# Risk-Aware Loan Portfolio Selection Using Dynamic Programming and Branch and Bound with Partial Approval Decisions

Agatha Tatianingseto - 13524008
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: atia.seto@gmail.com, 13524008@std.stei.itb.ac.id

***Abstract*—Loan approval is not only a credit-scoring problem, but also a portfolio optimization problem because lenders must allocate limited capital while controlling risk exposure. This paper proposes a risk-aware loan portfolio selection model using Greedy, Dynamic Programming, and Branch and Bound algorithms. Each loan application is represented by expected profit, approved amount, expected loss, and customer-count weight. The system first estimates probability of default from historical Lending Club data, applies eligibility screening, computes expected loss and expected profit, and then generates approval options such as reject, approve 50%, approve 75%, and approve 100%. The optimization problem is formulated as a multi-dimensional and multiple-choice knapsack problem. Experimental analysis shows that Greedy provides a fast feasible baseline, Dynamic Programming can solve the discretized problem systematically, and Branch and Bound demonstrates the computational challenge of exact search when the option space becomes large.**

***Index Terms*—loan approval, portfolio optimization, dynamic programming, branch and bound, greedy algorithm, knapsack problem, credit risk**

## I. Introduction

Loan approval is an important decision process in banks, financial institutions, and peer-to-peer lending platforms. A lender may receive many applications, but lending capital and acceptable risk exposure are limited. Therefore, loan approval should not only estimate whether an individual borrower may default, but also determine which combination of loans should be approved as a portfolio. Approving every apparently profitable loan can still produce an undesirable portfolio when the total budget, expected loss, or risk concentration exceeds the lender's constraints.

Many credit-risk projects focus mainly on credit scoring, where borrowers are classified into good or bad credit-risk categories. Although this classification is useful, it does not fully represent the portfolio-level approval decision. In real lending, the lender must also consider loan amount, interest rate, tenor, probability of default, expected loss, and the maximum number of customers or sectors that can be accepted. For this reason, this paper treats credit scoring as an input stage and focuses on the optimization stage of loan approval.

This paper proposes a risk-aware loan portfolio selection model. Each loan application is represented as an item with expected profit as its value and several weights, such as requested loan amount, expected loss, and customer slot. The objective is to select a subset of applications, or a reduced approval amount for each application, that maximizes total expected profit under lending budget, risk budget, maximum approved-customer, and optional sector exposure constraints. This formulation follows the structure of the multi-dimensional knapsack problem, where each selected item consumes several limited capacities [1]–[3].

The proposed system consists of two layers. The first layer is an eligibility screening layer that filters applications with severe risk indicators, such as previous default, extreme debt-to-income ratio, invalid loan amount, or missing critical information. The second layer is a portfolio optimization layer that selects the best feasible approval decisions among the remaining candidates. The implementation uses the Lending Club loan dataset because it provides loan-level attributes such as loan amount, interest rate, term, annual income, debt-to-income ratio, loan purpose, and loan status [4]. The loan status is used to train a simple probability-of-default model, and the resulting probability is used to compute expected loss and expected profit. The concepts of probability of default (PD), loss given default (LGD), and exposure at default (EAD) are adopted as credit-risk components [5]. The use of default probability is also supported by previous credit-risk research showing that probability estimation is useful for risk management [6].

The main contribution of this paper is the application and comparison of three algorithmic strategies for constrained loan portfolio selection: Greedy, Dynamic Programming, and Branch and Bound. Greedy is used as a baseline, Dynamic Programming is used to obtain an optimal solution for discretized instances, and Branch and Bound is used to search the solution space while pruning infeasible or unpromising branches. This paper also considers reduced approval options, such as approving 50%, 75%, or 100% of the requested amount, which transforms the problem into a multiple-choice knapsack variant [7]. The work is intended as an educational decision-support simulation and not as a production lending system.

## II. Theoretical Basis

### A. Credit Risk and Expected Loss

Credit risk is the risk that a borrower fails to fulfill loan repayment obligations. In this paper, credit risk is represented using probability of default (PD), loss given default (LGD), and exposure at default (EAD), where PD estimates the chance of default, LGD represents the proportion of loss if default occurs, and EAD represents the exposed loan amount [5]. In a simplified loan approval simulation, EAD is approximated by the approved loan amount. The expected loss of loan application $i$ is expressed as:

$$EL_i = PD_i \times LGD_i \times EAD_i. \tag{1}$$

This value is used as a risk weight in the optimization model because a high-interest loan may still be unattractive if its expected loss is also high.

### B. Knapsack Problem

The knapsack problem is a combinatorial optimization problem where items with values and weights are selected to maximize total value without exceeding capacity. In the 0-1 knapsack problem, each item is either selected or rejected [3]. The loan approval problem can be mapped to this model by treating each loan application as an item, expected profit as value, and approved loan amount as weight. Since the lender must also consider risk budget, maximum approved customers, and optional sector exposure, the problem is better represented as a multi-dimensional knapsack problem, where each item consumes several limited capacities [1], [2].

### C. Multiple-Choice Knapsack Problem

In a basic approval model, each loan is either approved or rejected. However, this paper also considers reduced approval options, such as reject, approve 50%, approve 75%, and approve 100% of the requested amount. Since exactly one option must be selected for each borrower, the model follows the structure of the multiple-choice knapsack problem [7]. This formulation is more flexible because the optimizer may reduce exposure while still accepting profitable borrowers.

### D. Dynamic Programming

Dynamic Programming solves a problem by dividing it into stages and states, then reusing the results of overlapping subproblems. In this paper, the stage is the processed loan application or borrower, while the state contains the available budget, risk capacity, and approved-customer slot. A simplified state is:

$$dp[i][b][r][k], \tag{2}$$

where $i$ is the number of processed applications, $b$ is the budget capacity, $r$ is the risk capacity, and $k$ is the customer-count capacity. For the multiple-choice version, the transition evaluates all options of a borrower and selects the feasible option that gives the best expected profit. Budget and risk values are discretized to keep the DP table manageable.

### E. Branch and Bound

Branch and Bound explores a state-space tree where each node represents a partial set of loan approval decisions. Branching creates child nodes by choosing an approval option, while bounding estimates the best possible profit that can still be achieved from that node. A node is pruned if it violates budget, risk, customer-count, or sector constraints, or if its upper bound cannot improve the current best solution. In this paper, the upper bound is estimated by adding the best positive expected profit from each remaining borrower while ignoring future capacity constraints.

### F. Evaluation Metrics

The algorithms are evaluated using portfolio-quality and computational metrics. Portfolio-quality metrics include total expected profit, total expected loss, budget utilization, risk utilization, approved loans, reduced loans, and rejected loans. Computational metrics include runtime, Dynamic Programming memory usage, Branch and Bound generated nodes, infeasible pruned nodes, and bound-pruned nodes. These metrics are used to compare Greedy, Dynamic Programming, and Branch and Bound under the same dataset and constraints.

## III. Problem Formulation

### A. Input Data

The input of the problem is a set of loan applications:

$$A = \{1, 2, \ldots, n\}, \tag{3}$$

where each application $i \in A$ represents one borrower request. Each application contains borrower and loan attributes such as requested amount, tenor, annual interest rate, income, debt-to-income ratio, payment history proxy, previous default indicator, loan purpose or sector, operational cost, estimated probability of default, and estimated loss given default. In the implementation, these attributes are derived mainly from the Lending Club dataset [4].

The global constraints are defined as follows:

- $B$: total lending budget,
- $R$: maximum expected-loss or risk budget,
- $K$: maximum number of approved customers,
- $S_s$: maximum exposure allowed for sector $s$, if sector constraints are used.

Before optimization, each application is checked by an eligibility screening layer. An applicant may be excluded from optimization if the applicant violates hard rules, such as previous default, extremely high debt-to-income ratio, invalid loan amount, invalid tenor, or missing critical fields. This rule-based screening is not the main optimization algorithm, but it ensures that the optimizer only processes feasible candidates.

### B. Financial Variables

For each application $i$, the following variables are used:

- $a_i$: approved or requested loan amount,
- $t_i$: loan tenor in months,
- $q_i$: annual interest rate,

- $PD_i$: probability of default,
- $LGD_i$: loss given default,
- $EAD_i$: exposure at default,
- $c_i^{fund}$: funding cost,
- $c_i^{op}$: operational cost.

In this simplified formulation, exposure at default is approximated by the approved loan amount:

$$EAD_i = a_i. \tag{4}$$

The expected loss is computed using the standard credit-risk relationship:

$$EL_i = PD_i \times LGD_i \times EAD_i. \tag{5}$$

Interest income is approximated by:

$$II_i = a_i \times q_i \times \frac{t_i}{12}. \tag{6}$$

Funding cost is approximated by:

$$c_i^{fund} = a_i \times q^{fund} \times \frac{t_i}{12}, \tag{7}$$

where $q^{fund}$ is the assumed annual cost of funds.

Operational cost is modeled as:

$$c_i^{op} = c^{fixed} + q^{op} \times a_i, \tag{8}$$

where $c^{fixed}$ is a fixed processing cost and $q^{op}$ is an operational cost rate.

The expected profit of application $i$ is:

$$EP_i = (1 - PD_i) \times II_i - EL_i - c_i^{fund} - c_i^{op}. \tag{9}$$

This formulation means that a loan contributes positively only if its risk-adjusted interest income is greater than its expected loss and costs.

### C. Binary Approval Formulation

The simplest formulation uses a binary decision variable:

$$x_i = \begin{cases} 1, & \text{if loan application } i \text{ is approved,} \\ 0, & \text{if loan application } i \text{ is rejected.} \end{cases} \tag{10}$$

The objective is to maximize the total expected profit:

$$\max \sum_{i=1}^{n} x_i EP_i. \tag{11}$$

The lending budget constraint is:

$$\sum_{i=1}^{n} x_i a_i \leq B. \tag{12}$$

The risk budget constraint is:

$$\sum_{i=1}^{n} x_i EL_i \leq R. \tag{13}$$

The maximum approved-customer constraint is:

$$\sum_{i=1}^{n} x_i \leq K. \tag{14}$$

For applicants that fail eligibility screening, the decision is forced to rejection:

$$x_i = 0, \quad \forall i \in A_{ineligible}. \tag{15}$$

This binary formulation is a multi-dimensional 0-1 knapsack problem because each approved loan contributes profit but also consumes multiple capacities, such as lending budget, risk budget, and customer count [1]–[3].

### D. Reduced-Amount Approval Formulation

To make the model more realistic, this paper also considers reduced approval decisions. For each borrower $i$, define a set of options:

$$O_i = \{\text{reject}, \text{approve\_50}, \text{approve\_75}, \text{approve\_100}\}. \tag{16}$$

The decision variable becomes:

$$y_{i,o} = \begin{cases} 1, & \text{if option } o \text{ is selected for borrower } i, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

Exactly one option must be selected for each borrower:

$$\sum_{o \in O_i} y_{i,o} = 1, \quad \forall i \in A. \tag{18}$$

Each option has its own amount $a_{i,o}$, expected loss $EL_{i,o}$, expected profit $EP_{i,o}$, and approval flag $z_o$, where $z_o = 0$ for rejection and $z_o = 1$ for approval options. The objective becomes:

$$\max \sum_{i=1}^{n} \sum_{o \in O_i} y_{i,o} EP_{i,o}. \tag{19}$$

The lending budget constraint becomes:

$$\sum_{i=1}^{n} \sum_{o \in O_i} y_{i,o} a_{i,o} \leq B. \tag{20}$$

The risk budget constraint becomes:

$$\sum_{i=1}^{n} \sum_{o \in O_i} y_{i,o} EL_{i,o} \leq R. \tag{21}$$

The maximum approved-customer constraint becomes:

$$\sum_{i=1}^{n} \sum_{o \in O_i} y_{i,o} z_o \leq K. \tag{22}$$

If a sector exposure constraint is used, then for each sector $s$:

$$\sum_{i=1}^{n} \sum_{o \in O_i} y_{i,o} a_{i,o} m_{i,s} \leq S_s, \tag{23}$$

where $m_{i,s} = 1$ if borrower $i$ belongs to sector $s$, and $m_{i,s} = 0$ otherwise.

This reduced-amount formulation is a multiple-choice knapsack problem because the optimizer must choose exactly one option from each borrower's option group [7]. It allows the system to approve smaller loan amounts when full approval would violate the lending budget or risk budget.

### E. Dynamic Programming Scaling

Dynamic Programming requires discrete state dimensions. Therefore, continuous or large monetary values are scaled before being used in the DP table. For example, the budget can be divided by a budget scale $\alpha_B$, and the risk budget can be divided by a risk scale $\alpha_R$:

$$\hat{a}_{i,o} = \left\lceil \frac{a_{i,o}}{\alpha_B} \right\rceil, \quad \widehat{EL}_{i,o} = \left\lceil \frac{EL_{i,o}}{\alpha_R} \right\rceil. \tag{24}$$

The scaled capacities are:

$$\hat{B} = \left\lfloor \frac{B}{\alpha_B} \right\rfloor, \quad \hat{R} = \left\lfloor \frac{R}{\alpha_R} \right\rfloor. \tag{25}$$

This scaling reduces memory usage but introduces approximation error. A smaller scale gives more accurate results but requires a larger DP table. A larger scale improves runtime and memory usage but may reduce precision.

### F. Expected Output

The output of the optimizer is a portfolio decision containing approved loans, rejected loans, and reduced-amount approvals. The system also reports total expected profit, total expected loss, budget utilization, risk utilization, number of approved customers, and algorithm statistics. For Branch and Bound, the reported statistics include the number of explored nodes, infeasible nodes pruned, and nodes pruned by upper bound. These outputs are used later to compare Greedy, Dynamic Programming, and Branch and Bound experimentally.

## IV. Algorithm Design

This section describes the proposed solvers for risk-aware loan portfolio selection. The input applications are first screened using eligibility rules, then converted into approval options: reject, approve 50%, approve 75%, and approve 100%. The resulting portfolio is solved using Greedy, Dynamic Programming, and Branch and Bound. The problem is modeled as a multi-dimensional knapsack problem because each selected loan consumes lending budget, risk budget, and customer slot [1]–[3]. With reduced approval options, it becomes a multiple-choice knapsack problem because exactly one option must be selected for each borrower [7].

### A. Overall Pipeline

The overall pipeline is shown in Algorithm **??**. Eligibility screening removes infeasible applications, risk and profit calculation produces optimization values, and the three solvers are executed on the same option set.

**Algorithm 1 Risk-Aware Loan Portfolio Selection**

**Require:** Applications $A$, constraints $C$
**Ensure:** Solutions $(S_g, S_d, S_b)$
1: $F \leftarrow$ eligible applications from $A$
2: $I \leftarrow$ compute $EP$ and $EL$ for each $i \in F$
3: $O \leftarrow$ generate reject, 50%, 75%, and 100% options
4: $S_g \leftarrow \textsc{Greedy}(O, C)$
5: $S_d \leftarrow \textsc{DynamicProgramming}(O, C)$
6: $S_b \leftarrow \textsc{BranchAndBound}(O, C)$
7: **return** $(S_g, S_d, S_b)$

The eligibility rules used before optimization are previous severe default proxy, debt-to-income ratio above threshold, income below threshold, invalid amount, invalid tenor, or missing critical fields. Applications that fail these rules are forced into rejection and are not considered by the optimizer.

### B. Greedy Baseline

Greedy is used as a fast baseline. Each option is ranked by profit density:

$$score_{i,o} = \frac{EP_{i,o}}{a_{i,o}}, \tag{26}$$

where $EP_{i,o}$ is expected profit and $a_{i,o}$ is approved amount. The solver selects options in descending score order as long as budget, risk, customer-count, sector, and one-option-per-borrower constraints remain feasible.

**Algorithm 2 Greedy Baseline**

**Require:** Options $O$, constraints $C$
**Ensure:** Greedy solution $S$
1: Sort $O$ by descending $EP_{i,o}/a_{i,o}$
2: $S \leftarrow \emptyset$
3: **for** each option $(i, o) \in O$ **do**
4: **if** borrower $i$ has no selected option and $(i, o)$ satisfies $C$ **then**
5: add $(i, o)$ to $S$
6: **end if**
7: **end for**
8: add reject option for every unselected borrower
9: **return** $S$

Greedy is efficient but not guaranteed to be optimal because a locally attractive loan may consume capacity needed by a better global combination.

### C. Dynamic Programming

For binary approval, the DP state is:

$$dp[i][b][r][k], \tag{27}$$

where $i$ is the number of processed applications, $b$ is scaled budget capacity, $r$ is scaled risk capacity, and $k$ is the approved-customer capacity. The binary recurrence is:

$$dp[i][b][r][k] = \max \begin{cases} dp[i-1][b][r][k], \\ dp[i-1][b-\hat{a}_i][r-\widehat{EL}_i][k-1] + EP_i \end{cases} \tag{28}$$

if $b \geq \hat{a}_i$, $r \geq \widehat{EL}_i$, and $k \geq 1$.

For reduced approval, each borrower $i$ has option set $O_i$. The multiple-choice recurrence is:

$$dp[i][b][r][k] = \max_{o \in O_i} \left( dp[i-1][b-\hat{a}_{i,o}][r-\widehat{EL}_{i,o}][k-z_o] + EP_{i,o} \right), \tag{29}$$

where $z_o = 1$ for approval options and $z_o = 0$ for rejection.

**Algorithm 3 Multiple-Choice Dynamic Programming**

**Require:** Options grouped by borrower $O_i$, constraints $C$
**Ensure:** DP solution $S$

Initialize all states to $-\infty$; $dp[0][0][0][0] \leftarrow 0$
**for** $i = 1$ to $n$ **do**
  **for** each state $(b, r, k)$ **do**
    **for** each option $o \in O_i$ that fits $(b, r, k)$ **do**
      update $dp[i][b][r][k]$ and store parent choice
    **end for**
  **end for**
**end for**
reconstruct selected options from parent table
**return** $S$

DP is exact for the discretized instance because it evaluates all feasible states, but its memory grows with the number of borrowers, budget capacity, risk capacity, and customer-count limit.

### D. Branch and Bound

Branch and Bound explores a decision tree where each node represents a partial portfolio. A node stores the borrower index, selected options, used budget, used risk, approved count, current profit, and upper bound. A node is pruned if it violates constraints or if its upper bound cannot improve the current best solution. The upper bound is estimated by adding the best positive expected profit from each remaining borrower while ignoring future capacity constraints. This bound is optimistic because it may include future options that cannot all fit within the remaining budget, risk, or customer-count capacity. Therefore, it is valid for pruning because it does not underestimate the best possible continuation.

**Algorithm 4 Branch and Bound**

**Require:** Options grouped by borrower $O_i$, constraints $C$
**Ensure:** Best solution $S^*$

$bestProfit \leftarrow 0$, $S^* \leftarrow \emptyset$
push root node into priority queue $Q$
**while** $Q$ is not empty **do**
  $u \leftarrow$ node with highest upper bound
  **if** $u.upperBound \leq bestProfit$ **then**
    continue
  **end if**
  **if** $u$ is complete **then**
    update $S^*$ if $u.profit > bestProfit$
    continue
  **end if**
  **for** each option $o$ of next borrower **do**
    $v \leftarrow$ child node after selecting $o$
    **if** $v$ is feasible **then**
      compute $v.upperBound$
      **if** $v.upperBound > bestProfit$ **then**
        push $v$ into $Q$
      **end if**
    **end if**
  **end for**
**end while**
**return** $S^*$

For a node $u$ with current profit $P_u$, the upper bound is computed as:

$$UB(u) = P_u + \sum_{j \in R(u)} \max(0, \max_{o \in O_j} EP_{j,o}), \qquad (30)$$

where $R(u)$ is the set of remaining borrowers after node $u$, and $O_j$ is the option set of borrower $j$. This upper bound ignores future capacity constraints, so it may overestimate the achievable profit. However, this makes it safe for Branch and Bound pruning because a node is only discarded when even this optimistic value cannot improve the current best solution.

### E. Correctness

Dynamic Programming is optimal for the discretized instance because it enumerates all feasible state transitions and stores the best profit for each state. Branch and Bound is exact when the upper bound is valid and the search terminates after all remaining nodes are explored or pruned. Infeasible nodes and nodes whose upper bound is not better than the incumbent can be safely discarded.

## V. Implementation Plan

### A. Data Processing Pipeline

The data processing pipeline converts raw Lending Club data into standardized optimization input. The pipeline consists of the following stages.

1) **Load raw data.** The program loads Lending Club CSV files from the `data/raw/` folder.
2) **Clean and standardize data.** Interest rates are converted into decimal values, loan terms are converted into months, missing values are handled, and invalid records are removed.
3) **Build features.** The program builds features such as debt-to-income ratio, income-to-loan ratio, employment length encoding, loan purpose encoding, and credit-grade encoding.
4) **Create default target.** The loan status field is mapped into a binary target for default prediction.
5) **Train PD model.** A logistic regression or random forest model is trained to estimate $PD_i$ for each applicant.
6) **Apply eligibility rules.** Applicants that violate hard constraints are marked as ineligible before optimization.
7) **Compute expected loss and expected profit.** The program computes EAD, expected loss, interest income, funding cost, operational cost, and expected profit.
8) **Generate approval options.** Each applicant is converted into several options: reject, approve 50%, approve 75%, and approve 100%.
9) **Run optimizers.** Greedy, Dynamic Programming, and Branch and Bound are executed using the same constraints.
10) **Export results.** The program exports portfolio decisions, algorithm statistics, and experiment summaries.

### B. Intermediate Data Schema

The implementation uses a standardized intermediate file named `loan_options.csv`. Each row represents one candidate option for one borrower. The schema is shown in Table I.

Table I
INTERMEDIATE DATA SCHEMA FOR LOAN OPTIONS

| Column | Description |
|---|---|
| `customer_id` | Unique borrower identifier |
| `option_name` | reject, approve_50, approve_75, approve_100 |
| `amount` | Approved loan amount for the option |
| `expected_profit` | Risk-adjusted expected profit |
| `expected_loss` | PD × LGD × EAD |
| `approved_flag` | 1 if approved, 0 if rejected |
| `sector` | Loan purpose or sector proxy |
| `eligibility_status` | Eligible or rejected by rule |

This intermediate representation separates the financial modeling stage from the optimization stage. Therefore, all optimization algorithms receive the same standardized input.

### C. Risk and Profit Calculation

For each approval option, the program recalculates approved amount, expected loss, and expected profit. The expected loss follows:

$$EL_{i,o} = PD_i \times LGD_{i,o} \times EAD_{i,o}. \tag{31}$$

The expected profit follows:

$$EP_{i,o} = (1 - PD_i) \times II_{i,o} - EL_{i,o} - C_{i,o}^{fund} - C_{i,o}^{op}. \tag{32}$$

The reject option has zero amount, zero expected loss, and zero expected profit. Reduced approval options use 50% or 75% of the requested loan amount and recalculate all dependent financial variables.

### D. Optimizer Implementation

The Greedy solver sorts options by risk-adjusted profit ratio and selects feasible options while respecting one-option-per-borrower, budget, risk, customer-count, and sector constraints. It is expected to be the fastest solver but not always optimal.

The Dynamic Programming solver uses scaled budget and scaled risk dimensions. For small and medium experiments, parent pointers are stored to reconstruct the selected portfolio. If memory usage becomes too large, rolling arrays can be used for objective computation, while a separate choice table is used for reconstruction.

The Branch and Bound solver uses a priority queue ordered by upper bound. Each node stores the current borrower index, current profit, used budget, used risk, approved count, selected options, and upper bound. The solver records statistics such as created nodes, explored nodes, infeasible pruned nodes, bound-pruned nodes, maximum queue size, and runtime.

## VI. EXPERIMENTAL RESULTS AND COMPLEXITY ANALYSIS

This section presents the experimental evaluation of the proposed loan portfolio selection system. The evaluation focuses on portfolio quality, algorithm runtime, and theoretical complexity. Portfolio quality measures how well each algorithm selects a feasible and profitable set of loan approval decisions, while runtime and complexity analysis explain the computational trade-offs among Greedy, Dynamic Programming, and Branch and Bound.

### A. Experimental Setup

The experiment uses the processed Lending Club accepted loan data after cleaning, feature engineering, probability-of-default estimation, eligibility screening, risk-profit calculation, and approval option generation. Each eligible borrower is converted into a set of approval options. In reduced-amount approval mode, each borrower has four options: reject, approve 50%, approve 75%, and approve 100%.

The experiment was conducted on 80 eligible borrowers. Since each borrower has four possible approval options, the optimizer receives 320 generated options in total. All algorithms are executed using the same optimization constraints, consisting of total lending budget, maximum expected-loss budget, and maximum approved-customer count. The optional sector exposure constraint is not used in the main experiment, so the comparison focuses on budget, risk, and customer-count constraints. To make Dynamic Programming computationally feasible, monetary values are discretized using budget and risk scaling factors.

Table II
OPTIMIZATION CONFIGURATION

| Parameter | Value |
|---|---|
| Approval mode | Reduced |
| Number of borrowers | 80 |
| Generated options | 320 |
| Options per borrower | 4 |
| Total lending budget | 500,000 |
| Maximum risk budget | 50,000 |
| Maximum approved customers | 100 |
| Budget scale | 1,000 |
| Risk scale | 100 |
| Funding cost rate | 4% |
| Operational cost rate | 1% |

### B. Portfolio Result Analysis

The portfolio result is evaluated using total expected profit, total expected loss, approved amount, budget utilization, risk utilization, number of approved loans, number of reduced loans, and number of rejected loans. These metrics show whether an algorithm only produces a feasible solution or also uses the available budget and risk capacity effectively.

The Greedy algorithm approved 17 loans and rejected 63 borrowers. Among the approved loans, one loan was approved with a reduced amount. The Greedy solution achieved a total expected profit of 254,703.50 and used 499,350.00 of the

Table III
PORTFOLIO COMPARISON OF OPTIMIZATION ALGORITHMS

| Metric | Greedy | DP | B&B |
|---|---|---|---|
| Expected profit | 254,703.50 | N/A** | 0.00* |
| Expected loss | 37,559.83 | N/A** | 0.00* |
| Approved amount | 499,350.00 | N/A** | 0.00* |
| Budget utilization | 99.87% | N/A** | 0.00%* |
| Risk utilization | 75.12% | N/A** | 0.00%* |
| Approved loans | 17 | N/A** | 0* |
| Reduced loans | 1 | N/A** | 0* |
| Rejected loans | 63 | N/A** | 80* |

*Branch and Bound was terminated by the node limit before reaching a complete solution. Therefore, the reported portfolio is the fallback all-reject solution, not the final optimal B&B solution.
**Dynamic Programming had not completed the full 80-borrower instance, so no final DP portfolio result is reported.

Table IV
COMPUTATIONAL PERFORMANCE OF OPTIMIZATION ALGORITHMS

| Metric | Greedy | DP | B&B |
|---|---|---|---|
| Runtime (s) | 0.0526 | N/A** | 1679.86 |
| Processed borrowers | 80 | 60/80 | 80 |
| DP final states | - | N/A** | - |
| DP maximum states | - | 1,495,669 | - |
| Created nodes | - | - | 588,644 |
| Explored nodes | - | - | 200,000 |
| Complete nodes | - | - | 0 |
| Pruned infeasible nodes | - | - | 211,357 |
| Pruned by bound | - | - | 0 |
| Maximum queue size | - | - | 388,644 |
| Stopped by limit | No | Yes** | Yes |

**DP progress is reported up to borrower 60/80 because the run had not completed the full instance.

lending budget. This corresponds to 99.87% budget utilization, meaning that the Greedy algorithm almost fully consumed the available lending budget. However, the risk utilization was only 75.12%, which indicates that the lending budget became the more binding constraint in this experiment.

Dynamic Programming was executed on the same option set and reached borrower 60 out of 80 with 1,495,669 states. Since the execution had not completed the full instance, the final DP portfolio metrics are not reported. This partial result still shows the rapid growth of DP states in the multi-dimensional multiple-choice formulation.

The Branch and Bound execution did not reach a complete solution before the maximum node limit was reached. As a result, the saved portfolio corresponds to the fallback all-reject solution. Therefore, the zero expected profit, zero expected loss, and zero approved amount should not be interpreted as the optimal Branch and Bound result. Instead, this result indicates that the search process did not converge within the configured node limit.

### C. Algorithm Runtime Analysis

Runtime is used to compare the computational efficiency of the three algorithms. Greedy is expected to have the shortest runtime because it mainly performs sorting and feasibility checking. Dynamic Programming requires more time and memory because it evaluates combinations of budget, risk, and customer-count states. Branch and Bound can be efficient when many nodes are pruned, but it may become expensive when the upper bound is not tight enough.

The Greedy algorithm completed in 0.0526 seconds, confirming that it is computationally efficient for this problem size. This result is consistent with its design, since Greedy only sorts the generated options and scans them while checking feasibility constraints.

The Dynamic Programming run shows the computational cost of the multi-dimensional state representation. At borrower 60 out of 80, the number of states had reached 1,495,669. This supports the theoretical analysis that DP memory and runtime grow with the number of borrowers, scaled budget capacity, scaled risk capacity, and customer-count limit.

Branch and Bound required 1679.86 seconds, or approximately 28 minutes, before stopping at the configured node limit. During the search, it created 588,644 nodes and explored 200,000 nodes. However, no complete solution was reached. The search also pruned 211,357 infeasible nodes, but no nodes were pruned by upper bound. This indicates that the upper-bound strategy was valid but too loose for this instance, so it could not effectively remove unpromising branches.

Table V
BRANCH AND BOUND SEARCH STATISTICS

| Metric | Value |
|---|---|
| Borrowers | 80 |
| Generated options | 320 |
| Created nodes | 588,644 |
| Explored nodes | 200,000 |
| Complete nodes | 0 |
| Infeasible-pruned nodes | 211,357 |
| Bound-pruned nodes | 0 |
| Maximum queue size | 388,644 |
| Maximum node limit | 200,000 |
| Stopped by node limit | Yes |
| Runtime (s) | 1679.86 |
| Runtime (min) | 28.00 |
| Best profit found | 0.00 |

The number of explored and pruned nodes is especially important for Branch and Bound. A high number of infeasible-pruned nodes indicates that feasibility constraints were useful for removing invalid branches. However, the absence of bound-pruned nodes shows that the current upper-bound function did not reduce the search space effectively. As a result, Branch and Bound approached a large-scale exhaustive search and could not finish within the node limit.

### D. Complexity Analysis

Let $n$ be the number of borrowers, $m$ be the maximum number of options per borrower, $\hat{B}$ be the scaled budget capacity, $\hat{R}$ be the scaled risk capacity, $K$ be the maximum number of approved customers, and $V$ be the number of explored Branch and Bound nodes.

For Greedy, there are at most $nm$ generated options. The algorithm sorts all options by profit density and then scans them

while checking feasibility. Therefore, the time complexity is:

$$O(nm \log(nm)). \tag{33}$$

The space complexity is $O(nm)$ because all generated options are stored. In this experiment, $n = 80$ and $m = 4$, so Greedy sorts 320 options. This explains why Greedy can finish in less than one second.

For Dynamic Programming, the state is represented by budget, risk, and approved-customer count. For each borrower, the algorithm evaluates up to $m$ options over the feasible states. Therefore, the time complexity is:

$$O(nm\hat{B}\hat{R}K). \tag{34}$$

The space complexity is $O(\hat{B}\hat{R}K)$ if only the current DP layer is stored. However, if parent choices are stored for solution reconstruction, memory usage can increase to:

$$O(n\hat{B}\hat{R}K). \tag{35}$$

This explains why budget and risk scaling are needed in the implementation.

For Branch and Bound, each borrower can generate up to $m$ branches. In the worst case, if no node can be pruned, the algorithm may explore all possible combinations:

$$O(m^n). \tag{36}$$

In this experiment, the reduced-amount mode uses $m = 4$ options per borrower and $n = 80$ borrowers, so the worst-case number of combinations is:

$$4^{80}. \tag{37}$$

This value is extremely large, which explains why Branch and Bound did not complete within the node limit.

In practice, the explored search space is usually smaller because infeasible nodes and nodes with weak upper bounds are pruned. If $V$ is the number of explored nodes, and the upper-bound calculation scans the remaining borrowers and their options, the practical runtime can be described as approximately:

$$O(Vnm), \tag{38}$$

where $V \leq m^n$. The space complexity is $O(V)$ in the worst case because active nodes are stored in a priority queue.

Table VI
THEORETICAL COMPLEXITY COMPARISON

| Algorithm | Time Complexity | Space Complexity |
|---|---|---|
| Greedy | $O(nm \log(nm))$ | $O(nm)$ |
| Dynamic Programming | $O(nm\hat{B}\hat{R}K)$ | $O(\hat{B}\hat{R}K)$ |
| Branch and Bound | $O(m^n)$ worst case | $O(V)$ |

### E. Discussion

The three algorithms show different trade-offs. Greedy is the fastest algorithm because it only sorts options and performs feasibility checks. In this experiment, Greedy produced a feasible portfolio with 17 approved loans, 99.87% budget utilization, and 254,703.50 total expected profit. However, its decision is based on local profit density, so it may miss a better global combination.

Dynamic Programming is more reliable for finding the best discretized solution because it evaluates feasible state transitions systematically. However, the partial execution already reached 1,495,669 states at borrower 60 out of 80, showing that its memory usage grows rapidly as the scaled budget, scaled risk, and customer-count dimensions increase.

Branch and Bound provides a search-based optimization approach. In theory, Branch and Bound can find the optimal solution if the search completes and the upper bound is valid. However, the experiment shows that the reduced-amount option set creates a very large search space. With 80 borrowers and four options per borrower, the worst-case number of combinations is $4^{80}$. The implemented upper bound was valid but not tight, because it ignored future capacity constraints and added the best positive expected profit from each remaining borrower. This made the bound optimistic, but it also reduced pruning effectiveness. As a result, no node was pruned by upper bound, and the algorithm stopped before reaching a complete solution.

Reduced-amount approval gives the optimizer more flexibility than binary approval. Instead of fully rejecting a borrower when full approval is too expensive or too risky, the optimizer can approve a smaller amount. This can improve budget and risk utilization. However, reduced approval also increases the number of options per borrower, which makes the optimization problem larger. Therefore, it may improve portfolio quality while increasing computational cost.

Overall, the experiment shows that Greedy provides a fast feasible baseline, while Dynamic Programming and Branch and Bound demonstrate the computational challenge of solving a multiple-choice loan portfolio optimization problem under multiple constraints.

## VII. CONCLUSION

This paper presents a risk-aware loan portfolio selection model that combines credit-risk estimation with portfolio-level optimization. Instead of treating loan approval only as an individual classification problem, the proposed approach considers limited lending budget, expected-loss budget, maximum approved customers, and partial approval decisions. Each borrower is transformed into a set of approval options, making the problem suitable to be modeled as a multi-dimensional and multiple-choice knapsack problem.

The implementation compares Greedy, Dynamic Programming, and Branch and Bound. Greedy provides a very fast feasible solution by selecting options based on profit density, but it is not guaranteed to find the global optimum. Dynamic

Programming is more systematic because it evaluates feasible states over budget, risk, and customer-count dimensions, although it requires discretization and can consume large memory. Branch and Bound provides an exact search framework, but the experiment shows that its performance depends strongly on the branching factor, node limit, and upper-bound tightness.

Overall, the result shows that loan approval can be modeled as an optimization problem rather than only a prediction problem. Reduced-amount approval also gives the optimizer more flexibility because it can approve partial loans instead of only accepting or rejecting each borrower. Future work can improve the model by using a stronger credit-risk model, a tighter Branch and Bound upper bound, larger-scale experiments, and more realistic banking constraints such as sector concentration, fairness, and regulatory requirements.

## ACKNOWLEDGMENT

The author would like to thank the IF2211 Algorithm Strategies course team for the guidance and learning materials related to algorithm design and analysis. The author also acknowledges the availability of the Lending Club dataset and related references, which supported the development of the loan portfolio optimization simulation in this paper.

## REFERENCES

[1] Google for Developers, "Knapsack optimization," https://developers.google.com/optimization/pack/knapsack, accessed: 2026-06-16.

[2] Google OR-Tools, "knapsack_solver.h," https://github.com/google/or-tools/blob/stable/ortools/algorithms/knapsack_solver.h, accessed: 2026-06-16.

[3] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[4] Kaggle, "Lending club loan data," https://www.kaggle.com/datasets/wordsforthewise/lending-club, accessed: 2026-06-16.

[5] Basel Committee on Banking Supervision, "CRE32: IRB approach: Risk components," https://www.bis.org/basel_framework/chapter/CRE/32.htm, accessed: 2026-06-16.

[6] I.-C. Yeh and C. hui Lien, "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, vol. 36, no. 2, pp. 2473–2480, 2009.

[7] E. M. Bednarczuk, J. Miroforidis, and P. Pyzel, "A multi-criteria approach to approximate solution of multiple-choice knapsack problem," *Computational Optimization and Applications*, vol. 70, no. 3, pp. 889–910, 2018.

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2026

Agatha Tatianingseto – 13524008