# Capstone Deliverable # 3

TAO Solutions Group

Aly Dimoglou || 1006784801 || aly.dimoglou@mail.utoronto.ca
Noah Gatfield || 1006960692 || noah.gatfield@mail.utoronto.ca
Andrew Jhin || 1005774725 || jhinandr@mail.utoronto.ca
Inho Kim || 1006751294 || inho.kim@mail.utoronto.ca
Seungjun Park || 1007290405 || drew.park@mail.utoronto.ca

Total Pages: 7

Friday 18th October, 2024

# Contents

# 1 Introduction

The goal of this capstone project is to develop a tool for a third-party lender that determines the optimal allocation of loans to a set of warehouse facilities, for which there are constraints imposed by the bank of origin. The approach our team is taking to create a solution to this complex problem is to break it down into simpler problems, which through solving, we tackle different components of our target solution. The first problem is to decide whether or not a loan *can* be accepted by a facility, using the Fannie Mae loan dataset and sample asset and pool-level facility covenants. This was accomplished by implementing OOP to create classes for constraints, facilities, and loans and output if loan i can be assigned to facility j, for all i facilities and j warehouses. We achieved desirable results for the tested sample asset-level and pool-level covenants, and with a few existing loans allocated into the facilities, for testing purposes. The next step is to formulate the optimization to maximize the loan value funded while meeting all the acceptability criteria. We have formulated this problem in Section 3 and will test it using Gurobi to examine the results on a small subset of test warehouses. Once the performance of the optimization is verified, we will then extend this implementation to any number of warehouses and more sample constraints. Finally, in order to eliminate the use of Gurobi for the third-party lending company, potential options are to implement a UI so that it is not required to have knowledge of Gurobi to utilize the lending software solution we propose, or to use an existing heuristic (or even potentially develop a new heuristic) to eliminate the use of any commercial solvers in our solution. Our current progress and more details on these future progressions to achieve an ideal outcome for the project are further explored in the following sections.

# 2 Current Progress

## 2.1 OOP in Python to Determine Loan Eligibility for Warhouses

Link to Github Repository

The team has implemented Object-Oriented Programming in Python to create a framework to analyze whether a new loan would be accepted into a Facility with pre-defined constraints and existing loans:

1. Randomly select loans from the Fannie Mae data as sample loans, and create a Class for Loans.
2. Create a Class for Facilities that can take in asset-level covenants, pool-level covenants, and existing loans.
3. Create a Class for Asset-level Covenants with inputs of constraint values (min value, max value), operations (greater than or equal to, less than or equal to, etc), and the loan property to analyze with the boundary values.
   (a) For example, an Asset-level Covenant with properties of:

   $$\text{(constraint} = 60, \text{ operation} = \text{“}\geq\text{”}, \text{ property} = \text{“oltv”)}$$

   would analyze whether the loan's “oltv” value is greater than or equal to 60.
4. Create a Class for Pool-level Covenants with inputs of constraint values, type (sum, mean, etc), operations, and the loan property to analyze the boundary values.
   (a) The “type” input will analyze the combination of the existing loans and the new loan to be assigned to the facility. For example, type = “sum” will calculate the sum of the existing loans and the new loan's “property” to compare against the constraint values.
   (b) For example, a Pool-level Covenant with properties of:

   $$\text{(constraint} = 200000, \text{ type} = \text{“sum”}, \text{ operation} = \text{"}\leq\text{"}, \text{ property} = \text{"}orig_{amt}\text{"})$$

would analyze whether the sum of the "$orig_{amt}$" property of the existing and new loans would be less than or equal to 200000.

5. Create a mechanism to return "True" if and only if a new loan would satisfy all of the Asset-level Covenants and the Pool-level Covenants, and "False" otherwise. Then return if it is accepted only if both the asset-level and pool-level covenants hold, and rejected if either of them fail to be met.

The results of implementing and testing this methodology for 6 sample loans and 2 sample covenants per facility are as follows. First, the Fannie Mae data frame was filtered to exclude any loans which did not meet the asset level criteria from the set of loans to be "pre-assigned". This was so that no loans would be randomly allocated to facilities that would not have the potential to be allocated by the program (as they would be in the real use case of this program), as this may limit our testing capabilities. Then, the program we developed was used to decide whether or not the remaining 3 test loans could be assigned to a given warehouse. The code also outputs which set of covenants (asset level or pool level) caused the loan to pass or fail the acceptance criteria so that we can more easily understand and validate the performance of our code in comparison to what we expect to happen. The output of this simple test case is given in Figure 1. This test was successful, as the program made sensible decisions on whether or not a loan could be allocated to a specific warehouse, given the criteria we fed in as covenants for each warehouse.

Figure 1: Code output for test case with 6 loans and 2 facilities with both asset and pool level constraints

```
For loan 000137980826:
AssetCovenants: False
PoolCovenants: True
  - Loan is rejected by Facility f1
AssetCovenants: False
PoolCovenants: True
  - Loan is rejected by Facility f2
For loan 000137585631:
AssetCovenants: True
PoolCovenants: True
  - Loan is accepted by Facility f1
AssetCovenants: True
PoolCovenants: True
  - Loan is accepted by Facility f2
For loan 000137602540:
AssetCovenants: True
PoolCovenants: False
  - Loan is rejected by Facility f1
AssetCovenants: True
PoolCovenants: True
  - Loan is accepted by Facility f2
[[0. 0.]
 [1. 1.]
 [0. 1.]]
```

To further test this code, we will introduce more loans, warehouse facilities, and types of covenants to see if increased complexity in the number of facilities/loans or types of covenants cannot be handled by the current system. In that case, we will debug or add additional functionality to our Class for covenant allocation to ensure a wide range of covenants that a bank could enter can be accepted and used by our program.

The next step in progressing functionality is to use this framework to optimize the amount of loan value funded when there are multiple time periods for loan allocation, multiple warehouses, and more warehouse constraints. On a high level, we will start with determining which loans to assign to a single facility, then branch out to assigning multiple loans to multiple facilities. To do this, we would also need a mechanism to transform new loans into "existing loans" if they were assigned to a facility in the previous allocation period.

# 3 Mathematical Formulation

The objective is to maximize the loan balance funded while adhering to asset-level and facility-level constraints. There are $n$ loans being considered and $m$ facilities available. The balance of loan $i$ is denoted $p_i$. The decision variables $x_{ij}$ indicate optimal facility assignment where $x_{ij} = 1$ if loan $i$ is assigned to facility $j$, otherwise $x_{ij} = 0$.

$$\max_{x_{ij}} \sum_{i=1}^{n} \sum_{j=1}^{m} p_i x_{ij}$$

$$x_{ij} \in \{0, 1\}$$

Each loan can only be assigned to one facility.

$$\sum_{j=1}^{m} x_{ij} \leq \text{ for } i = 1, \ \dots \ , \ n$$

## 3.1 Facility-Level Constraints

For facility-level constraints, there are two types of constraints to consider. Parameters are defined here and examples are provided in the sections on each type. First, $a_{ij_k}$ is the property of interest of loan $i$ for constraint $j_k$. Second, $b_{ij_k}$ is the weight assigned to the property of interest for loan $i$ in constraint $j_k$. Third, $c_{j_k}$ is the average, weighted average, or proportion for constraint $j_k$, and $c_{j_h}$ is the total for constraint $j_h$. Facility $j$ has constraints of the first type $j_k$ for $j_k = 1, \ \dots \ , \ j_K$ and constraints of the second type $j_h$ for $j_h = 1, \ \dots \ , \ j_H$. Finally, the loans already in facility $j$ are denoted $j_l$ for $j_l = 1, \ \dots \ , \ j_L$.

### 3.1.1 First Type

The constraints of the first type handle averages, weighted averages and proportions based on loan properties.

$$\frac{\sum_{i=1}^{n} a_{ij_k} b_{ij_k} x_{ij} + \sum_{j_l=1}^{j_L} a_{j_l j_k} b_{j_l j_k}}{\sum_{i=1}^{n} b_{ij_k} x_{ij} + \sum_{j_l=1}^{j_L} b_{j_l j_k}} \leq c_{j_k} \text{ for } j_k = 1, \ \dots \ , \ j_K; \ j = 1, \ \dots \ , \ m$$

Example 1: If facility $j$ has constraint $j_k$ that weighted average amortization must be less than 180, then $a_{ij_k}$ is the amortization for loan $i$, $b_{ij_k} = p_i$ and $c_{j_k} = 180$.

Example 2: If facility $j$ has constraint $j_k$ that weighted average credit score must be greater than 650, then $a_{ij_k}$ is the credit score for loan $i$, $b_{ij_k} = p_i$ and $c_{j_k} = -650$.

Example 3: If facility $j$ has constraint $j_k$ that no more than 5% of loans can have LTV greater than 90, then $a_{ij_k} = 1$ if loan $i$ has LTV greater than 90 and $a_{ij_k} = 0$ otherwise, $b_{ij_k} = 1$ and $c_{j_k} = 0.05$.

The facility level constraints of the first type can be written in the standard form.

$$\sum_{i=1}^{n} (a_{ij_k} - c_{j_k}) b_{ij_k} x_{ij} \leq \sum_{j_l=1}^{j_L} (c_{j_k} - a_{j_l j_k}) b_{j_l j_k} \text{ for } j_k = 1, \ \dots \ , \ j_K; \ j = 1, \ \dots \ , \ m$$

### 3.1.2 Second Type

The constraints of the second type handle totals of the facility based on loan properties.

$$\sum_{i=1}^{n} a_{ij_h} b_{ij_h} x_{ij} + \sum_{j_l=1}^{j_L} a_{j_l j_h} b_{j_l j_h} \leq c_{j_h} \text{ for } j_h = 1, \ \dots \ , \ j_H; \quad j = 1, \ \dots \ , \ m$$

Example 1: If facility $j$ has constraint $j_h$ that total loan balance must not exceed $100,000,000$, then $a_{ij_h} = 1$, $b_{ij_h} = p_i$ and $c_{j_h} = 100,000,000$.

Example 2: If facility $j$ has constraint $j_h$ that no more than 100 loans can be in Ontario, then $a_{ij_h} = 1$ if loan $i$ is in Ontario and $a_{ij_h} = 0$ otherwise, $b_{ij_h} = 1$ and $c_{j_h} = 0.5$.

Example 3: If facility $j$ has constraint $j_h$ that no more than $50,000,000$ in loan balance can be in Ontario, then $a_{ij_h} = 1$ if loan $i$ is in Ontario and $a_{ij_h} = 0$ otherwise, $b_{ij_h} = p_i$ and $c_{j_h} = 50,000,000$.

The facility level constraints of the second type can be written in the standard form.

$$\sum_{i=1}^{n} a_{ij_h} b_{ij_h} x_{ij} \leq c_{j_h} - \sum_{j_l=1}^{j_L} a_{j_l j_h} b_{j_l j_h} \text{ for } j_h = 1, \ \ldots \ , \ j_H; \quad j = 1, \ \ldots \ , \ m$$

### 3.2 Asset-Level Constraints

Finally, asset level constraints for each facility must be enforced. The parameters $y_{ij} \in \{0, 1\}$ indicate if asset $i$ is eligible for pool $j$. They are obtained deterministically prior to optimization by checking each parameter for each asset $i$ against the minimum, maximum, inclusion and exclusion criteria for each facility $j$. If all asset level constraints are satisfied for asset $i$ for pool $j$, then $y_{ij} = 1$, otherwise $y_{ij} = 0$.

$$x_{ij} \ \leq \ y_{ij} \text{ for } i = 1, \ \ldots \ , \ n; \quad j = 1, \ \ldots \ , \ m$$

The asset level constraints can be written in the standard form.

$$x_{ij} \ - \ y_{ij} \ \leq 0 \text{ for } i = 1, \ \ldots \ , \ n; \quad j = 1, \ \ldots \ , \ m$$

# 4 Next Steps

There are multiple potential and required objectives in the scope of our next steps. Firstly and most importantly, we will have to formulate and optimize our problem using Python and Gurobi. Previously we had decided to rule out Gurobi, as the cost of a license and unfamiliarity with the software at the company is not worth the cost for the speed and flexibility of optimization methods. However, after discussion with David, we have decided to utilize Gurobi for the sake of testing the functionality of our formulation in Python and our ability to take in and linearize constraints, and make optimal decisions for allocating loans to warehouses. Therefore, using the preliminary step in the data filtering with the pool and asset level covenants, it will be fed into a Python Optimizer class that will have a function to prepare data, and another function that will have the optimization segment, passing through the data prepared in the previous function. Testing and data modeling functions may be implemented for reporting purposes.

Once the optimization, analysis, and testing of our model in Gurobi is complete and extended to the k-facility problem with increased number and more complex constraints, heuristics may be applied to eliminate the use of Gurobi or another commercial solver. Some preliminary research has been done into heuristics that could be used for this problem, including Very Large Search Networks or Branch and Bound methods. Our team's options are to manipulate existing heuristics to solve this problem or create a new heuristic that is even better suited for this problem than existing methods. However, implementing heuristics to solve this problem has been classified as our secondary objective. The most important result for our team is to find a solution to the problem of optimally allocating loans to warehouses, having the ability to take in any potential constraint that could be imposed on a warehouse and automatically alter it in the formulation such that the problem can always be solved. Therefore, this aspect of the solution will only be implemented if there is time and we have a strong solution to the primary objective.

The primary objective will be prioritized, with the secondary being in a brainstorming phase. Given that the secondary objective may not be achieved, in addition to solving the problem using Gurobi, there is potential for part of the delivery of the final product to be a front-end application that will be developed to create an interactable UI. Potential features would be to allow the user to upload loan data for allocation to facilities, the ability to edit/add constraints for each warehouse facility, and the output of optimal allocation decisions. A summary or report page can be incorporated to show the results of the optimization for our stakeholders, to show easily digestible results. For security purposes, an employee login page may have to be implemented as our stakeholders will use this tool for internal use within the company.

## References

[1] TAO Solutions. *UT_ TAO Solutions_ Warehouse Administration Solution_ August 2024.*