Optimizing Bond Allocation for Collateral Posting Under Multiple CSAs

Smriti Tiwari

August 30, 2024

Abstract

This project aims to propose a solution, highlight the underlying assumptions, and explore possible extensions for optimizing bond allocation to meet collateral requirements under multiple CSAs.

1 Problem Statement Summary

The CSA (Credit Support Annex) collateral requirement is determined based on trades marked-to-market daily or periodically. The CSA value can be positive (in the money, receiving collateral) or negative (out of the money, posting collateral). The goal is to find the most efficient solution for distributing collateral across multiple CSAs i.e, optimize bond allocation to maximize the benefit or minimize the cost while ensuring the CSA requirements.

Collateral can be in the form of cash or bonds with different credit ratings. A cost-efficient approach to allocating collateral for posting is to use the collateral that has been received, minimizing any funding costs. This works well when the collateral required to be posted is less than the collateral received. However, if the collateral to be posted exceeds the collateral received, additional costs arise from funding the collateral requirement, leading to increase in leverage. In situations where the received collateral is insufficient to meet the posting requirements across multiple CSAs, deciding whetherand howto default requires careful consideration of several factors. The objective is to minimize the negative impact of a default while taking into account the legal, financial, and reputational consequences.

Other feature of variability is substitution, if market conditions change then party posting collateral may substitute bonds with others that are more favorable in terms of cost of availability.

For example, in a 2 party scenario:
1. Party A and Party B enter into a derivatives transaction, and the CSA requires Party A to post collateral if the value of the derivative moves against them.
2. Party A holds a portfolio of bonds and selects those that meet the CSA's criteria for eligible collateral.
3. The selected bonds are valued, haircuts are applied, and the bonds are posted as collateral to Party B.

4. If the market value of the derivatives changes, Party A may need to post additional bonds or substitute the existing bonds with others.

2 Solution

Solution for the above problem requires implementation and consideration of the following:

1. **Haircut** on eligible collateral: Bonds offered as collateral are valued according to market prices, and often "haircut" is applied. CSA under the ISDA master agreement specifies how to value bond and the respective haircuts that need to be applied.
2. **Parameter checks** for eligible collateral, minimum transfer amount, threshold.
3. **Optimization** : When optimizing bond allocation in scenarios where there are costs with each bond, you need to consider several factors to minimize the total cost while meeting collateral requirements and utilize advanced optimization tools and libraries to solve bond allocation problem efficiently.

Approaches for optimization:
(i). Minimize total cost : Objective function for this is to allocate bonds in a way that total cost of holding or posting bonds as collateral is minimized. Use of **Linear programming or Mixed Integer programming** to minimize total cost subject to value and eligibility. The problem can be formulated as an LP where objective function is total cost of the bonds, and constraints ensure the total value of bonds meet or exceeds the required collateral amount.

(ii). Cost benefit analysis : Evaluate trade-off between the cost of bonds and the benefit of using them as collateral. Approach is to use **weighted cost i.e., assign weights based on the cost-benefit ratio**. Higher weights for bonds with lower cost relative to their value. Calculate the cost-benefit ratio for each bond and prioritize bonds with a favorable ratio for collateral allocation. Allocate bonds by selecting the least costly bonds first, ensuring that the required collateral is met while minimizing the total cost.

Approach :
1. Sort Bonds: Sort the bonds based on their cost per unit of adjusted value(after applying haircuts).
2. Select Bonds: Starting from the least costly bond, add bonds to the collateral pool until the required threshold is met.
3. Minimize Total Cost: The algorithm continues to add bonds in order of increasing cost until the collateral requirement is satisfied.

(iii). **Greedy algorithm** : Another solution to optimize bond allocation when there is a cost associated with each bond is to use Greedy Algorithms. This approach is simpler and faster compared to more complex optimization techniques like Linear Programming. It iteratively selects the bond with the lowest cost per unit of collateral value until the collateral requirement is met.

(iv)Greedy algorithm for cases where the required collateral is less than collateral received and **leverage cost** needs to be computed.
Step 1: Optimize bond allocation to minimize the cost of using available bonds

while considering leverage cost.
Step 2: If the available bonds are insufficient to cover the net collateral requirement, calculate the additional leverage cost.
Step 3: Select bonds that minimize both the bond cost and leverage cost.

3 Code Structure

Class Bond : represents a bond with attributes like issuer, market value, credit rating, currency, maturity Date. Calculate the value of the bond as collateral after applying the haircut.

```
class Bond:
    def __init__(self, id, issuer, market_value, credit_rating,
                                    currency, maturity_date,
                                    haircut):
        self.id = id
        self.issuer = issuer
        self.market_value = market_value
        self.credit_rating = credit_rating
        self.currency = currency
        self.maturity_date = maturity_date
        self.haircut = haircut

    def collateral_value(self) -> float:
        """Calculate the value of the bond as collateral after
                                applying the haircut."""
        return self.market_value * (1 - self.haircut)
```

Parameter check snippet to check eligibility check :

```
eligible_ratings = ['AAA', 'AA']  # Eligible ratings for collateral
minimum_transfer_amount = 50000  # Minimum transfer amount in USD
threshold = 100000  # Threshold below which no collateral is
                                  required

def filter_eligible_bonds(bonds, eligible_ratings):
    return bonds[bonds['rating'].isin(eligible_ratings)]
```

Optimizing bond allocation function using **Linear programming** optimization:

```
def optimize_bond_allocation(bonds, haircuts, eligible_ratings,
                                  threshold):
    # Apply haircuts
    bonds['adjusted_value'] = bonds.apply(lambda row: row['value']
                                    * (1 - haircuts.get(row['
                                    rating'], 0)), axis=1)

    # Filter eligible bonds
    eligible_bonds = bonds[bonds['rating'].isin(eligible_ratings)]

    # Define the problem
    prob = LpProblem("Bond_Allocation_Optimization", LpMinimize)

    # Define decision variables
    bond_vars = LpVariable.dicts("Bond", eligible_bonds['bond_id'],
                                     cat='Binary')
    # Objective function: Minimize total cost
    prob += lpSum([bond_vars[bond_id] * eligible_bonds.loc[
                                    eligible_bonds['bond_id'] ==
                                    bond_id, 'cost'].values[0]
                                    for bond_id in eligible_bonds
                                    ['bond_id']])

    # Constraint: Ensure total adjusted value meets the threshold
    prob += lpSum([bond_vars[bond_id] * eligible_bonds.loc[
                                    eligible_bonds['bond_id'] ==
                                    bond_id, 'adjusted_value'].
                                    values[0] for bond_id in
                                    eligible_bonds['bond_id']]) >
                                    = threshold

    # Solve the problem
    prob.solve()

    # Results
    selected_bonds = [bond_id for bond_id in eligible_bonds['
                                    bond_id'] if bond_vars[
                                    bond_id].varValue == 1]
    total_cost = value(prob.objective)
    return selected_bonds, total_cost
```

Optimizing bond allocation function using **Greedy algorithm** :
Use of Cost per Value Calculation function that calculates the cost per unit of adjusted bond value (cost_per_value). Sort bonds by cost_per_value in ascending order to prioritize cheaper bonds. Iteratively add bonds with the lowest cost_per_value to the selected list until the required collateral value (threshold) is met.

```
def greedy_bond_allocation(bonds, threshold):
    # Calculate cost per unit of adjusted value
    bonds['cost_per_value'] = bonds['cost'] / bonds['adjusted_value
                                      ']

    # Sort bonds by cost per unit of adjusted value (ascending)
    bonds = bonds.sort_values(by='cost_per_value')

    # Select bonds until the threshold is met
    selected_bonds = []
    total_value = 0
    total_cost = 0

    for index, bond in bonds.iterrows():
        if total_value >= threshold:
            break
        selected_bonds.append(bond['bond_id'])
        total_value += bond['adjusted_value']
        total_cost += bond['cost']

    return selected_bonds, total_value, total_cost
```

Extension of greedy algorithm can be used when the collateral required to be posted is more than the collateral received. The leverage is roughly computed from the shortfall and leverage cost per unit. Bonds are allocated using the greedy approach as before, prioritizing bonds with the lowest cost per unit of adjusted value. After bond allocation, if there is a shortfall, the leverage cost is calculated.

```
def calculate_leverage_cost(shortfall, leverage_cost_per_unit):
    return shortfall * leverage_cost_per_unit

def greedy_bond_allocation_with_leverage(bonds, threshold,
                                         collateral_received,
                                         leverage_cost_per_unit):
    # Calculate the net collateral requirement
    net_collateral_required = max(threshold - collateral_received,
                                   0)

    # Calculate cost per unit of adjusted value
    bonds['cost_per_value'] = bonds['cost'] / bonds['adjusted_value
                                   ']

    # Sort bonds by cost per unit of adjusted value (ascending)
    bonds = bonds.sort_values(by='cost_per_value')

    # Select bonds until the net collateral requirement is met
    selected_bonds = []
    total_value = 0
    total_cost = 0

    for index, bond in bonds.iterrows():
        if total_value >= net_collateral_required:
            break
        selected_bonds.append(bond['bond_id'])
        total_value += bond['adjusted_value']
        total_cost += bond['cost']

    # If there's still a shortfall, calculate leverage cost
    shortfall = max(net_collateral_required - total_value, 0)
    leverage_cost = calculate_leverage_cost(shortfall,
                                   leverage_cost_per_unit)

    return selected_bonds, total_value, total_cost, leverage_cost
```

Sample I/P for dummy run :

```
bonds = pd.DataFrame({
    'bond_id': ['B001', 'B002', 'B003', 'B004'],
    'value': [100000, 150000, 120000, 180000],  # Market value of
                                   the bonds
    'cost': [500, 700, 600, 800],  # Cost associated with each bond
    'rating': ['AAA', 'AA', 'AAA', 'A'],  # Credit rating of the
                                   bonds
})

# CSA parameters
haircuts = {
    'AAA': 0.02,
    'AA': 0.05,
    'A': 0.10
}

eligible_ratings = ['AAA', 'AA']  # Eligible ratings for collateral
threshold = 200000  # Total CSA to be posted
collateral_received = 150000  # CSA collateral received
leverage_cost_per_unit = 0.05  # Leverage cost per unit of
                                   additional collateral needed
```

4 Assumptions

In the problem definition and the subsequent solution approach, several key assumptions are made to simplify and structure the bond allocation and leverage cost optimization problem. These assumptions are crucial to understanding the limitations and applicability of the proposed solutions. Here are the key assumptions:

1. Bond Valuation and Haircuts:
- Fixed Haircuts: It is assumed that the haircut values applied to each bond based on its credit rating are fixed and known in advance. This means the haircut percentages (e.g., 2% for AAA bonds, 5% for AA bonds) are constant and do not vary with market conditions.
- Market Value of Bonds: The market value of the bonds is assumed to be accurate and static during the allocation process, with no consideration of market fluctuations during the allocation period.

2. Collateral Eligibility:
- Eligibility Criteria: Bonds are selected based on predefined eligibility criteria (e.g., credit ratings). It is assumed that only bonds meeting these criteria can be used as collateral.
- All-or-Nothing Allocation: Bonds are either fully included in the collateral pool or not included at all. The model does not consider partial allocation of bond values.

3. Leverage Cost:
- Fixed Leverage Cost: The leverage cost per unit of additional collateral required is assumed to be constant. This assumes that the cost of borrowing or obtaining additional collateral does not fluctuate with the amount borrowed or with market conditions.
- Uniform Cost for All Collateral: It is assumed that the leverage cost is the same for all forms of collateral, whether it is cash, bonds, or other securities.

4. Collateral Requirements:
- Fixed CSA Requirement: The CSA collateral to be posted is a fixed value and does not change during the allocation process. This implies that the total amount of collateral required is known in advance and is not subject to variation.
- No Over-Collateralization: The model assumes that the goal is to meet the collateral requirement exactly, without considering the benefits or risks of over-collateralization.

5. Optimization Objective:
- Cost Minimization: The primary objective is to minimize the total cost, which includes the cost of bond allocation and any leverage cost. The model assumes that other factors, such as the potential return on collateral or strategic considerations, are secondary or irrelevant.
- Greedy Approach: The solution assumes that a greedy algorithm, which selects bonds with the lowest cost per unit of adjusted value first, is a suitable method for solving the problem. This assumes that locally optimal choices lead to a globally optimal solution, which may not always be true in more complex scenarios.

6. Simplification of Financial Instruments:
- Homogeneous Leverage Sources: It is assumed that all leverage is sourced uniformly, without distinguishing between different types of financial instruments or their associated risks (e.g., borrowing cash vs. repo agreements).
- No Transaction Costs: The model does not account for transaction costs, such as fees for purchasing or selling bonds, which could impact the overall cost.

7. Risk Considerations:
- No Credit Risk Impact: The model does not explicitly account for credit risk or the potential impact of a bonds downgrade during the allocation period.
- No Market Liquidity Constraints: It is assumed that all bonds can be liquidated or used as collateral without any liquidity constraints or market impact.

8. Static Environment:
- No Time Variation: The model assumes a static environment where all parameters (bond prices, haircuts, leverage costs, etc.) remain constant throughout the allocation process. There is no consideration of changes over time, such as interest rate changes or market volatility.

Summary:
These assumptions help simplify the problem and make it tractable for a straightforward optimization approach like the greedy algorithm or linear programming. However, in real-world scenarios, some of these assumptions may not hold, and the model might need to be adjusted to account for dynamic factors, market conditions, and more complex financial considerations. Understanding these assumptions is crucial for interpreting the results and determining the applicability of the model in practical situations.

5 Extension

The framework can be extended to test different scenarios and strategies to find the optimal bond allocation. Few of the ideas are as follows:
1. MC simulation to explore allocation strategies and access their impact on total cost and compliance.

2. Another extension could be in the leverage cost scenario:
a. Borrowing Funds: If additional cash collateral is required, calculate the cost based on the borrowing rate.
b. Using Other Assets: If other assets (e.g., securities) need to be liquidated or borrowed, include the associated transaction costs.
c. Repo or Lending Arrangements: Consider the cost of entering into repurchase agreements (repos) or securities lending to obtain the necessary collateral.

3. Investors might want to use leverage to enhance returns, but this comes with costs and risks, including margin requirements. The goal is to optimize the portfolio's return while minimizing the costs associated with leverage. The leverage cost concept from the bond allocation problem can be directly applied here. The optimization would involve selecting assets that maximize returns while considering the costs of leverage and the constraints imposed by margin requirements.

4. Capital Allocation in Corporate Finance: Companies allocate capital to different projects or investments, aiming to maximize returns while minimizing

costs (e.g., financing costs, project execution risks).The bond allocation model can be used to optimize the selection of projects or investments, considering both the costs of capital (similar to bond costs) and the potential need for additional financing (analogous to leverage).

5. Credit Risk Management in Banks: Banks need to manage credit risk by allocating capital to different loan portfolios, considering regulatory requirements, cost of capital, and expected returns. The principles of optimizing bond allocation under constraints can be applied to credit risk management, where the goal is to allocate capital in a way that minimizes risk-weighted assets (RWAs) while considering the cost of capital.

6. Multi-Asset Portfolio Allocation with Factor Models : Investors often allocate assets based on factor exposures (e.g., value, momentum, size). The goal is to construct a portfolio that optimally balances these factor exposures while minimizing costs. The bond allocation model can be adapted to optimize factor exposures, treating each factor as a constraint and minimizing costs associated with achieving the desired exposure levels.

Summary of Extension Approaches:
1. Cost Minimization: Extend the cost minimization framework to include different types of costs (e.g., transaction costs, financing costs, tax implications) in various portfolio allocation problems.
2. Leverage Management: Adapt the leverage cost concept to different financial scenarios where borrowing or additional funding is needed, considering the impact on portfolio returns and risks.
3. Constraints Handling: Incorporate various constraints (e.g., regulatory requirements, risk limits, liquidity needs) into the optimization process, similar to how eligibility criteria and haircuts were handled in the bond allocation problem.
4. Dynamic and Multi-Period Optimization: Consider extending the model to handle dynamic or multi-period scenarios, where portfolio allocation decisions are made over time, considering changing market conditions and evolving portfolio needs.

By adapting these core principles to the specific characteristics and constraints of different portfolio allocation problems, the methodology used in the bond allocation problem can serve as a versatile tool in financial optimization and risk management across various domains.

6 Conclusion

Optimizing bond allocation when costs are associated involves using various strategies, including minimizing total costs, performing cost-benefit analysis, diversifying bonds, using optimization tools, and conducting simulations. Implementing these strategies can help efficiently manage collateral while controlling costs. The provided Python code snippets illustrates how to use linear programming to address such optimization problems. The greedy algorithm approach is a straightforward method to optimize bond allocation by iterative selecting bonds with the lowest cost per unit of adjusted value. This method is particularly useful when you need a simple, fast solution that does not require the complexity of optimization tools like Linear Programming. However, it may not always find the absolute optimal solution in all cases, especially when there are more complex constraints. Nonetheless, it offers a good balance between simplicity and efficiency for many practical scenarios.