1 Problem Formulation: Bundle Optimization in a Rental Marketplace

We consider a rental marketplace in which a user requests a bundle of rental items for a given time interval, location, and maximum budget. Unlike a standard marketplace search, the goal is not to return individual products independently. Instead, the system constructs a complete bundle by selecting one suitable item for each requested requirement.

2 Input Definition

Let:

$$S = \{1, 2, \ldots, n\}$$

be the set of required item slots.

For each slot $s \in S$, let:

$$I_s = \{i_1, i_2, \ldots, i_k\}$$

be the set of candidate rental items that can satisfy requirement s.

The global candidate set is:

$$I = \bigcup_{s \in S} I_s$$

Each item $i \in I$ has:

- p_i – rental price
- d_i – distance from the user
- r_i – final reliability score combining lender reliability and item rating confidence
- a_i – availability score
- $L(i)$ – lender / pickup owner
- g_i – average item rating
- v_i – number of distinct users who rated the item

3 Decision Variables

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

4 Hard Constraints

4.1 Assignment Constraint

Exactly one item must be selected for each requested slot:

$$\forall s \in S : \sum_{i \in I_s} x_i = 1$$

4.2 Budget Constraint

$$\sum_{i \in I} p_i x_i \leq B$$

4.3 Availability Constraint

An unavailable item cannot be selected:

$$x_i \leq a_i$$

5 Bundle-Level Metrics

5.1 Total Price

$$P(x) = \sum_{i \in I} p_i x_i$$

The price score is budget-relative:

$$M_{\text{price}}(x) = \begin{cases} 0 & \text{if } 10\left(1 - \frac{P(x)}{B}\right) < 0 \\ 10\left(1 - \frac{P(x)}{B}\right) & 0 \leq 10\left(1 - \frac{P(x)}{B}\right) \leq 10 \\ 10 & \text{if } 10\left(1 - \frac{P(x)}{B}\right) > 10 \end{cases}$$

5.2 Distance

Distances are computed from geocoded coordinates using the Haversine formula.

Let:

$$\bar{d}(x) = \frac{1}{n} \sum_{i \in I} d_i x_i$$

$$d_{\max}(x) = \max_{i : x_i = 1} d_i$$

The distance metric combines average distance and the farthest selected item:

$$D(x) = \eta \cdot \bar{d}(x) + (1 - \eta) \cdot d_{\max}(x)$$

The normalized distance score is:

$$M_{\text{distance}}(x) = 10 \cdot e^{-D(x)/30}$$

A separate max-distance penalty is also applied:

$$Penalty_{\max d}(x) = \gamma \cdot \left(1 - e^{-d_{\max}(x)/30}\right)$$

5.3 Reliability

The reliability model is based on rating, transaction history, cancellations, complaints, response score, and verification level.

In addition to lender-level reliability, the system also considers item-level rating confidence. The influence of an item's rating is not determined only by the average rating itself, but also by the number of distinct users who rated the item.

For each item i, let:

$$g_i$$

be the average item rating, and let:

$$v_i$$

be the number of distinct users who rated item i.

The rating confidence is defined as:

$$c_i = \min\left(1, \frac{v_i}{K}\right)$$

where K is the number of ratings required for full confidence, for example $K = 30$.

The confidence-adjusted item rating is:

$$\hat{g}_i = g_i \cdot c_i + \mu \cdot (1 - c_i)$$

where μ is the marketplace prior rating, for example $\mu = 3.7$.

The item rating score is:

$$itemRatingScore_i = 2 \cdot \hat{g}_i$$

If $v_i = 0$, the item rating score is not used in the reliability calculation. Instead, the system marks the item as having insufficient rating information.

A Bayesian adjusted rating is used:

$$sampleWeight = \min\left(1, \frac{completedTransactions}{30}\right)$$

$$bayesianRating = rating \cdot sampleWeight + 3.7 \cdot (1 - sampleWeight)$$

$$ratingScore = 2 \cdot bayesianRating$$

The final item reliability score combines lender-level reliability and item-level rating confidence:

$$r_i = \begin{cases} \theta \cdot lenderReliability_i + (1-\theta) \cdot itemRatingScore_i & \text{if } v_i > 0 \\ lenderReliability_i & \text{if } v_i = 0 \end{cases}$$

where $\theta \in [0,1]$ controls the balance between lender reliability and item-specific rating. For example, $\theta = 0.7$ gives more weight to the lender's reliability while still considering the item's own rating when enough rating data exists.

New or unproven lenders receive an uncertainty penalty:

$$newPenalty = \begin{cases} 1.5 & completedTransactions < 2 \\ 1.0 & completedTransactions < 5 \\ 0 & \text{otherwise} \end{cases}$$

The lender-level reliability score is:

$$lenderReliability_i = \max(0, ratingScore - newPenalty)$$

The reliability score is normalized to:

$$r_i \in [0, 10]$$

Bundle reliability is computed as:

$$M_{\text{reliability}}(x) = \frac{1}{n} \sum_{i \in I} r_i x_i$$

5.4 Availability

Availability is first treated as a hard constraint. Items blocked, booked, under maintenance, or outside their rental-day limits are removed.

The bundle availability score is conservative:

$$M_{\text{availability}}(x) = \min_{i:x_i=1} a_i$$

5.5 Pickup Complexity

Let $Q(x)$ be the set of unique pickup points in the bundle.

$$Q(x) = \{L(i) \mid x_i = 1\}$$

Pickup complexity includes both the number of pickup points and the spatial distance between them:

$$PickupCost(x) = \frac{1}{25} \sum_{q_a, q_b \in Q(x)} dist(q_a, q_b) + \max(0, |Q(x)| - 1)$$

The pickup penalty is:

$$Penalty_{\text{pickup}}(x) = \beta \cdot PickupCost(x)$$

6 Metric Normalization

All bundle metrics are normalized to:

$$M_j(x) \in [0, 10]$$

where higher is better.

The four core metrics are:

$$M(x) = (M_{\text{price}}, M_{\text{distance}}, M_{\text{reliability}}, M_{\text{availability}})$$

7 Weighted Utility

The user's preferences are represented by weights:

$$w_1, w_2, \ldots, w_m$$

where:

$$\sum_{j=1}^{m} w_j = 1$$

The weighted utility is:

$$U(x) = \sum_{j=1}^{m} w_j M_j(x)$$

8 Personalized Semantic Objective

In the basic model, the weights w_j are fixed according to predefined profiles such as cheapest, closest, or most reliable. However, different users may have different semantic intentions.

For example, a user planning a professional event may care more about reliability, and availability, while a user renting equipment for a one-time home use may care more about price and distance.

Therefore, the objective function can be made user-dependent:

$$Score(x \mid u)$$

where u represents the user's intent, context, and preferences.

Instead of using fixed weights, the system defines:

$$w_j = w_j(u)$$

Thus, the weighted utility becomes:

$$U(x \mid u) = \sum_{j=1}^{m} w_j(u) M_j(x)$$

The user's intent is mapped into optimization parameters:

$$f(u) = (w(u), \lambda(u), \alpha(u), \beta(u), \gamma(u), \rho(u))$$

This means that not only the metric weights, but also the penalty strengths can change according to the user's goal.

For example:

- For a professional production, reliability and availability receive higher weights.
- For a low-budget user, price receives a higher weight.
- For a user who wants minimum effort, pickup complexity receives a stronger penalty.

The personalized objective is therefore:

$$Score(x \mid u) = U(x \mid u) - Penalty_{\text{total}}(x, u)$$

This changes the question from "what is the best bundle?" to "what is the best bundle for this specific user?"

9 Variance Penalty

The mean metric value is:

$$\bar{M}(x) = \frac{1}{m} \sum_{j=1}^{m} M_j(x)$$

The variance is:

$$Var(M(x)) = \frac{1}{m} \sum_{j=1}^{m} \left(M_j(x) - \bar{M}(x)\right)^2$$

The variance penalty is:

$$Penalty_{\text{variance}}(x) = \lambda \cdot Var(M(x))$$

10 Bottleneck Bonus

The bottleneck score is:

$$B_n(x) = \min_j M_j(x)$$

The bottleneck bonus is:

$$Bonus_{\text{bottleneck}}(x) = \alpha \cdot B_n(x)$$

11 Low Score Penalty

To strongly punish weak dimensions, the system adds an explicit low-score penalty.

For each metric j:

$$Penalty_j(x) = \rho_j \cdot \max(0, \tau_j - M_j(x))$$

where:

- τ_j is the minimum acceptable threshold for metric j
- ρ_j is the penalty weight for metric j

The total low-score penalty is:

$$Penalty_{\text{low}}(x) = \sum_{j=1}^{m} \rho_j \cdot \max(0, \tau_j - M_j(x))$$

Default thresholds:

$$\tau_{\text{price}} = 4, \quad \tau_{\text{distance}} = 4, \quad \tau_{\text{reliability}} = 6.5, \quad \tau_{\text{availability}} = 7$$

This means that weak lender reliability, poor availability, high distance, or poor price fit directly reduce the final score.

12 Final Objective Function

The raw final score is:

$$\begin{aligned} Score_{\text{raw}}(x \mid u) = U(x \mid u) - Penalty_{\text{variance}}(x) + Bonus_{\text{bottleneck}}(x) \quad (1) \\ - Penalty_{\text{pickup}}(x) - Penalty_{\max d}(x) - Penalty_{\text{low}}(x) \quad (2) \end{aligned}$$

The final score is bounded to the range $[0, 10]$:

$$Score(x \mid u) = \begin{cases} 0 & \text{if } Score_{\text{raw}}(x \mid u) < 0 \\ Score_{\text{raw}}(x \mid u) & 0 \leq Score_{\text{raw}}(x \mid u) \leq 10 \\ 10 & \text{if } Score_{\text{raw}}(x \mid u) > 10 \end{cases}$$

The optimization problem is:

$$\max_x Score(x \mid u)$$

subject to:

$$\forall s \in S : \sum_{i \in I_s} x_i = 1$$

$$\sum_{i \in I} p_i x_i \leq B$$

$$x_i \leq a_i$$

$$x_i \in \{0, 1\}$$

13 Scalable Candidate Retrieval and Precomputation

For large datasets, the system should not evaluate every item in real time. If each slot contains thousands of possible items, computing full scores for all candidates becomes expensive.

Therefore, the system can use an offline-online architecture.

13.1 Offline Precomputation

Before the user searches, the system can precompute and index stable item properties:

- category index
- price range index
- availability index
- geographic index using grid cells or geohashing
- reliability tiers

These indexes allow the system to quickly retrieve only relevant candidates.

13.2 Online Candidate Retrieval

At search time, instead of starting from the full candidate set I_s, the system retrieves a smaller candidate set:

$$C_s(u) = Retrieve(s, u)$$

where $C_s(u) \subseteq I_s$.

The retrieval function may use:

$$Retrieve(s, u) = IndexLookup(category_s, date, location_u, budget_u, intent_u)$$

Only the retrieved candidates are passed to pruning and beam search. This significantly reduces the effective branching factor before heuristic search begins.

13.3 Hierarchical Filtering

The filtering process can be performed in stages.

First, cheap filters are applied:

- category match
- availability
- rough price range
- geographic region

Then, only the remaining candidates receive full scoring, including reliability, exact distance, pickup complexity, and low-score penalties.

This improves scalability because expensive computations are performed only on a small subset of promising candidates.

14 Algorithmic Solution Approach

Solving the problem exactly is computationally expensive. If each requested slot has k candidates and the user requests n items, the naive search space is:

$$k^n$$

For example:

$$30^5 = 24,300,000$$

possible bundles.

Therefore, the system uses a heuristic search strategy.

14.1 Step 1: Constraint-Based Filtering

The system first removes infeasible items:

- unavailable items
- inactive listings
- items outside price constraints
- items outside distance constraints
- items violating minimum or maximum rental days

This reduces:

$$I_s \to \tilde{I}_s$$

14.2 Step 2: Candidate Pruning

For each slot s, the system keeps only the top-k candidates:

$$I'_s \subseteq \tilde{I}_s$$

$$|I'_s| \leq k$$

The preliminary score considers price, distance, reliability, and availability.

14.3 Step 3: Beam Search

Let B_t be the set of partial bundles after processing t slots.

At step $t+1$, every partial bundle is extended with candidates from the next slot. Then only the top w partial bundles are retained:

$$|B_t| \leq w$$

This reduces the practical search complexity approximately from:

$$O(k^n)$$

to:

$$O(n \cdot w \cdot k)$$

14.4 Step 4: Final Scoring

Each complete bundle is scored using:

$$Score(x \mid u) = \begin{cases} 0 & \text{if } Score_{\text{raw}}(x \mid u) < 0 \\ Score_{\text{raw}}(x \mid u) & 0 \leq Score_{\text{raw}}(x \mid u) \leq 10 \\ 10 & \text{if } Score_{\text{raw}}(x \mid u) > 10 \end{cases} \tag{3}$$

where (4)

$$Score_{\text{raw}}(x \mid u) = U(x \mid u) - Penalty_{\text{variance}}(x) + Bonus_{\text{bottleneck}}(x) \tag{5}$$

$$- Penalty_{\text{pickup}}(x) - Penalty_{\max d}(x) - Penalty_{\text{low}}(x) \tag{6}$$

14.5 Step 5: Explanation Generation

The system explains why a bundle was recommended or penalized.

Examples:

- the bundle is under budget
- the pickup route is short

- all items are available
- one lender has relatively low reliability
- one pickup point is far away
- an item has a high rating but few distinct raters, so its rating influence is reduced
- an item has enough distinct ratings, so its rating receives stronger confidence
- an item has no ratings, so the system does not use item rating and reports insufficient rating information

15 System Output

The system returns a ranked list:

$$x^{(1)}, x^{(2)}, \ldots, x^{(q)}$$

where:

$$Score(x^{(1)} \mid u) \geq Score(x^{(2)} \mid u) \geq \cdots \geq Score(x^{(q)} \mid u)$$

The top bundle is presented as the recommended bundle.

16 Conclusion

The rental bundle selection problem is formulated as a constrained multi-objective combinatorial optimization problem.

The system selects one item per requested slot while respecting budget, availability, location, and inventory constraints. It then ranks feasible bundles using a score that combines price, distance, reliability, pickup complexity, variance, bottleneck awareness, max-distance penalty, and low-score penalties.

The key contribution is that the system does not simply maximize an average score. It actively penalizes weak dimensions, low lender reliability, long pickup distances, and logistical complexity. This makes the recommended bundle not only good on average, but also balanced, practical, and robust.

Implementation Note The mathematical formulation describes the conceptual scoring model. In the actual system, additional engineering considerations such as fallback values, data availability, and performance optimizations slightly modify the exact computations.

Model vs. Implementation. The mathematical formulation above presents a clean and structured optimization model. However, the actual system implementation introduces several deviations motivated by practical engineering constraints.

First, while the model assumes continuous and well-defined metric functions $M_j(x)$, the implementation relies on discrete data, fallback values, and partial information. For example, availability is treated primarily as a hard constraint (items are filtered out) rather than a smooth scoring function, and missing data fields may be replaced with default values.

Second, some scoring components differ in their exact formulation. The distance metric in the model is expressed as a weighted combination of average and maximum distance, whereas in practice it is normalized using heuristic functions and combined with an additional explicit maximum-distance penalty. Similarly, pickup complexity is modeled abstractly as the number of unique lenders, but the implementation may incorporate spatial dispersion between pickup points.

Third, the reliability model is simplified mathematically, but the implementation uses a more detailed aggregation including Bayesian smoothing, transaction counts, verification levels, and penalties for new or unproven lenders. These additions ensure robustness against sparse or biased data.

Fourth, although the objective function is written in a compact analytical form, the system does not solve it exactly. The combinatorial nature of the problem leads to an exponential search space, so the implementation relies on heuristic search techniques (filtering, candidate pruning, and beam search). As a result, the returned solution is an approximation of the optimal bundle rather than a guaranteed global optimum.

Finally, the implementation includes additional safeguards not explicitly modeled, such as bounding the score to the range $[0, 10]$, explicit low-score penalties for weak dimensions, and defensive handling of edge cases. These adjustments improve stability, interpretability, and user experience.

Overall, the mathematical model captures the conceptual structure of the optimization problem, while the implementation adapts it to real-world data, performance constraints, and system reliability requirements.

Key Insight. The strength of the system lies not in solving the exact mathematical formulation, but in approximating it effectively under real-world constraints.