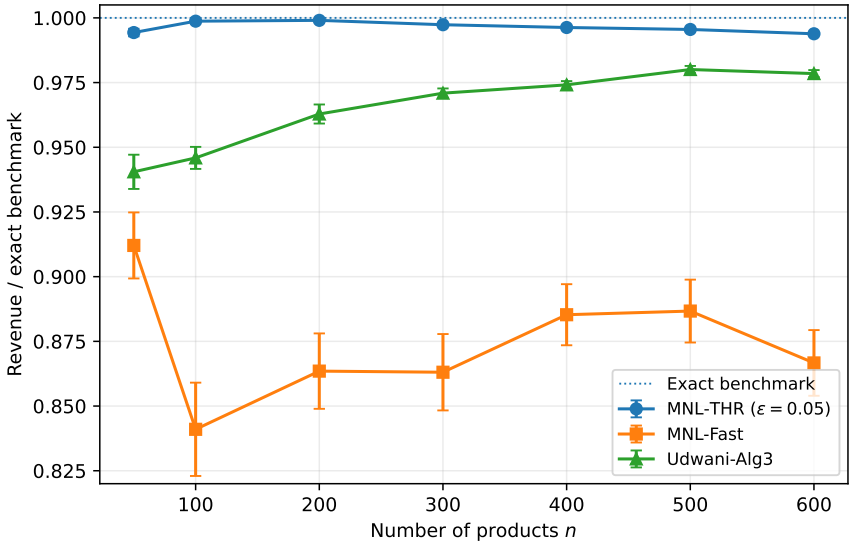


Figure EC.3 Budget constraints: mean revenue relative to the exact MILP benchmark as a function of the number of products. Error bars indicate one standard error. `MNL-THR` uses $\epsilon = 0.05$.

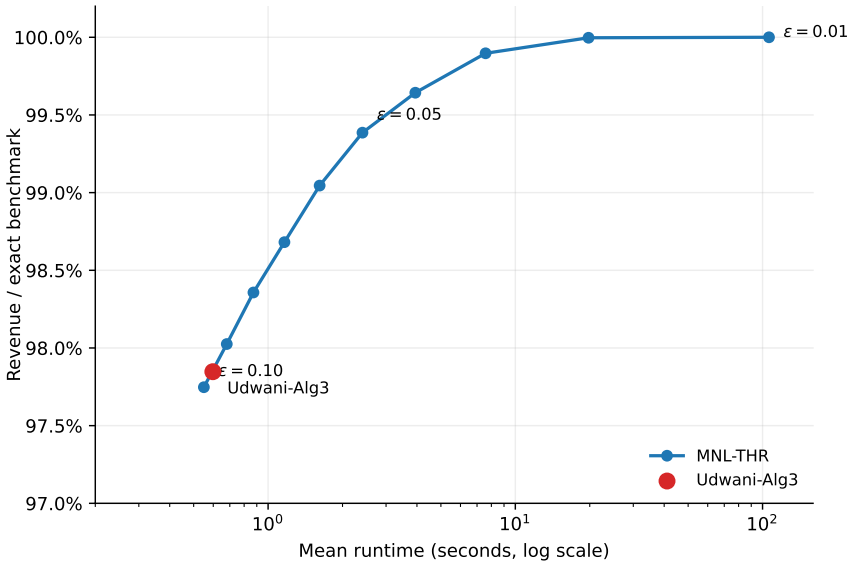


Figure EC.4 Budget constraints: quality–runtime frontier of `MNL-THR` at $n = 600$ as the grid parameter varies over $\epsilon \in \{0.01, 0.02, \ldots, 0.10\}$. The runtime axis is logarithmic.

B.4. Discussion

The computational results yield three main observations. First, the grid-search implementations achieve revenues close to the exact benchmark on the tested instances. With $\epsilon = 0.05$, both `MNL-THR` under cardinality constraints and `MNL-THR` under budget constraints achieve more than 99.3% of the exact benchmark at every tested problem size. Second, the geometric search generates a meaningful improvement in solution quality relative to the single-slice choice $\psi = 1/2$, but at a higher computational cost. This directly quantifies the quality–runtime trade-off underlying the `MNL-THR` and `MNL-fast` constructions. Third, the grid parameter ϵ provides a transparent tuning mechanism: in the largest tested instances, moving from $\epsilon = 0.01$ to $\epsilon = 0.05$ reduces runtime by roughly a factor of 44–49 while retaining more than 99.3% of the exact benchmark.

These experiments therefore support the practical interpretation of the theoretical framework. The dense-grid version is attractive when solution quality is the primary objective, whereas the single-slice version is useful when very low latency is required. Intermediate grid resolutions provide a flexible compromise between these two extremes. The comparisons with Udwani's algorithms also show higher mean revenues for the grid-search implementations on the tested instances, with the magnitude of the improvement depending on the problem size.