MASTER'S THESIS

Interuniversity Master in Statistics and Operations Research UPC-UB

Title: Cash Inventory Optimization

Author: Elena Picazo Gurina

UPC Representative: Jordi Castro Pérez

Company supervisor: Alisa Kazimirchik

Department: Statistics and Operations Research

University: Universitat Politècnica de Catalunya

Academic year: 2022-2023

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

UNIVERSITAT DE BARCELONA

I would like to express my gratitude to Alisa Kazimirchik, my mentor, and Marc Blanchart, both from Accenture.

Abstract

Keywords: Cash management problem, idle cash, genetic algorithm, MILP

In this study, we analyze the optimal level of cash held at branch offices across different regions and currencies. Freeing up excess of cash in balance enables the parent organization to utilize it for strategic investments. The main objective is to develop a cash inventory tool that maximizes net earnings of the parent organization, which implies a reduction of idle cash held at branches. The earnings come from the compound interest by investing the excess of cash in central location. However, there are foreign exchange transaction costs from transferring funds from office to central office. We need to determine the optimal fund quantities that maximize the objective function. We assume that offices are given specific safety cash levels, which do not need to be the same. Besides, there are fixed and variable transactions costs derived from the exchange of currencies.

The branch-office cash optimization problem stated above is known as the cash management problem that many firms must take care of in order to have a healthy financial balance sheet. It is similar to an inventory optimization problem since there is a target inventory level to be maintained. This level is high enough to be able to meet fluctuations in cash demand, yet low enough that unnecessary cash is minimized. The problem is formulated as a mixed integer linear programming model (MILP) in a constrained search space. To approach the problem, we use genetic algorithm, a heuristic technique based on natural selection, and Branch-and-Cut algorithm.

In terms of execution time, we show that Branch-and-Cut algorithm is a better approach for this optimization problem than genetic algorithm. However, in terms of maximizing the objective function, both algorithms do a similar improvement of the net earnings compared to the case where there is no optimization strategy (baseline approach).

List of Figures

List of Tables

Contents

Chapter 1
Introduction

Cash management is a very important and necessary activity for firms. It monitors the daily inflow and outflow of cash in balance. The opportunity cost of excess of cash, idle cash, in divisions is that the central location is losing income that it could have earned by investing it. As an example, idle cash in branches generates approximately 1% in interest whereas invested capital 3% in central office. On the other hand, if the office keeps too little cash, it takes the risk of not being able to meet the day-to-day demand for cash. Therefore, there is a clear trade-off between keeping too much or too little cash in balance in offices. Nevertheless, subsidiaries should only keep the necessary cash in balance in order to meet their obligations.

Cash management problem is commonly considered as an inventory management problem where cash is treated as a type of stock that is used for the daily functioning of an organization. It is a special form of supply chain where cash should be either made available at the right time in the right quantity or to be returned to the cash cycle. In order to maximize total earnings and minimize total costs, we need to optimize the cash level at each branch of the supply chain. It is a classic problem in a firm's financial management in order to maintain cash at the desired level. In addition, it is also known as cash balance problem.

The cash flow statement highlights a company's cash management since it displays the change in cash per period, as well as the beginning and ending balance of cash. It measures how well a company generates cash to finance its liabilities, fund its operating expenses and fund investments. It allows investors to understand how a company's operations are running, where its money is coming from, and how money is being spent. In short, CFS is one of the main input data necessary for developing a cash inventory optimization tool.

However, the main focus of this study is the management of the cash level at branch offices across the globe and the cash transfers between central location and branches, which is a very important role in cash management of international institutions. It is not the same as optimizing cash in an ATM network. The latter is specific to bank institutions and due to cash withdrawal, banks must also consider customer satisfaction when approaching the problem. In addition, ATM branches are highly affected by seasonal indices such as bank holidays, labor days and weekends. Last,

ATM cash management generally allows only decisions regarding the increase of inventory level, i.e. cash uploads.

Moreover, the branch cash optimization is more complex than just optimizing the cash levels for a single firm. For instance, it requires knowledge of the foreign exchange market when transferring funds in different currencies from branch to central hub and vice versa. When dealing with branches, they could be located in countries where the banking system is underdeveloped as well as their IT systems.

Finally, to determine the desired level of money to hold in balance, cash demand must be known in advance. In our case, it is randomly generated and it is considered as fixed input data in our study.

1. Objectives

The major challenge of this study is to determine the amount of cash to be transferred in and out of cash balances over a time horizon in order to maximize net earnings. The end goal of our project is to invest in central office the idle cash held in branch offices. The costs in this project come from exchanging currencies between branch offices and central office. We assume fixed and variable transaction costs.

The other objective of this report is to compare the solution approach of two different algorithms. Both should determine the efficient transfer quantities of funds. Branch-and-Cut (BnC) is an exact algorithm whereas GA is an heuristic algorithm. We would expect different solution sets as well as different time execution.

2. Report Organization

This thesis is organized in 6 chapters. In chapter 1 we introduce the scope of our work, which is the study of cash management problem. Chapter 2 presents the state of the art of CMP by reviewing the available research literature. Our study approach is defined in chapter 3. Chapter 4 describes the solution structure by means of BnC and GA algorithms. In chapter 5 we discuss and compare the results obtained by using the different algorithms. Last, chapter 6 concludes our analysis, in here we examine our findings and summarize the paper.

Chapter 2
Literature Review

This chapter introduces the necessary knowledge in order to better understand and follow the preceding chapters. In the first section, we discuss previous studies carried out by various researchers that tried to solve the optimization problem in hand. In the second and third sections, we review the principles of the evolutionary algorithm.

1. Cash Management Problem

The earliest cash management model is the Baumol model, also known as Baumol-Allais-Tobin (BAT) model. It is named after Baumol (1952)[**6**], Allais (1952)[**3**] and Tobin (1956)[**29**] who were the firsts economists to approach the cash optimization problem. They independently developed the model of the transactions demand for cash, which is based on establishing a firm's optimum cash balance under certainty. The firm attempts to minimise the sum of the cost of holding cash and the cost of converting marketable securities to cash. Therefore, the objective is to find the optimal withdrawal size that minimizes total costs. The model assumptions are that the firm is able to forecast its cash requirements in an accurate way. The firm's payouts are uniform over a period of time. The opportunity cost of holding cash is known and does not change with time. The firm will incur the same transaction cost for all conversions of securities into cash.

The BAT model was further developed by Álvarez and Lippi (2009)[**4**] that allowed a dynamic environment with the possibility of withdrawing cash at random times at a low cost. But, they were mainly focused on households' cash management. In addition, all random withdrawals were assumed to be free.

Nevertheless, the Baumol approach is very limited for several reasons. Foremost, firm's inflow and outflow of cash are assumed to be uniform and deterministic over a period of time, this is very unrealistic for many institutions. The transaction cost cannot be considered constant over the entire period since it usually depends on the size of the withdrawal. Moreover, interest rate is not usually fixed throughout the period when investing in securities. Finally, this approach is not suitable for international organizations with branch offices. However, mathematically speaking, the BAT model is identical to the Economic Order Quantity model (EOQ), which

is used extensively in the management of physical inventories. They both lead to the same same square-root-formula which helps in the estimation of the optimal lot size for each production run and also the number of production runs needed in a year.

Eppen and Fama (1968)[**11**] introduced linear programming techniques for the cash balance problem. The objective was to study the optimal operating policies or decision rules for stochastic cash-balance that minimizes the expected costs over some time horizon. They believed that it is similar to the dynamic portfolio problem of mutual funds and other financial institutions. They assumed that holding and penalty costs are proportional to the level of cash balance, costs incurred in transferring funds between cash and earning assets are a linear function of the amount of funds transferred. They did not prove the form of the optimal policy. However, from conducting experimental scenarios they knew that cash balance had to be between a lower and an upper bounds. These researchers had a more realistic approach to the CMP. For instance, they believed that cash flow is to some extent unpredictable and they also considered variable transaction costs. Chen and Simchi-Levi (2009)[**9**] extended Eppen and Fama research by considering fixed costs for both cash inflows and cash outflows.

In recent papers, cash management is to a large extent for optimizing cash levels in bank institutions. Moreover, literature review has a high emphasis on ATM branch cash optimization. In general, ATM cash management deals with finding the minimal amount of cash able to meet the demand over an ATM network. Management generally relies on human experience and corporate policy, which leads to static model parameters.

Bilir and Doseyen (2018) [**7**] developed an integrated cash requirement forecasting and cash inventory optimization model for both the branch and ATM networks of a mid-sized bank. The goal is to minimize idle cash for both types of branches without decreasing the customer service level (CSL) at the right time and location. It is believed to be the first integrated model in the literature. They introduced information of seasonal indices in the forecasting algorithm. They used an integer programming formulation to obtain the optimal transfer schedule for ATMs and branches by minimizing total costs. They were able to reduce both the idle cash levels and the operational cash-in-transit costs.

Salas-Molinas et al. (2020) [**25**] proposed a multiple-criteria cash management problem. They relied on goal programming and stochastic goal programming to produce cash management policies. They believe that assets have different expected returns as well as particular liquidity terms, meaning that the time period from the selling decision to the availability of cash is not necessarily zero. Goal programming aims to conciliate the achievement of a set of goals instead of optimizing every goal. In their case, to minimize cost and risk in a two-assets setting with a cash and investment accounts. They used cash balance deviations are a measure of risk and a goal programming approach by transforming hard constraints in soft ones by means of stochastic goal programming. They expressed uncertainty as lack of predictive accuracy from forecasting cash flow. The main limitation of this paper is that they specified goal weights according to the preferences of cash managers rather than relying on subjective judgments.

Up to this point, there is not much research on approaching the CMP in the ways we are going to in further sections. Our study is solely focused on optimizing cash balance for branch-office-level, no ATM network is considered, and it is not specific to bank institutions. Our cash supply chain involves only two agents, the central office and the branch offices.

2. Evolutionary Algorithms

Evolutionary Algorithms are a subset of algorithms in search of global optimization inspired by biological evolution. As Simon (2013)[**28**] mentions in his book, the terminology of evolutionary algorithms is imprecise and context-dependent. For instance, EAs are defined as population-based techniques but, they also include single-individual algorithms in the sense that they can have a single candidate solution at each iteration. On the other hand, EAs are comprised of algorithms not necessarily motivated by nature such as differential evolution and estimation of distribution algorithms. In addition, evolutionary algorithms can be referred as meta-heuristics since they use common sense approaches to solve a problem. They tend to find solutions that are close to the best solution. These meta-heuristics maintain the diversity in population and avoid the solutions being stuck in local optima.

The basic features of natural selection are the following: a biological system that includes a population of individuals, many of which have the ability to reproduce; individuals have a finite life span; there is variation in the population; ability to survive is positively correlated with the ability to reproduce.

The classic evolutionary algorithms are genetic algorithms, evolutionary programming, evolution strategies and genetic programming. Our study mainly analyses genetic algorithms.

2.1. Genetic Algorithm. (GA) is the first implementation of EAs. They are simulations of natural selection that can solve optimization problems. However, GAs are not limited to optimization applications.

We assume that genetic algorithm is a population based search algorithm. Each potential solution is a "candidate solution" or "individual" (analogous to chromosome). Every individual has a set of features (analogous to genes) that can be evolved and changed. Just as in a chromosome, each gene controls a particular characteristics of the individual, similarly, each bit in the string represents a characteristics of the solution. There most popular encoding methods are binary, octal, hexadecimal, permutation, value-based and tree. In addition, features are considered as points in the solution space.

Each individual in the population is coded as a finite length vector (chromosome) of components (genes). A group of individuals is called the "population" of the GA. Figure 1 illustrates that population of individuals are maintained within the search space.

FIGURE 1. Search Space: Gene, Chromosome, Population

To begin the genetic algorithm, we randomly generate a set of individuals. Then, new populations are produced by iterative use of genetic operators on individuals present in the population.

For every generation, the fitness of each individual in the population is evaluated. The fitness is the value of the objective function being solved. In cash management optimization, the fitness function could be the minimization of transaction costs or maximization of net earnings. Individuals with good fitness score are given more chance to reproduce. Thus, each successive generation is more suited for their environment. Moreover, the population size is static so the room has to be created for new arrivals. For instance, some individuals die and get replaced by new individuals and eventually creating a new generation when all the mating opportunity of the old population is exhausted. It is expected that over successive generations, better solutions will arrive while least fit die.

The algorithm stops for two reasons. One possibility is that the GA can run for a predetermined number of generations. Another possibility is for the GA to run until the fitness of the best individual is better than some user-defined threshold. Another possibility is for the algorithm to run until the fitness of the best individual stops improving from one generation to the next. This means that the offspring generated has no significant difference compared to the offspring generated by the old population. This indicates that the algorithm is said to be converged to a set of solutions for the problem.

Once the initial generation is created, the GA evolves the generation using some operators discussed as follows. Figure 2 depicts the basic structure as well as the sequence of the operators of the algorithm.

The selection operator gives preference to the individuals with good fitness scores and enables them to pass their genes to successive generations of the individual. It is also known as reproduction operator since it determines whether an individual will participate in the reproduction process or not. The most common selection techniques are roulette-wheel and tournament.

Roulette-wheel selection is referred as fitness-proportional selection or fitness proportionate selection. Each individual has a probability of being selected that is proportional to the amount by which its fitness is greater or less than its competitors' fitness. The wheel is rotated randomly to select specific solutions that will participate in formation of the next generation. The main downside of this technique is that it introduces errors due to its stochastic nature.

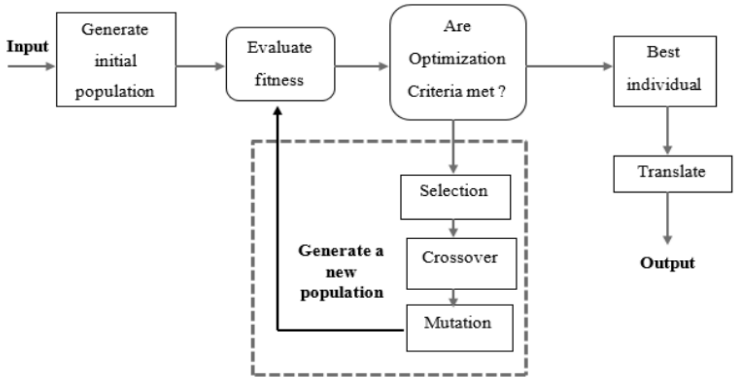


FIGURE 2. The sequence of operators in Genetic Algorithm

Tournament selection reduces the computational cost associated with selection. Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. For instance, in a K-way tournament selection, only the individual with the highest fitness value from each k-subgroup is chosen to reproduce. This selection strategy has a parameter called the selection pressure, which is a probabilistic measure of a candidate's likelihood of participation in a tournament. If the tournament size is larger, weak candidates have a smaller chance of getting selected as it has to compete with a stronger candidate. Hence, the selection pressure parameter determines the rate of convergence of the algorithm. More the selection pressure, the more will be the convergence rate. GAs are able to identify optimal or near-optimal solutions over a wide range of selection pressures. Tournament Selection can also work for negative fitness values.

The advantage of tournament selection compared to other types of selection is that it can work with only subjective comparisons between individuals. In other words, we only need to know the relative fitness values of the individuals in the tournament.

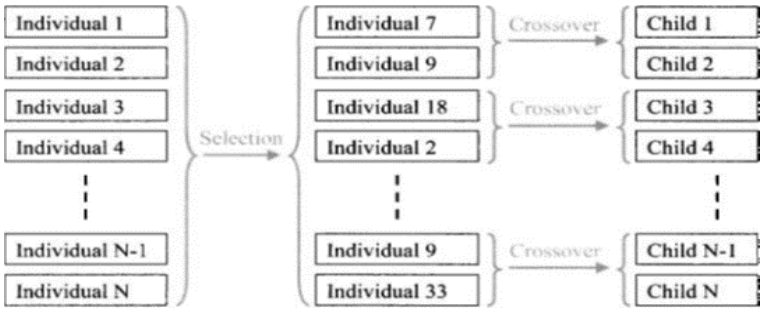


FIGURE 3. Crossover of parents to create children

Once the individuals have been selected, the next step is to generate the offspring with the crossover operator. For instance, two parents are picked from the mating pool at random to crossover (analogous to sexual reproduction) to produce two children. We then repeat the process to obtain two more parents, mate them, and

obtain two more children. This process is repeated until the population of children is the same size as the population of parents. This idea is illustrated in Figure 3.

There are many different types of crossover such as two-point or k-point, uniform or shuffle, amongst others. But the most common one is the single point crossover. In a single point crossover, a random crossover point on the parent individual is selected. All data/genes beyond that point in the two parents is swapped between each other. In Figure 4 there is a clear representation of this one point crossover.

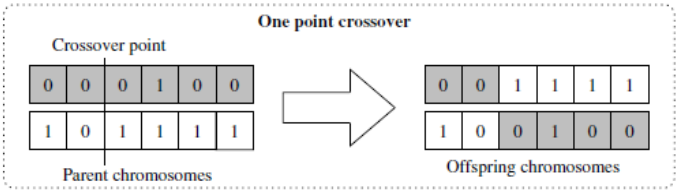


FIGURE 4. Single Point Crossover

Mutation operator maintains genetic diversity from one population to the next population. The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. If the gene is selected for mutation, you can either change it by a small amount or replace it with a new value. Additionally, it ensures that offspring are not all exactly the same, you allow for a small chance of mutation.

In practice, mutation allows the evolutionary process to explore new potential solutions to the problem. If some genetic information is missing from the population, mutation provides the possibility of introducing that information into the population. This is very relevant for GAs since they tend to work with very small population sizes (around 100) that inbreeding can easily become a problem, and evolutionary dead ends are more common in GAs than in biological evolution.

The mutation rate determines whether a gene of an individual is selected for mutation or not. Since mutation in biology is relatively rare, the mutation rate should be very low. For instance, it could be around the order of 2%. For example, assume that features (genes) are encoded as binary digits as strings of 0s and 1s and crossover generated the offspring. A mutation probability of 1% implies that each bit in each child has a 1% probability of flipping to the opposite value (a 1 changes to a 0, a 0 changes to a 1). It is important to select a reasonable mutation probability. Too high of a mutation probability makes the algorithm behave like a random search, which is not usually a good approach to solve a problem. Too low of a mutation probability results in problems with inbreeding and evolutionary dead ends, which also prevents the GA from finding a good solution.

Even if genetic algorithm was one of the first evolutionary algorithms, it still remains as one of the most popular. It is easy to implement, it has a very intuitive approach and it has good performance on a variety of problems. However, we must keep in mind that there are many variations of GA such as the messy GA, gender-based GA, island GA, and so on.

Now, we present two research papers that use genetic algorithms for solving the cash management problem. Armenise et al. (2011) [**5**] optimized the ATMs cash management problem by genetic algorithms. The application of GAs as means for searching and generating optimal upload strategies, aimed at identifying a set of uploading rules able to minimize the daily amount of stocked money and to guarantee service availability at the same time. They considered a pool 30 ATMs with different characteristics in terms of location, position, cash capacity and usage. Instead of forecasting future demand of cash, they are interested in finding a set of rules that if applied to current operation of ATMs can lead to the decision of when and how to upload the machine. They named this approach as Condition Based Cash Management (CBCM). Experimental results proved that this approach is feasible and able to improve cash management if compared to human expertise.

Moraes and Nagano (2013)[**21**] compared the Miller-Orr Model [**20**] with computational evolutionary models to minimize the total cost of cash balance maintenance. The objective of the paper is to develop a management policy of cash balance, based on the assumptions of cost minimization by applying genetic algorithms and particle swarm optimization (PSO) and comparing the results with the traditional Miller-Orr model. They developed computational experiments from cash flows simulated to implement the algorithms. They showed that both algorithms present significant gains in relation to cost and time compared to the optimized Miller-Orr model. In addition, both algorithms can be applied for the definition of more complex cash policies, without the limitations of the Miller-Orr model. This study presents its contribution to the validation of GA and PSO algorithms.

There is extensive research of genetic algorithm in inventory management so as to minimize the total supply chain cost. However, there are few analyses using genetic algorithm for the CMP. Therefore, our study will expand the literature review on genetic algorithms and cash balance problem.

Chapter 3
Methodology

In this chapter, we explain how we approach the cash management problem. For instance, we define the problem, describe the input data and derive the objective function with its corresponding constraints.

1. Problem Definition

It is a deterministic cash management problem because cash demand is known and it is given as input. Demand of cash can be negative or positive. For instance, negative demand of cash refers to accounts payable (cash outflow) and positive demand of cash refers to accounts receivable (cash inflow).

We are concerned with a two-echelon supply chain model, where multiple divisions take cash from a a central office. This scenario describes pooling, which occurs when cash is held as well as managed in a central location. In this centralization system, funds can be moved from the central location to where they are needed. Moreover, we assume independence of branch offices. For example, if the need in a branch is unusually high, it can be met from the central pool because there will not normally be unusually high drains in other countries at the same time. In addition, we do not consider complete centralization of management by leaving some cash in offices since local representation is often necessary for dealing with local clients and banks.

We are focused on cash optimization and not forecasting cash flow for a planning horizon of 31 days. The decisions that need to be made are based on a daily basis: how much cash inventory to hold as well as the size and timing of the cash transfers. Since both divisions and central office have positive interest rates, there is a trade-off between where money should be hold while taking into account the cost of the transactions. But, we do specify our preference of keeping more cash in central office by assigning a larger annual interest rate for the central office than for the offices. For instance, central office has an interest rate of 12%, then, offices should have interest rates below 12%.

As commented in previous sections, we are concerned in maximizing total net earnings, which is computed as total earnings from offices and central office minus transaction costs involved in exchanging currencies.

Earnings are determined by the compound interest. The formula of the compound interest is as follows:

$$X = P\left(1 + \frac{r}{n}\right)^{nt} - P \tag{1}$$

- X is the compound interest
- P is the principal
- r is the annual interest rate
- n is the number of compounding periods per unit of time
- t is the number of time units the money is invested

In our case, P is the cash balance at end of day, n is 365 days since the compounding frequency is daily and t is $\frac{1}{365}$ because the money is invested in a daily basis. Therefore, equation 1 becomes:

$$X = P\left(1 + \frac{r}{365}\right)^{365*\frac{1}{365}} - P = P\left(1 + \frac{r}{365}\right) - P \tag{2}$$

With regard to the costs, there are fixed and variable transactions costs from exchanging currencies. The fixed transaction cost could be seen as a lump sum fee for sending/receiving cash to/from central office. On the other hand, variable transaction costs depend on the ask price, which is the lowest price at which we would pay with local currency (LCY) to a dealer, perhaps a bank, in order to buy USD currency. The problem's perspective is from the point of view of the branch office in the sense that funds are in LCY. And central location works with USD currency. This is the reason why we use the ask price for defining the variable transaction cost instead of the bid price. The bid price is the highest price at which the bank sells USD currency in order to buy LCY. We are in the situation where we convert local currency in USD currency: LCY $\rightarrow \frac{1}{FXspot}USD$. FXspot for us, indicates the foreign exchange spot rate, which is the exchange rate at which the agreement between two parties to buy one currency against selling another currency at an agreed price for settlement on the spot (current) date. In our case, the two parties are branch and central office that agree to buy USD currency for the selling of LCY. For example, if the exchange is between Japanese currency (JPY) and USD currency with $FXspot = 110JPYUSD$, this implies that 1USD is equivalent to 110JPY. However, we are interested in $\frac{1}{FXspot}USD$. If there was no foreign exchange spread, we would only need $\frac{1}{FXspot}$ in order to convert local currency in USD currency. Yet, in our problem, we do specify foreign exchange spread and, therefore, we define the variable transaction cost as follows:

$$\frac{FXspread}{FXspot * (1 - FXspread)}, \frac{1}{FXspot * (1 - FXspread)} \tag{3}$$

FXspread is the foreign exchange spread, FXspot is the foreign exchange spot rate and the ask price is the second term in equation 3. In addition, from now on, we define the variable transaction cost as the spread cost.

If funds were in USD currency, we would need to specify the spread cost with the bid price. It would be as in equation 3, but with a positive sign in the denominator since the bid price must be higher than the ask price. For instance, it would be like this:

$$\frac{FXspread}{FXspot * (1 + FXspread)} \tag{4}$$

The transaction amounts are one of the components that define the cash balance at the end of the day. More specifically, the end cash balance depends on funds (transfer quantities) and cash demand from current period. It also depends on the end cash balance from previous period. Whereas central office only takes into account funds from current period and end cash balance from previous period. From branch perspective, a negative fund implies that there is an outflow of cash from division to central office. However, a negative fund from the central office perspective indicates that the outflow of cash is from central office to division.

We assume that funds and cash end of branch offices have lower and upper bounds. For the former, the range where transfer quantities can vary from is between negative and positive values, it is branch-specific. For the latter, the end cash balance should fluctuate around the safety level assigned for each branch. For instance, Japan office has a safety level of $20000JPY$, this means that the cash at the end of the day should oscillate around this value.

Finally, we assume that there are no lead days between the day where a transfer is generated and the day where the transfer is received. In other words, there is instant cash transfers. In addition, our study does not take into account cash-in-transit companies, those that are involved in transporting cash. Transactions are made within bank accounts and we do not take into account the number of transactions to be made as a parameter of the problem.

2. Input Data

The data has been generated randomly and the time horizon is of 31 days, from 1st March 2019 to 31st March 2019. The central office is assumed to be United States (USD) and the offices are located in Japan (JPY), Germany (EUR), India (INR), Australia (AUD) and United Kingdom (GBP). As commented before, we assume that the cash end in offices oscillate around their safety cash levels and, hence, they all must have positive cash balances. Last, we assume that central office has

unlimited cash in balance and, therefore, it is not restricted to oscillate around a specific quantity.

The parameters specific for the central office are the interest rate (12%) and fixed transaction costs per transaction (10USD from office to central, 20USD from central to office). On the other hand, there are seven country-specific parameters: the interest rate, initial cash end, foreign exchange spot rate, demand of cash, safety cash level, upper bound for transfer quantities, foreign exchange spread. These parameters might be different between offices. For example: Ã§

- For Japan, United Kingdom, Australia the interest rate is of 1%, and for India and Germany is 2%.
- The FX spread is 4 basis points for India and United Kingdom, 5 basis points for Japan, 6 basis points for Australia and 2 basis points for Germany.
- The safety cash level for Japan is 1M JPY, 10M LCY for United Kingdom and Germany, 100M LCY for Australia and India.

For the optimization process, we only require the cash end from 28th February 2019. However, we also generate randomly data for funds (transaction quantities), generated without following any optimization strategy. With this input data, that can be found in Table 1 in Appendix B, we compute the baseline end cash balance, which will be useful to compare it with the end cash balance optimized with Branch-and-Cut and genetic algorithms.

3. Model Definition

We are interested in optimizing the transfer quantities for each office and since they are independent from each other, the mathematical formulation does not aggregate all offices but instead, it only considers the relationship between one branch and the central office.

3.1. Parameters. These are the parameters of the model:

- D number of days of the planning horizon, i in $1..D$
- $CFOC$ lump sum fee from office to central office (in USD currency)
- $CFCO$ lump sum fee from central office to office (in USD currency)
- $FXspread$ foreign exchange spread from office to central office (in basis points)
- $FXspot_i$ foreign exchange spot rate from office j to central office for day i
- $demand_i$ cash demand for office for day i (in local currency)
- rO annual interest rate for office
- rC annual interest rate for central office
- $startO$ initial cash end for day $i = 0$ for office (in local currency)
- $startC$ initial cash end for day $i = 0$ for central office (in USD currency)
- ss safety stock level for office (in local currency)
- ub upper bound for transfer quantity for office (in local currency)
- lb lower bound for transfer quantity (in local currency)

3.2. Decision Variables. The problem is a mixed integer programming model because there are integer, binary and continuous decision variables:

- X_i transfer to office from central office for day i (in local currency)
- Y_i transfer from office to central office for day i (in local currency)
- W_i 1 if $X_i >= 0$, 0 otherwise
- Z_i 1 if $Y_i >= 0$, 0 otherwise
- $endO_i$ cash end for office for day i (in local currency), i=0,...,D
- $endC_i$ cash end for central office for day i (in USD currency), i=0,...,D

However, the relevant decision variables of this MILP model are the funds transferred in and out of the cash balance from each branch office, which are assumed to be integers: X_i, Y_i.

3.3. Objective function. The objective function is defined as net_earnings = total_earnings - transaction_costs.

Total earnings is the sum of earning for both office and central office for the whole time horizon:

$$
\begin{aligned}
earnings\_offices &= \sum_{i=1}^{D} \left(\frac{I_i \left(\left(1 + \frac{rO}{365} \right) - 1 \right)}{FXspot_i} \right) \\
earnings\_central &= \sum_{i=1}^{D} \left(I_i \left(\left(1 + \frac{rC}{365} \right) - 1 \right) \right)
\end{aligned} \tag{5}
$$

Then, these are the transaction costs from transferring funds between office and central office for the whole time horizon:

$$
TC = \sum_{i=1}^{D} (W_i * CFCO + X_i * spread_{cost} + Z_i * CFOC + Y_i * spread_{cost}) \tag{6}
$$

where spread cost (variable transaction cost) is:

$$
\frac{FXspread}{FXspot * (1 - FXspread)} \tag{7}
$$

3.4. Constraints. There are eight constraints, which define the search space (feasible region) of this problem.

Initial cash end:

- For office: $endO_0 = startO$ for $i = 0$
- For central office: $endC_0 = startC$ for $i = 0$

Cash Balance:

- For office: $endO_i = endO_{i-1} + X_i - Y_i + demand_i \quad \forall i \in \mathcal{D}$
- For central: $endC_i = endC_{i-1} - X_i * ask + Y_i * ask \quad \forall i \in \mathcal{D}$, where ask is the ask price $\dfrac{1}{FXspot_i * (1 - FXspread)}$

Logical constraints:

- $X_i \leq W_i * M \quad \forall i \in \mathcal{D}$
- $Y_i \leq Z_i * M \quad \forall i \in \mathcal{D}$
- We assume that for each day, there can be only an outflow or inflow of funds or no funds at all, but not both at the same time: $W_i + Z_i \leq 1 \quad \forall i \in \mathcal{D}$

In this last constraint, we specify that the end cash in each office should not be below its safety cash level nor greater than 1.5 times its safety level:

- $ss \leq endO_i \leq ss * 1.5 \quad \forall i \in \mathcal{D}$

Chapter 4
Model Implementation

In this chapter, we describe how we implement the mixed integer programming model, defined in previous chapter, in AMPL and in Python. In AMPL is where we execute BnC algorithm and in Python we execute genetic algorithm.

1. CPLEX Implementation

The optimization software package cplex used in AMPL implements by default branch-and-cut, which is based on branch-and-bound and cutting planes in order to find the optimal solution through relaxing the problem to produce the upper bound and not the lower bound since we are in a maximization problem. However, it may be the case where cplex can solve the problem without implementing Branch-and-Cut and, instead implements some internal heuristics.

In order to solve this problem in AMPL, we define .mod, .dat and .run files. The .mod file contains the mathematical formulation and it is common for all countries. Whereas, the .dat contains the input data and it is country-specific. Then, the .run file contains AMPL script in order to execute BnC with the corresponding .mod and .dat files. For instance, we are analyzing 5 countries, therefore, in our example we have one .mod file and five .dat files. Below is the specification of the .mod file.

.mod file

```
param D;

#transaction cost variables
param CFCO;
param CFOC;
param FXspread;

param demand {1..D};# cash dda for day i in office
param rO>=0; #positive interest for office
param rC >=0;#positive interest for central office
param startO>=0;#cash start at office for day 0
param startC >=0;#cash start for central office for day 0
param ss>=0;#safety level fot office
param ub>=0;#upper limit for transfer quantities
```

```
param lb<=0;#lower limit for transfer quantities
param bigM;
param FXspot {1..D} >=0;#fx rate (LOCAL/USD) for day i for office

# decision variables
var X {i in 1..D} <= ub, >= lb integer;
var Y {i in 1..D} <= ub, >= lb integer;
var endO  {0..D} >=0;#cash_end for officeon day i
var endC {0..D} >=0;#cash_end for central office on day i
var W {i in 1..D} binary;#W=1 if X>=0
var Z {i in 1..D} binary;#Z=1 if Y>=0

# objective function
maximize net_earnings: sum{i in 1..D} endC[i]*((1+rC/365)-1)
            + sum{i in 1..D} (endO[i]*((1+(rO/365))-1))/FXspot[i]
            - sum{i in 1..D} W[i]*CFCO
            - sum{i in 1..D} X[i]*(FXspread/(FXspot[i]*(1-FXspread)))
            - sum{i in 1..D} Z[i]*CFOC
            - sum{i in 1..D} Y[i]*(FXspread/(FXspot[i]*(1-FXspread)));

#constraints
subject to c1: endO[0] = startO;
subject to c2: endC[0] = startC;

subject to c3 {i in 1..D}: endO[i] = endO[i-1] - Y[i] + X[i] + demand[i];

subject to c4 {i in 1..D}: endC[i] = endC[i-1]
                        + Y[i]*(1/(FXspot[i]*(1-FXspread)))
                        - X[i]*(1/(FXspot[i]*(1-FXspread)));

subject to c5 {i in 1..D}: X[i] <= W[i]*bigM;
subject to c6 {i in 1..D}: Y[i] <= Z[i]*bigM;
subject to c7 {i in 1..D}: Z[i] + Z[i] <= 1;
subject to c8 {i in 1..D}: ss <= endO[i] <= ss*1.5;
```

The .run file executes the .mod file for each .dat file in a for loop. For instance, it first executes the .mod file with cash__optimizer_1.dat, then with cash__optimizer_2.dat until the fifth country. The data for Australia is in cash__optimizer_1.dat, for Germany in cash__optimizer_2.dat, for India in cash__optimizer_3.dat, for Japan in cash__optimizer_3.dat and for United Kingdom in cash__optimizer_5.dat.

.run file

```
reset;
model cash_optimizer.mod;
set COUNTRIES = 1..5;
for {j in COUNTRIES}{
        print ("Country" & " " & j);
        reset data;
        data ("cash_optimizer_" & j &".dat");
        option solver cplex;
        solve;
```

```
        option display_precision 12;
        display X, W, Y, Z, endO, endC;
}
```

Below is the .dat file for Japan. For the other offices, you can find them in the Appendix B. The central office parameters (CFCO, CFOC, D, rC, startC) and ub are common in all .dat files.

cash_optimizer_4.dat file

```
param D:=31;

#transaction cost variables
param CFCO:=20;
param CFOC:=10;
param FXspread:=0.0005;

param rO:=0.01;
param rC:=0.12;

param startO:=1500429;
param startC:=6000000000;

param ss:=1000000;

param ub:=50000000;
param lb:=0;

param bigM:=100000000;

param demand:=
1          -8021355.96
2          -8021355.96
3          -8021355.96
4          -8021355.96
5          -8021355.96
6          -8021355.96
7          -8021355.96
8         -11576593.48
9         -11576593.48
10        -11576593.48
11        -11576593.48
12        -11576593.48
13        -11576593.48
14        -11576593.48
15          6818423.52
16         -2188840.2
17         -4218884.2
18         -4218884.2
19         -4218884.2
20         -4218884.2
21         -4218884.2
```

```
22          -4218884.2
23            791064.13
24            791064.13
25            791064.13
26            791064.13
27            791064.13
28            791064.13
29            791064.13
30            791064.13
31            791064.13;

param FXspot:=
1          111.716
2          111.716
3          111.716
4          111.966
5          111.916
6          111.806
7          111.711
8          111.066
9          111.066
10         111.066
11         111.126
12         111.356
13         111.301
14         111.601
15         111.706
16         111.706
17         111.706
18         111.531
19         111.276
20         111.586
21         110.496
22         110.781
23         110.781
24         110.781
25         109.996
26         110.046
27         110.601
28         110.171
29         110.766
30         110.766
31         110.766;
```

2. GA Implementation

To implement the genetic algorithm in Python, we use the ga package developed by Accenture, which is based on deap package. In addition, the tools module contains the operators for the GA such as, initialization, crossover, mutation and selection.

Now, the decision variables W_i, Z_i are no longer needed because in Python their role of activating or deactivating decision variables X_i, Y_i is replaced by if conditional statements. Moreover, we just use one decision variable, for instance *decision*, that can have negative and positive values. If X_i takes negative values, it refers to Y_i since there is an outflow of funds from office to central office. On the other hand, if the decision variable is positive, it refers to X_i since there is an inflow of funds from central to office. Below is the structure of the pseudo-code.

GA pseudocode

```
START
Generate initial population
Compute fitness of each individual in the population
WHILE optimization criteria not achieved do {
        Select individuals (parents) with better fitness
        Build new individuals (offspring)
            Crossover
            Mutate
        Evaluate ossfpring
        Replace population with new individuals
}
END ALGORITHM
```

To generate the initial population, we register a population function that creates individuals (chromosomes) from randomness and/or from user input. For our example, the number of individuals for all countries is 300, except for Germany. Germany requires a larger number of individuals, for instance, 400, so that there is enough diversity for convergence. More specifically, Germany office with a population size of 300 outputs as local optimum the predefined individual. This is illustrated in Figure 6 in Appendix B. Whereas, with a population size of 400, genetic algorithm is able to converge to a local optimum different from the predefined individual.

An individual is a solution, list of transfer quantities (genes) for each simulation date. In this case, the individual is of length 31 since the time horizon is of 31 days. If the individual is generated randomly, it is generated within a specified interval delimited by upper and lower bounds with function random.randint() from numpy package. For example, for Japan office an individual contains 31 transfer quantities that are generated randomly within the range -50M and 50M in JPY currency. An individual generated from user input can speed up the convergence since it gives an initial individual that satisfies the constraints mentioned in the previous chapter.

Once the initial population is generated, we assign a fitness value to each individual. Since we are in a maximization problem, the more positive fitness values, the better. But first, the individuals must satisfy the last constrained defined in the last chapter in order to be evaluated. If they satisfy the constraint, they will be given a positive fitness value, which refers to net earnings. This positive fitness value is the value of the objective function specified in Chapter chapter 3 for a given set of transfer quantities.

An individual that does not satisfy the constraint $ss \leq endO_i \leq ss * 1.5 \quad \forall i \in \mathcal{D}$ is considered infeasible and it is given a negative fitness value. This negative fitness value is formed by a penalty, a fixed term, and a distance. The distance is proportional to how far the individual is to be feasible. For example, assume that the safety cash level is 10 for a given office, the time horizon is of 4 days and the penalty is 5. Assume individual A has end balances $-10, 5, 0, 12$ for a given random list of transfer quantities and individual B has end balances $0, 11, 15, 2$ given another list of random transfer quantities. In this toy example, both individuals are infeasible since both have at least one day where end cash balance is below 10 and, therefore, both are given the fixed penalty of 5. But, individual B is closer to be feasible (its fitness score is closer to 0) than individual A because it only has 2 days where it has end balance below 10. Individual A has fitness score of $-40 = -penalty - distance = -5 - (20 + 5 + 10)$: 20 is the quantity it needs in order to have 10LCY in end cash balance for day 1, it needs 5LCY to reach 10LCY for day 2, 10LCY for day 3 and none for day 4. Whereas, individual B has only a fitness score of $-23 = -5 - (10 + 8)$. In our example, we specified a penalty of 1M in LCY for each country.

This initial population with evaluated individuals is considered the generation 0. In order to enter in the while loop, we must check whether the optimization criteria is met or not. The optimization criteria is based on parameters related to the number of generations. The algorithm will stop if at least one of these conditions is not met:

- The number of generations with feasible solutions (parameter "gen") is smaller than parameter "generations".
- The number of generations with the same solution (parameter "same") is smaller than parameter "convergence_generations".
- The total number of generations (parameter "real_gen") is smaller than parameter "force_end_generations".

Parameters "ngen", "same", "real_gen" are set to 1, 0, 1 respectively. Therefore, we start the while loop since any of the optimization criterion is violated. In addition, for all offices parameter "generations" is 50, "convergence_generations" is 30 and parameter "force_end_generations" is 60.

The parent individuals that are selected for mating (crossover) is based on the Tournament selection described in Chapter chapter 2. The selection operator applied to the population is deap.tools.selTournament(), which takes as arguments the number of tournaments and the size of each tournament, the criteria of selection is based on the fitness score of the individuals. For us, the tournament size is 30% of the total number of individuals in the population (parameter "sel_size"), hence, 120 for Germany and 90 for the other countries and there are 400 or 300 tournaments, depending whether it is Germany or not.

The mating process amongst the selected parents is done with the cross-over operator deap.tools.cxTwoPoint(), which executes a two-point crossover on the parent individuals. The probability of two consecutive individuals to be selected for crossover is 0.9. (parameter "cross_prob") Figure 1 depicts an example of how it would be done for an individual with 0-1 values as genes.

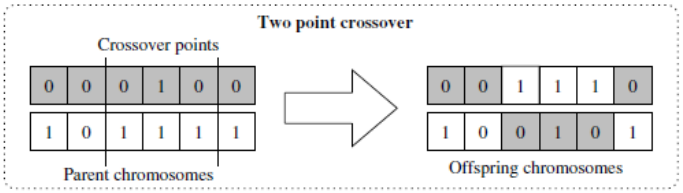


FIGURE 1. Two Point Crossover

After the mating process, the parent population is completely replaced by the offspring. However, before evaluating this new population, some children in the offspring are selected for mutation with probability of 1 (parameter "mut_prob"). The mutation rate is $\frac{1}{num\_genes} = \frac{1}{31} = 0.03225$ so that in average only one gene is mutated. Then, the probability that a gene is mutated is 3% for an individual that has been selected for mutation. We apply Gaussian mutation centered at the real value of the gene (mean is 0) and sigma 0.1 (parameter "mut_sigma") on the selected gene with random.gauss().

Once the mutation is done, the offspring can be evaluated in the same way as for the initial population. In each generation, we save the best individual found so far in the Hall of Fame with the class deap.tools.HallofFame(). If in the current generation, the algorithm does not find a better individual than in previous generation, then, Hall of Fame is not updated and it keeps as best the individual from previous generation.

This first iteration is finished. We move onto the second while loop if the optimization criterion are still holding. For all five countries, the while loop stops when we reach the value 50 of parameter "generations".

Below is the configuration file for Japan office, needed in order to execute genetic algorithm, with some of the parameters mentioned before. The remaining "config_ga" parameters are defined as follows:

- "mut_start_generations": number of initial generations where the number of genes to be mutated is incremented.
- "mut_start_weight": probability to increment the mutation process during the start generations.
- "mut_local_generations": given number of generations obtaining the same best individual.
- "mut_local_optimal_weight": probability to increment the mutation process during previous local generations.
- "multiprocessing": evaluation step is done in parallel
- "num_processors": number of processors to perform parallelization

For the other countries, the "config_ga" parameters are the same, except for Germany that has "population_size" 400. For example, for a given country, for the first 10 generations ("mut_start_generations") the mutation rate is $p \in [0.03225, 0.5]$ and it is generated with random.uniform(), where 0.5 is the "mut_start_weight". For generations greater than 10, the mutation rate is just 0.03225.

We assume we reach a local optima if after 30 consecutive generations ("mut_local_generations") the best individual is still the same one. If so, the mutation rate is $p \in [0.03225, 1]$ and it is generated with random.uniform(), where 1 is the "mut_local_optimal_weight".

In general, the execution of genetic algorithm is rather slow, therefore, we introduce multiprocessing. It allows the algorithm to evaluate the individuals in parallel by using eight Python processes ("num_processors"). Moreover, we add to the initial population, an individual that satisfies the model constraint $ss \leq endO_i \leq ss * 1.5 \quad \forall i \in \mathcal{D}$. Therefore, the initial population will be composed of random individuals and one predefined individual. However, this predefined individual does not take into account the objective function, it is just a feasible solution satisfying the mentioned constraint.

After some manual parameter tuning, we believe that the values of the parameters in the configuration file are reasonable because they guarantee diversity and convergence. The configuration files for the remaining countries and the initialization of the algorithm in Python can be found in Appendix A.

ga_config_Japan.json

```
{
    "config_ga": {
            "random_seed": 64,
            "population_size": 300,
            "generations": 50,
            "force_end_generations": 60,
            "convergence_generations": 30,
            "sel_size": 0.3,
            "mut_integer": "gaussian",
            "mut_sigma": 0.1,
            "mut_prob": 1,
            "cross_prob": 0.9,
            "mut_start_generations": 10,
            "mut_start_weight": 0.5,
            "mut_local_generations": 30,
            "mut_local_optimal_weight": 1,
            "multiprocessing": true,
            "num_processors": 2
    },
    "optimization_range": [-50000000,50000000],
    "constraint_penalty": 1000000,
    "previous_solutions": [[7940000, 8002279, 8002279, 8002279,
                         8002279, 8002279, 8002279, 11700079,
                        11602200, 11599200, 11593339, 11243589,
                        11900696, 11219056, -6900000, 2209488,
                        4177168, 4277168, 4477168, 4377168,
                        3777168, 4477168, -988198, -900009,
                        -709999, -857667, -322052, -1116568,
                        -922000, -424010, -988888]],
    "objectives": {
```

```
                "net_earnings": "max"
        },
        "output": {
                "write_outputs": 1,
                "output_path": "./ga_output/Japan/"
        }
}
```

Chapter 5
Results

The comparison between Branch-and-Cut and genetic algorithms is done in terms of computational time and net earnings. For the former, we compare the computational time and the number of iterations needed to execute each algorithm for each country. For the latter, we compare the net earnings obtained from executing BnC algorithm, GA and baseline approach.

From Table 1, we show that for any office, cplex did not implemented Branch-and-Cut algorithm. Instead, cplex applied MIP simplex iterations, except for Japan office. For this branch, cplex could solve the problem by just applying internal heuristics. Perhaps it is because the cash end of Japan oscillates around 1M JPY. Whereas for the remaining countries, the cash end is around 10M or 100M LCY. For BnC algorithm we did not required the use of additional features in order to reduce the computational time. For each country, it only took 1 second to output the results.

The results for genetic algorithm in Table 1 take into consideration the use of a predefined solution and the multiprocessing module. As expected, multiprocessing speeds up the computational time of executing GA compared to the execution without multiprocessing. More specifically, the computational time is reduced by more than 70%. However, genetic algorithm is still slower than BnC even with multiprocessing. In addition, for all countries, GA stop at iteration 50 and not before because the algorithm did not find 30 consecutive generations with the same solution. There are additional results from executing genetic algorithm in Appendix B.

The main difference in introducing the predefined solution to the genetic algorithm is the execution time. With multiprocessing module, genetic algorithm converges to a solution after 22 hours of execution for Japan office and the net earnings are only 10USD higher than the case with the predefined solution. The results without the predefined solution can be found in Appendix B. Perhaps, the execution time for the other offices without their predefined solutions is similar to Japan office.

In terms of net earnings, BnC algorithm does a slightly better job when maximizing the objective function compared to genetic algorithm, as we can see in Table 2. The % increase in net earnings with BnC is slightly greater than the % increase in net

Country	BnC		Genetic Algorithm		
	Time	Iterations	Multiprocessing	No Multiprocessing	Iterations
Australia	1 sec.	98	3 min.	12 min.	50
Germany	1 sec.	15	4 min.	16 min.	50
India	1 sec.	65	3 min.	12 min.	50
Japan	1 sec.	0	3 min.	11 min.	50
UK	1 sec.	97	3 min.	11 min.	50

TABLE 1. Algorithm Results

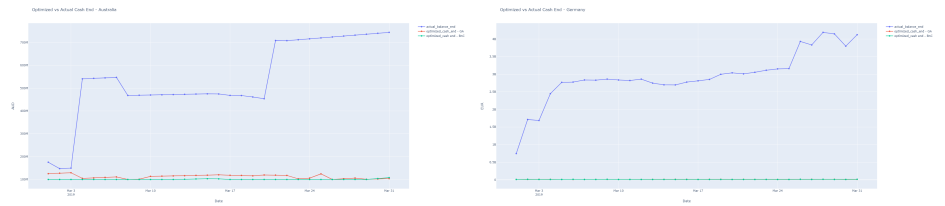
Country	Baseline	BnC	% increase	GA	% increase
Australia	60950523.9	63837005.11	4.74%	63732386.71	4.56%
Germany	55704296.17	83049206.1	49.1%	83018087.88	49%
India	60896002.81	61010592.91	0.19%	61006739.51	0.18%
Japan	60766453.31	61139160.22	0.61%	61139144.48	0.61%
UK	55600072.42	61613267.3	10.82%	61581444.42	10.76%

TABLE 2. Comparison Table - Net Earnings

earnings with GA for any country, given the net earnings of the baseline approach (net earnings with random transfer quantities). Moreoever, for Germany and UK offices there is a considerable increase in net earnings when optimizing transfer quantities with BnC or genetic algorithms, 49% and 10% respectively.

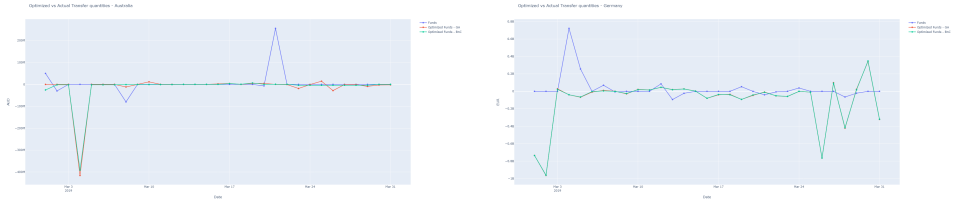
Both algorithms guarantee that cash end in each office is positive and there is no excess of cash (cash end always below $1.5 * safety\_level$). Whereas, with the baseline approach, Japan and United Kingdom offices have 5 days with negative cash end. Moreover, for most of the days in the time horizon, all countries have an excess of cash. This is illustrated in Figure 1 for Australia and German branches. The blue line represents the actual (baseline) cash end without any optimization approach. The red line represents the optimized cash end with genetic algorithm and the green line is the optimized cash end with Branch-and-Cut algorithm. From the plots, we see that for both offices the red and green lines are one on top of the other, hence, reinforcing the fact that the optimal solution from both algorithms is very similar. It is the same case for the other offices, their plots are in Figure 2 in Appendix B. All these results can be found in table and graphical formats in Appendix B.

The optimal transfer quantities from both algorithms are also displayed in table and graphical formats in Appendix B. Here, we just present the graphical format in Figure 2 for Australia and German offices. The blue line represents the funds with the baseline approach, the red line is the optimized funds applying genetic algorithm and the green line is the optimized funds applying Branch-and-Cut algorithm. For these two countries, the optimal transfer quantities are far from being similar to the randomly generated funds (baseline approach). In addition, the optimal transfer quantities for both algorithms is very similar. This is a reasonable finding given



(A) Australia (B) Germany

FIGURE 1. Optimized Cash End - Australia, Germany



(A) Australia (B) Germany

FIGURE 2. Optimized Funds - Australia, Germany

the fact that they both output similar results in Tables 1 and 2. We get the same results for the other offices, their plots can be checked in Figure 1 in Appendix B.

From Table 3, we see that the increase in net earnings when using BnC or GA algorithms for Germany office is mainly because the earnings are much greater than its respectively transaction costs. For UK office, it is because the earnings are greater than with the baseline approach and the transaction costs are smaller than with the baseline approach.

Country	Baseline		BnC		GA	
	Earnings	TC	Earnings	TC	Earnings	TC
Australia	61130543.28	180019.38	64038185.73	201180.61	63959957.04	227570.33
Germany	56208220.66	503924.49	84531375.85	1482169.74	84518273.21	1500185.34
India	60907811.07	11808.26	61043902.4	33309.49	6104368.6	36950.08
Japan	60800553.28	34099.97	61140475.6	1315.37	61140467.13	1322.65
UK	56009575.07	409502.65	61818564.44	205297.14	61800198.84	218754.42

TABLE 3. Comparison Table - Earnings vs Transaction Costs

Chapter 6
Conclusions

From the previous section, we have seen that Branch-and-Cut is better than genetic algorithm in terms of execution time. For all offices, cplex implementation only takes a second to execute the MILP model for each office. Whereas, genetic algorithm takes around 3 minutes for each office to execute the MILP model, even with a predefined solution and with multiprocessing module. However, they both generate similar results when maximizing the objective function since the % increase in net earnings for each country is nearly the same, compared to the baseline approach. Therefore, we can state that they are both good optimization algorithms for maximizing net earnings.

In addition, given the random input data, it is clear that both algorithms are a better approach than the approach where the transfer quantities are not optimized (baseline approach). There are considerable reductions in cash held in each office, hence, less idle cash in balances. There is only a minimum of unnecessary cash available in the offices in order to guarantee that they can meet unexpected demands of cash. And the excess of cash in each office is sent to the central office, where it is properly invested.

In fact, GA is a good alternative to BnC in terms of financial aspects. The former is implemented in Python, which is an open source programming language. Whereas the latter is implemented through cplex in AMPL, which is a very expensive mathematical programming language. Then, choosing one algorithm or the other is mainly based on whether you are willing to pay a high price for getting a license of AMPL for a slight improve in maximizing the objective function. Or you rather prefer a free option with very similar results but with slightly longer execution time.

There are also cash flow management software tools available in the web. However, most of them are private solutions which means that you have to pay a subscription in order to use them. Moreover, you are not aware of the underlying mathematical models that these platforms implement in order to optimize the cash management. Besides, these solutions are generic whereas clients may be different in size, in industry, in service, etc. Anaplan or Workday Adaptive Planning are some of the private options that are available for enterprises. Again, for financial reasons, GA is also a better option than these private software tools. Furthermore, our approach is

more flexible because the MILP model can be customized to a specific client given its needs.

This report is a basic approach to the cash management problem for the branch-office optimization with the genetic algorithm implementation. It is similar to the study of Osorio and Toro (2012)[**22**] but without including an ATM network. We believe that the findings from this analysis would be even more interesting if we used real world data as well as safety cash levels based on financial strategies. For instance, we could use data from a given client. Up to now, the input data as well as the safety cash levels are generated randomly. Yet, we tried to generate data as realistic as possible. We used five offices located in different parts of the world that use different currencies, each of them with their own parameters. In addition, we introduced the concept of spread cost in order to compute the variable transaction cost. Last, we defined earnings as the compound interest from holding cash in balance.

Genetic algorithm is a very popular algorithm implemented in supply chain problems. The CMP can be seen as a supply chain problem where cash is the inventory, that should be kept in balance at some desired level. Therefore, we believe that GA is a good option for solving this problem. Moreover, this report is one of the few studies that approach the cash management problem with genetic algorithm.

In relation to GA implementation, perhaps, it could be improved if we applied an algorithm to tune the configuration parameters such as the population size, mutation sigma, etc. Instead, we manually selected the values for the configuration parameters. In this process, we observed that the population size is related to diversity. For instance, GA was not able to find a local optimum different from the predefined individual for Germany office if the population size is below 400.

Tuning the parameters with an algorithm might speed up the computational time of executing the genetic algorithm if there were more appropriate values for these parameters. On the other hand, it would be interesting to analyse other scenarios where we apply features that we have not used until now. It would be of interest to analyse the effect of keeping a % of the parent population for the next generation instead of replacing the entire parent population with the child population.

This mixed integer linear programming model that we formulated for the CMP, can be expanded into more complex forms. As commented before, it can be adapted to the specific needs of a given client. For example, we could introduce the concept of lead days, where cash transfers are not instant. But instead, the transfers generated in a given day are received after some days. Another feature that could be added in the model is to allow for interactions between branch offices. Now, they are assumed to be independent and, therefore, we decided to execute the algorithm separately for each office. For further studies, we could also relax the assumption that the central office has unlimited funds.

In conclusion, implementing genetic algorithm in Python is a good and free alternative to Branch-and-Cut algorithm and CMP private software tools. As long as we introduce a predefined solution and the multiprocessing module in the execution of GA, it is as suitable as BnC algorithm.

Appendix A
Code

1. AMPL Code

1.1. .dat files.

cash_optimizer_1.dat file

```
param D:=31;

#transaction cost variables
param CFCO:=20;
param CFOC:=10;
param FXspread:=0.0006;

param r0:= 0.01;
param rC:=0.12;

param start0:=105403072;
param startC:=6000000000;

param ss:= 100000000;

param ub:=500000000;
param lb:=0;

param bigM:=1000000000;

param demand:=
1         20097588.5
2          2097588.5
3          2097588.5
4        390975808.5
5          2097588.5
6          2097588.5
7          2098225.68
8           963573.1
9           963573.1
10          963573.1
```

```
11          963573.1
12          963573.1
13          963573.1
14         1337096.8
15          963573.1
16         -627832.01
17        -6278320.01
18         -627832.01
19        -6270832.01
20         -627832.01
21         -627832.01
22         -627832.01
23         4030850.81
24         4030850.81
25         4030850.81
26         4030850.81
27         4030850.81
28         4030850.81
29         4030850.81
30         4030850.81
31         4030850.81;

param FXspot:=
1         1.412831145
2         1.412831145
3         1.412831145
4         1.411835214
5         1.414227812
6         1.423677479
7         1.42651675
8         1.419037436
9         1.420244962
10        1.420244962
11        1.415227125
12        1.414028119
13        1.41044327
14        1.416227852
15        1.41123833
16        1.41123833
17        1.41123833
18        1.410244645
19        1.411437236
20        1.40391807
21        1.406086413
22        1.4130305
23        1.4130305
24        1.4130305
25        1.407271973
26        1.401952648
27        1.412432604
28        1.413229911
29        1.412233418
```

```
30        1.412233418
31        1.412233418;
```

cash_optimizer_2.dat file

```
param D:=31;

#transaction cost variables
param CFCO:=20;
param CFOC:=10;
param FXspread:=0.0003;

param rO:=0.02;
param rC:=0.12;

param startO:=161206928;
param startC:=6000000000;

param ss:=10000000;

param ub:=1000000000;
param lb:=0;

param bigM:=10000000000;

param demand:=
1          585255590
2          966977790
3          -27459788
4           39261666
5           65041444
6            8408944
7           -9856834
8             444834
9           30173321
10         -24228238
11         -15125254
12         -46270901
13         -18651453
14         -26753345
15          -2001000
16          79736090
17          36448589
18          39065678
19          92636453
20          42146654
21          11142453
22          50795567
23          58646456
24           -531444
25           9407643
26         769783111
```

```
27          -98795211
28          422292444
29          -22333342
30         -348055656
31          324985677;

param FXspot:=
1          0.88046
2          0.88046
3          0.88046
4          0.88108
5          0.88372
6          0.88611
7          0.88544
8          0.89381
9          0.89381
10         0.89381
11         0.89119
12         0.88973
13         0.88728
14         0.88454
15         0.88454
16         0.88454
17         0.88454
18         0.88306
19         0.88229
20         0.88236
21         0.87619
22         0.87992
23         0.87992
24         0.87992
25         0.88615
26         0.88501
27         0.88929
28         0.88949
29         0.89151
30         0.89151
31         0.89151;
```

cash_optimizer_3.dat file

```
param D:=31;

#transaction cost variables
param CFCO:=20;
param CFOC:=10;
param FXspread:=0.0004;

param r0:= 0.02;
param rC:=0.12;

param start0:=713893980;
```

```
param startC:=6000000000;

param ss:= 100000000;

param ub:=800000000;
param lb:=0;

param bigM:=1000000000;

param demand:=
1           90726681.7
2          190711444.4
3         -190711444.4
4         -190711444.4
5          190711444.4
6         -190711444.4
7          190711444.4
8         -302207808.1
9         -302207808.1
10        -302207808.1
11        -302207808.1
12         302207808.1
13        -302207808.1
14        -302207808.1
15        -302207808.1
16         -79776735.03
17         -86725481.03
18         -49455598.03
19         -85278584.03
20         -88172378.03
21          86725481.03
22          86725481.03
23        -214520298
24         145202984
25        -202217237
26        -214521315
27         145202984
28         145202984
29        -214520298
30        -213771630
31         145202984;

param FXspot:=
1         70.89475
2         70.89475
3         70.89475
4         70.91225
5         70.801
6         70.54225
7         70.03475
8         70.091
9         70.091
10        70.091
```

```
11        69.886
12        69.62975
13        69.6935
14        69.591
15        69.29475
16        69.29475
17        69.29475
18        68.85225
19        68.471
20        69.07475
21        68.8335
22        68.6085
23        68.6085
24        68.6085
25        69.0535
26        68.8435
27        68.8785
28        69.02725
29        69.0785
30        69.0785
31        69.0785;
```

cash_optimizer_5.dat file

```
param D:=31;

#transaction cost variables
param CFCO:=20;
param CFOC:=10;
param FXspread:=0.0004;

param r0:= 0.01;
param rC:=0.12;

param start0:=44457120;
param startC:=6000000000;

param ss:= 10000000;

param ub:=200000000;
param lb:=0;

param bigM:=1000000000;

param demand:=
1               35534250.16
2                3591781.51
3                3591781.51
4                3714235.51
5                3631860.12
6                3604617.45
7                3597810.4
```

```
8                  1341541.17
9                  1341154.39
10                 1341154.39
11                 1347025.51
12                  -90675.78
13                 1341154.39
14                 1342237.21
15                 1371778.33
16                  543928.68
17                 5439288.68
18                  597509.96
19              -123397790.5
20                16428709.6
21                 5525691.64
22                33293218.28
23                -3330932.93
24                -3330932.93
25                -3322930.51
26               -33305576.26
27                -3330549.6
28               -33243811.42
29                -5311621.15
30               -33309322.93
31                -3330932.93;


param FXspot:=
1         0.75543
2         0.75543
3         0.75543
4         0.75688
5         0.76082
6         0.76223
7         0.75963
8         0.76508
9         0.76508
10        0.76508
11        0.7715
12        0.75828
13        0.7652
14        0.75622
15        0.75605
16        0.75605
17        0.75605
18        0.75353
19        0.75426
20        0.7554
21        0.75771
22        0.76241
23        0.76241
24        0.76241
25        0.75912
26        0.75946
```

```
27        0.75949
28        0.75892
29        0.76608
30        0.76608
31        0.76608;
```

2. Python Code

2.1. Configuration files.

ga_config_Australia.json

```
{
  "config_ga": {
            "random_seed": 64,
            "population_size": 300,
            "generations": 50,
            "force_end_generations": 60,
            "convergence_generations": 30,
            "sel_size": 0.3,
            "mut_integer": "gaussian",
            "mut_sigma": 0.1,
            "mut_prob": 1,
            "cross_prob": 0.9,
            "mut_start_generations": 10,
            "mut_start_weight": 0.5,
            "mut_local_generations": 30,
            "mut_local_optimal_weight": 1,
            "multiprocessing": true,
            "num_processors": 2
  },
    "optimization_range": [-500000000,500000000],
    "constraint_penalty": 1000000,
    "previous_solutions": [[0, 0, 0, -410000000, 0, 0, 0,
                         11700079, 0, 11599200, 0, 0, 0, 0, 0,
                         2209488, 4177168, 0, 4477168, 4377168,
                         0, 0, -18819800, -900009, -709999,
                        -857667, -322052, -1116568, -9220000,
                        -424010, -9888880]],
  "objectives": {
            "net_earnings": "max"
  },
  "output": {
            "write_outputs": 1,
            "output_path": "./ga_output/Australia/"
```

```
  }
}
```

ga_config_Germany.json

```
{
  "config_ga": {
            "random_seed": 64,
            "population_size": 400,
            "generations": 50,
            "force_end_generations": 60,
            "convergence_generations": 30,
            "sel_size": 0.3,
            "mut_integer": "gaussian",
            "mut_sigma": 0.1,
            "mut_prob": 1,
            "cross_prob": 0.9,
            "mut_start_generations": 10,
            "mut_start_weight": 0.5,
            "mut_local_generations": 30,
            "mut_local_optimal_weight": 1,
            "multiprocessing": true,
            "num_processors": 2
  },
    "optimization_range": [-1000000000,1000000000],
    "constraint_penalty": 1000000,
    "previous_solutions": [[-735400000, -965227900, 29022790,
                -40022790, -68032891, -9023999, 10022790,
                1022480, -29022000, 21933440, 15933390,
                45890000, 19890000, 27905600, 2900000,
                -80094880, -39771680, -34781799, -92599716,
                -46771680, -7771680, -52477168, -58819800,
                -1000009, -9099990, -765766700, 99220520,
                -422965680, 22922000, 343401000, -322888000]],
  "objectives": {
            "net_earnings": "max"
  },
  "output": {
            "write_outputs": 1,
            "output_path": "./ga_output/Germany/"
  }
}
```

ga_config_India.json

```
{
  "config_ga": {
            "random_seed": 64,
```

```
            "population_size": 300,
            "generations": 50,
            "force_end_generations": 60,
            "convergence_generations": 30,
            "sel_size": 0.3,
            "mut_integer": "gaussian",
            "mut_sigma": 0.1,
            "mut_prob": 1,
            "cross_prob": 0.9,
            "mut_start_generations": 10,
            "mut_start_weight": 0.5,
            "mut_local_generations": 30,
            "mut_local_optimal_weight": 1,
            "multiprocessing": true,
            "num_processors": 2
  },
    "optimization_range": [-800000000,800000000],
    "constraint_penalty": 1000000,
    "previous_solutions": [[-680000000, -214000000, 200227900,
                            200002279, -200002279, 200227900,
                            -200002279, 300897999, 300227909,
                            300537810, 300227770, -300227650,
                            300827900, 300227901, 300956900,
                            90227650, 80346654, 60445654,
                            80524650, 80524650, -65246500,
                            -95246481, 219804300, -119008043,
                            154000000, 240000000, -122205200,
                            -155329522, 219220000, 199988517,
                            -169088880]],
  "objectives": {
            "net_earnings": "max"
  },
  "output": {
            "write_outputs": 1,
            "output_path": "./ga_output/India/"
  }
}
```

ga_config_UK.json

```
{
  "config_ga": {
            "random_seed": 64,
            "population_size": 300,
            "generations": 50,
            "force_end_generations": 60,
            "convergence_generations": 30,
            "sel_size": 0.3,
```

```
            "mut_integer": "gaussian",
            "mut_sigma": 0.1,
            "mut_prob": 1,
            "cross_prob": 0.9,
            "mut_start_generations": 10,
            "mut_start_weight": 0.5,
            "mut_local_generations": 30,
            "mut_local_optimal_weight": 1,
            "multiprocessing": true,
            "num_processors": 2
  },
    "optimization_range": [-200000000,200000000],
    "constraint_penalty": 1000000,
    "previous_solutions": [[-69040000, -4002279, -2002279,
                            -3002279, -3002279, -4002279,
                            -5002279, -1700079, -2602200,
                            -999200, -1593339, -43589,
                            -900696, -219056, -2900000,
                            2209488, -4177168, -4277168,
                            125997168, -17377168, -3777168,
                            -32477168, -988198, 8000009,
                            -709999, 35576670, 3322052,
                            31116568, 8922000, 33240100,
                            -988888]],
  "objectives": {
            "net_earnings": "max"
  },
  "output": {
            "write_outputs": 1,
            "output_path": "./ga_output/UK/"
  }
}
```

2.2. GA Initialization. Below is the function that initializes ga custom package. The input parameter evaluation_parameters is a dictionary that contains the input data as in .dat files in AMPL.

For a given country, attributes dict must be specified in the following way: attributes_dict = {"transfer_Q1": ["integer", min, max], "transfer_Q2": ["integer", min, max], ..., transfer_Q31": ["integer", min, max]}.

If we are interested in introducing predefined solutions to the initial population, it must be input in the form of sub-lists in a list: previous_solutions = [ind1, ind2, ...] where ind1 = [transfer_1, transfer_2, ...]. For example, if we were to optimize only 5 days, and we want to input one predefined solution, it would be as follows: previous_solutions = [[100, -100, 0, 0, 40]] where ind1 = [100, -100, 0, 0, 40].

The evaluation function is a custom function that evaluates the objective function for a given set of transfer quantities. The parameter indpb in mutation function refers to the mutation rate.

```
def configure_genetic_algorithm(self, ga_config, evaluation_parameters):

    # Create attributes dict
    attributes_dict = {}
    for i in range(self.decisions_to_optimize):
        attributes_dict["transfer_Q{0}".format(i + 1)] = ["integer",
            ga_config["optimization_range"][0],
            ga_config["optimization_range"][1]]

    # GA initialization
    ga = CustomGeneticAlgorithm()

    params_ga = ga_config['config_ga']
    ga.set_params(**params_ga)

    if self.previous_solution:
        previous_solutions = ga_config["previous_solutions"]
        print("previous_solution is True")
    else:
        previous_solutions = []
        print("previous_solution is False")

    # Configure GA
    ga.configure_ga(attributes_dict=attributes_dict,
        objectives=ga_config['objectives'],
                  previous_solution=previous_solutions)

    # Define custom evaluation function
    ev_func = Evaluation(attributes_dict, evaluation_parameters)
    ga.register_function(alias='evaluate', function=ev_func.evaluate)

    # Mutate function
    num_genes = ga.num_genes
    mut_func = Mutation()
    ga.register_function(alias='mutate', function=mut_func.mutate,
        attributes=attributes_dict,
                       mut_integer='gaussian', mu=0, sigma=0.1, indpb=1 /
                           num_genes)

    # not specify selection function -> by default use tools.selTournament
    # not specify cross-over (mate) function -> by default use
        tools.cxTwoPoint

    # Custom user constraint included.
    ct = PositiveBalance(name="cash_positive_balance",
        attribute_dict=attributes_dict,
                       evaluation_parameters=evaluation_parameters)
    ga.register_constraints(constraint_dict={},
        constraint_user_dict={'constraint_user': ct},
```

```
                        constraint_penalty=ga_config["constraint_penalty"],
                        attributes_dict=attributes_dict,
                            objectives=ga_config['objectives'])

    return ga
```

2.3. GA Structure. Figure 1 shows the basic structure of how to execute ga custom package. In our case, we registered all the optional modules, except for local search. A local search is a greedy technique that can be used to converge to local optimums. When used within a Genetic Algorithm, the local search can be used before the selection step of each generation to create one additional individual that is created from the best parts of all other individuals. The idea is to create a custom individual from the best partial solutions within each chromosome. The resulting solution might not be better than the existing ones, but constitutes a greedy approximation of where the local minimum should be located. However, there is no need of this module if we are already using a predefined individual and the multiprocessing module.

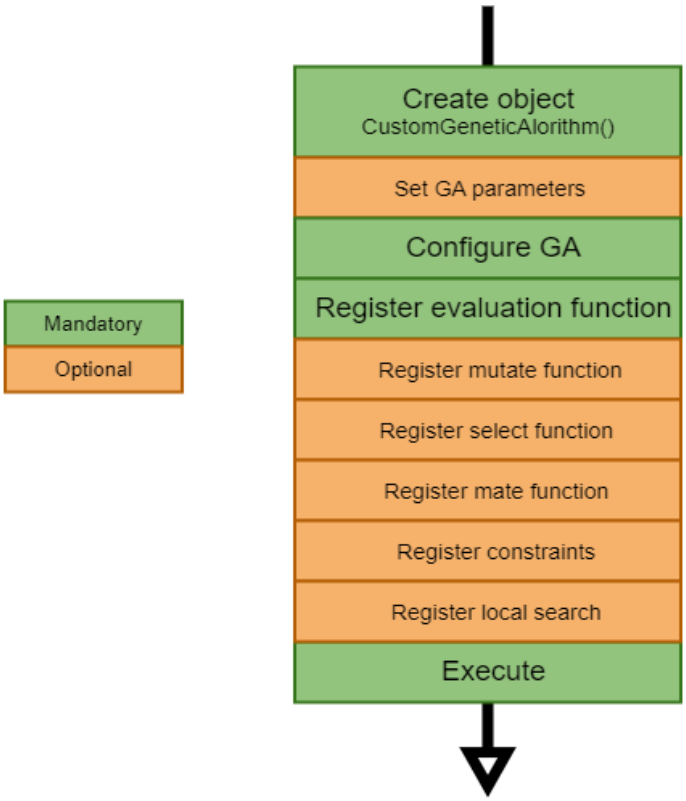


FIGURE 1. GA Structure

The mutation function is specific for integer variables and only applies Gaussian mutation.

```
class Mutation:

    def mutate(self, individual, attributes, mut_integer, mu, sigma, indpb):
        """Mutate an individual by replacing attributes, with probability
            *indpb*, by a integer uniformly drawn between *low* and *up*
            inclusively.
        :param individual: individual to be mutated.
        :param attributes: the details of each attribute.
        :param indpb: Independent probability for each attribute to be
            mutated.
        :returns: A tuple of one individual.
        """
        size = len(individual)
        gaussian_int = mut_integer.upper() == 'GAUSSIAN'
        p = indpb

        for i in range(size):
            if random.random() <= p:
                attr = i % len(attributes)
                attr_name = list(attributes.keys())[attr]
                if attributes[attr_name][0].upper() == 'INTEGER':
                    if gaussian_int:
                        individual[i] += round(random.gauss(mu, sigma_v))
                        # Guarantee that the mutated value is inside limits
                        if individual[i] < min_val:
                            individual[i] = min_val
                        elif individual[i] > max_val:
                            individual[i] = max_val

        return individual,
```

The class PositiveBalance checks whether an individual is feasible or not. In addition, if it is not feasible, it computes its distance, how far it is from being feasible.

```
class ConstraintUser(ga_constraint.Constraint):
    def __init__(self, name, attributes, evaluation_parameters):
        super().__init__(name, attributes, evaluation_parameters)

    @abstractmethod
    def feasible(self, individual):
        pass

    @abstractmethod
    def distance(self, individual):
        pass

class PositiveBalance(ga_constraint_user.ConstraintUser):
    def __init__(self, name, attributes_dict, evaluation_parameters):
        super().__init__(name, attributes_dict, evaluation_parameters)
```

```
def feasible(self, individual):

    """
    some code
    """

    for i in range(0, len(individual)):
        # iterate until find one day where x < ss or x > 1.5*ss
        if summary["optimized_cash_end"][i] <
            self.evaluation_parameters["safety_stock"]:
            return False
        elif summary["optimized_cash_end"][i] >
            self.evaluation_parameters["safety_stock"] * 1.5:
            return False
    return True

def distance(self, individual):

    distance = 0
    for i in range(0, len(individual)):
        if individual.summary["optimized_cash_end"][i] <
            self.evaluation_parameters["safety_stock"]:
            distance += abs(
                individual.summary["optimized_cash_end"][i]
                - self.evaluation_parameters["safety_stock"]
            )
        elif individual.summary["optimized_cash_end"][i] >
            self.evaluation_parameters["safety_stock"] * 1.5:
            distance += abs(
                self.evaluation_parameters["safety_stock"] * 1.5
                - individual.summary["optimized_cash_end"][i]
            )

    return distance
```

Appendix B
Additional Results

1. Actual vs Optimized Funds

Generated funds for baseline approach for each country in Table 1.

date	Australia	Germany	India	Japan	UK
01/03/2019	50000000,00	0,00	0,00	0,00	-50000000,00
02/03/2019	-30000000,00	0,00	0,00	0,00	-50000000,00
03/03/2019	0,00	0,00	2000000000,00	-5000000,00	0,00
04/03/2019	0,00	722204898,00	15544000,00	-5000000,00	0,00
05/03/2019	0,00	256265556,00	0,00	0,00	0,00
06/03/2019	0,00	0,00	0,00	1000000000,00	0,00
07/03/2019	0,00	69551222,00	0,00	0,00	0,00
08/03/2019	-80000000,00	-6426066,00	0,00	0,00	677999453,00
09/03/2019	0,00	0,00	0,00	0,00	2518900,00
10/03/2019	0,00	-571989,00	0,00	0,00	0,00
11/03/2019	0,00	0,00	0,00	0,00	0,00
12/03/2019	0,00	85824333,00	33440989,00	0,00	0,00
13/03/2019	0,00	-94165155,00	0,00	0,00	0,00
14/03/2019	0,00	-22634221,00	0,00	0,00	0,00
15/03/2019	0,00	0,00	0,00	6000000000,00	0,00
16/03/2019	0,00	0,00	0,00	0,00	0,00
17/03/2019	0,00	0,00	0,00	0,00	0,00
18/03/2019	0,00	0,00	0,00	0,00	0,00
19/03/2019	0,00	53331717,00	0,00	-21000000,00	0,00
20/03/2019	-7000000,00	-571818,00	8000000,00	-11000000,00	-1000000,00
21/03/2019	255555500,00	-40165556,00	-8000000,00	0,00	0,00
22/03/2019	0,00	-6141555,00	-8000000,00	0,00	0,00
23/03/2019	0,00	0,00	0,00	0,00	0,00
24/03/2019	0,00	38353588,00	0,00	0,00	0,00
25/03/2019	0,00	0,00	0,00	0,00	0,00
26/03/2019	0,00	0,00	0,00	0,00	0,00
27/03/2019	0,00	0,00	0,00	0,00	0,00
28/03/2019	0,00	-64941699,00	0,00	543444876,00	0,00
29/03/2019	0,00	-21224787,00	0,00	0,00	0,00
30/03/2019	0,00	0,00	0,00	0,00	0,00
31/03/2019	0,00	0,00	0,00	0,00	0,00

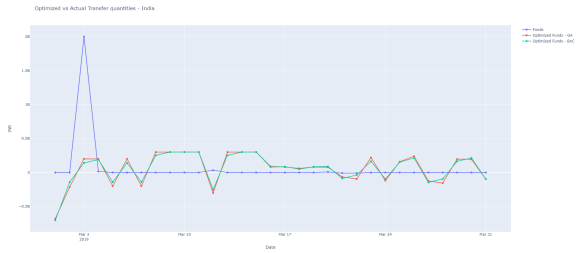
TABLE 1. Countries' Actual Funds

date	Australia	Germany	India	Japan	UK
01/03/2019	-98170	-735400000	-680000000	7940000	-69040000
02/03/2019	0	-965227900	-214000000	8002279	-4002279
03/03/2019	-78742	29022790	200227900	8002279	-2002279
04/03/2019	-415733517	-40022790	200002279	8002279	-3002279
05/03/2019	0	-68032891	-200145026	7993357	-3002279
06/03/2019	0	-9023999	200227900	8002279	-4002279
07/03/2019	0	10022790	-200002279	8002279	-5002279
08/03/2019	-11925076	1022480	300897999	11700079	-1700079
09/03/2019	0	-29022000	300227909	11602200	-2602200
10/03/2019	11599200	21933440	300537810	11599200	-999200
11/03/2019	0	15875920	300227770	11593339	-1593339
12/03/2019	0	45890000	-300553588	11243589	-43589
13/03/2019	0	19890000	300827900	11900696	-900696
14/03/2019	0	27905600	300227901	11219056	-219056
15/03/2019	0	2900000	300956900	-6900000	-2900000
16/03/2019	2209488	-80094880	90227650	2209488	2209488
17/03/2019	4177168	-39771680	80346654	4177168	-4177168
18/03/2019	0	-34781799	60445654	4277168	-4277168
19/03/2019	4477168	-92599716	80524650	4477168	125997168
20/03/2019	4377168	-46771680	77102950	4377168	-17377168
21/03/2019	0	-7771680	-65246500	3777168	-3916739
22/03/2019	-707828	-52477168	-95246481	4477168	-32477168
23/03/2019	-18819800	-58819800	219804300	-988198	-988198
24/03/2019	-900009	-1000009	-119008043	-900009	8000009
25/03/2019	14456201	-9099990	157635139	-709999	-709999
26/03/2019	-28454857	-765766700	240000000	-857667	35576670
27/03/2019	-839257	99220520	-124213347	-322052	3322052
28/03/2019	-1116568	-422965680	-156302692	-1116568	31116568
29/03/2019	-9220000	22922000	197834751	-922000	8922000
30/03/2019	-2943260	343401000	195346845	-642660	33240100
31/03/2019	-2319709	-322390350	-96894079	-804234	-250273

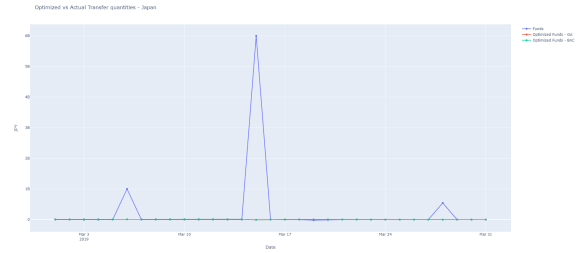
TABLE 2. Countries' Optimized Funds - GA

date	Australia	Germany	India	Japan	UK
01/03/2019	-25500660	-736462518	-704620660	7520928	-69991370
02/03/2019	-2097589	-961977790	-140711447	8021355	-3591781
03/03/2019	-2097588	22459788	140711447	8021356	-3591782
04/03/2019	-390975809	-39261666	190711444	8021356	-3714235
05/03/2019	-2097588	-65041444	-140711446	8021356	-3631860
06/03/2019	-2097588	-3408944	140711445	8021356	-3604617
07/03/2019	-2098226	4856834	-140711445	8021356	-3597811
08/03/2019	-963573	-444834	252207810	11576594	-1341541
09/03/2019	-963573	-25173321	302207808	11576593	-1341154
10/03/2019	-963573	19228238	302207808	11576594	-1341154
11/03/2019	-963573	15125254	302207808	11576593	-1256350
12/03/2019	-963573	46270901	-252207810	11576594	0
13/03/2019	0	18651453	252207809	11576593	-1341154
14/03/2019	0	26753345	302207808	11576594	-1342238
15/03/2019	0	2001000	302207809	-6318425	-1371779
16/03/2019	0	-79736090	79776735	2188840	-543928
17/03/2019	3641909	-36448589	86725481	3718885	-1036800
18/03/2019	627832	-39065678	49455598	4718884	0
19/03/2019	6270832	-92636453	85278584	3718885	118397791
20/03/2019	627832	-42146654	88172378	4718883	-16428709
21/03/2019	627832	-11142453	-86725481	4218884	-5525692
22/03/2019	627832	-50795567	-36725483	3718885	-28293219
23/03/2019	-4030850	-58115012	164520299	-291065	0
24/03/2019	-4030851	0	-95202985	-1291063	1661867
25/03/2019	-4030851	-9407643	152217238	-791064	3322931
26/03/2019	-4030851	-764783111	214521315	-291065	33305577
27/03/2019	-4030851	93795211	-145202984	-1291063	3330549
28/03/2019	-4030851	-417292444	-95202985	-291065	33243811
29/03/2019	-4030851	17333342	164520299	-791064	5311621
30/03/2019	0	348055656	213771630	-1291064	33309323
31/03/2019	0	-319985677	-95202985	-791064	3330933

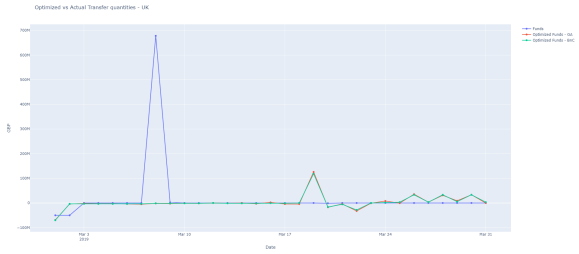
TABLE 3. Countries' Optimized Funds - BnC



(A) India



(B) Japan



(C) UK

FIGURE 1. Optimized Funds - India, Japan, UK

Figure 1 only shows the optimal values for the transfer quantities in order to maximize net earnings for India, Japan, UK offices. The blue line is the funds with the baseline approach, the red line the optimized funds applying genetic algorithm and green line the optimized funds applying Branch-and-Cut algorithm. For all countries, the optimized transfer quantities from both algorithms are very similar.

2. Countries Cash End

The following tables aggregate the cash end held in each country for the three approaches, baseline approach, genetic algorithm and Branch-and-Cut algorithm.

date	cash_end baseline	cash_end GA	cash end BnC
01/03/2019	175500660.5	125402490.5	100000000.5
02/03/2019	147598249	127500079	100000000
03/03/2019	149695837.5	129518925.5	100000000.5
04/03/2019	540671646	104761217	100000000
05/03/2019	542769234.5	106858805.5	100000000.5
06/03/2019	544866823	108956394	100000001
07/03/2019	546965048.7	111054619.7	100000000.7
08/03/2019	467928621.8	100093116.8	100000000.8
09/03/2019	468892194.9	101056689.9	100000000.9
10/03/2019	469855768	113619463	100000001
11/03/2019	470819341.1	114583036.1	100000001.1
12/03/2019	471782914.2	115546609.2	100000001.2
13/03/2019	472746487.3	116510182.3	100963574.3
14/03/2019	474083584.1	117847279.1	102300671.1
15/03/2019	475047157.2	118810852.2	103264244.2
16/03/2019	474419325.2	120392508.2	102636412.2
17/03/2019	468141005.2	118291356.2	100000001.2
18/03/2019	467513173.2	117663524.2	100000001.2
19/03/2019	461242341.1	115869860.1	100000001.1
20/03/2019	453614509.1	119619196.1	100000001.1
21/03/2019	708542177.1	118991364.1	100000001.1
22/03/2019	707914345.1	117655704.1	100000001.1
23/03/2019	711945195.9	102866754.9	100000001.9
24/03/2019	715976046.7	105997596.7	100000001.7
25/03/2019	720006897.5	124484648.5	100000001.5
26/03/2019	724037748.4	100060642.4	100000001.4
27/03/2019	728068599.2	103252236.2	100000001.2
28/03/2019	732099450	106166519	100000001
29/03/2019	736130300.8	100977369.8	100000000.8
30/03/2019	740161151.6	102064960.6	104030851.6
31/03/2019	744192002.4	103776102.4	108061702.4

TABLE 4. Australia's cash end

date	cash_end baseline	cash_end - GA	cash end - BnC
01/03/2019	746462518	11062518	10000000
02/03/2019	1713440308	12812408	15000000
03/03/2019	1685980520	14375410	10000000
04/03/2019	2447447084	13614286	10000000
05/03/2019	2768754084	10622839	10000000
06/03/2019	2777163028	10007784	15000000
07/03/2019	2836857416	10173740	10000000
08/03/2019	2830876184	11641054	10000000
09/03/2019	2861049505	12792375	15000000
10/03/2019	2836249278	10497577	10000000
11/03/2019	2821124024	11248243	10000000
12/03/2019	2860677456	10867342	10000000
13/03/2019	2747860848	12105889	10000000
14/03/2019	2698473282	13258144	10000000
15/03/2019	2696472282	14157144	10000000
16/03/2019	2776208372	13798354	10000000
17/03/2019	2812656961	10475263	10000000
18/03/2019	2851722639	14759142	10000000
19/03/2019	2997690809	14795879	10000000
20/03/2019	3039265645	10170853	10000000
21/03/2019	3010242542	13541626	10000000
22/03/2019	3054896554	11860025	10000000
23/03/2019	3113543010	11686681	10531444
24/03/2019	3151365154	10155228	10000000
25/03/2019	3160772797	10462881	10000000
26/03/2019	3930555908	14479292	15000000
27/03/2019	3831760697	14904601	10000000
28/03/2019	4189111442	14231365	15000000
29/03/2019	4145553313	14820023	10000000
30/03/2019	3797497657	10165367	10000000
31/03/2019	4122483334	12760694	15000000

TABLE 5. Germany's cash end

date	cash_end baseline	cash_end - GA	cash end - BnC
01/03/2019	804620661.7	124620661.7	100000001.7
02/03/2019	995332106.1	101332106.1	149999999.1
03/03/2019	2804620662	110848561.7	100000001.7
04/03/2019	2629453217	120139396.3	100000001.3
05/03/2019	2820164662	110705814.7	149999999.7
06/03/2019	2629453217	120222270.3	100000000.3
07/03/2019	2820164662	110931435.7	149999999.7
08/03/2019	2517956854	109621626.6	100000001.6
09/03/2019	2215749046	107641727.5	100000001.5
10/03/2019	1913541237	105971729.4	100000001.4
11/03/2019	1611333429	103991691.3	100000001.3
12/03/2019	1946982226	105645911.4	149999999.4
13/03/2019	1644774418	104266003.3	100000000.3
14/03/2019	1342566610	102286096.2	100000000.2
15/03/2019	1040358802	101035188.1	100000001.1
16/03/2019	960582067.1	111486103.1	100000001.1
17/03/2019	873856586	105107276	100000001
18/03/2019	824400988	116097332	100000001
19/03/2019	739122404	111343398	100000001
20/03/2019	658950026	100273970	100000001
21/03/2019	737675507	121752951	100000001
22/03/2019	816400988	113231951	149999999
23/03/2019	601880690	118515953	100000000
24/03/2019	747083674	144710894	149999999
25/03/2019	544866437	100128796	100000000
26/03/2019	330345122	125607481	100000000
27/03/2019	475548106	146597118	100000000
28/03/2019	620751090	135497410	149999999
29/03/2019	406230792	118811863	100000000
30/03/2019	192459162	100387078	100000000
31/03/2019	337662146	148695983	149999999

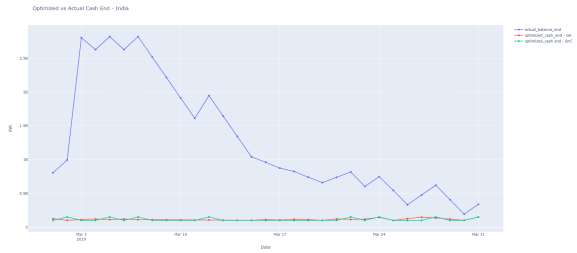
TABLE 6. India's cash end

date	cash_end baseline	cash_end - GA	cash end - BnC
01/03/2019	-6520926.96	1419073.04	1000001.04
02/03/2019	-14542282.92	1399996.08	1000000.08
03/03/2019	-27563638.88	1380919.12	1000000.12
04/03/2019	-40584994.84	1361842.16	1000000.16
05/03/2019	-48606350.8	1333843.2	1000000.2
06/03/2019	943372293.2	1314766.24	1000000.24
07/03/2019	935350937.3	1295689.28	1000000.28
08/03/2019	923774343.8	1419174.8	1000000.8
09/03/2019	912197750.3	1444781.32	1000000.32
10/03/2019	900621156.8	1467387.84	1000000.84
11/03/2019	889044563.4	1484133.36	1000000.36
12/03/2019	877467969.9	1151128.88	1000000.88
13/03/2019	865891376.4	1475231.4	1000000.4
14/03/2019	854314782.9	1117693.92	1000000.92
15/03/2019	6861133206	1036117.44	1499999.44
16/03/2019	6858944366	1056765.24	1499999.24
17/03/2019	6854725482	1015049.04	1000000.04
18/03/2019	6850506598	1073332.84	1499999.84
19/03/2019	6825287714	1331616.64	1000000.64
20/03/2019	6810068829	1489900.44	1499999.44
21/03/2019	6805849945	1048184.24	1499999.24
22/03/2019	6801631061	1306468.04	1000000.04
23/03/2019	6802422125	1109334.17	1499999.17
24/03/2019	6803213189	1000389.3	1000000.3
25/03/2019	6804004253	1081454.43	1000000.43
26/03/2019	6804795318	1014851.56	1499999.56
27/03/2019	6805586382	1483863.69	1000000.69
28/03/2019	7349822322	1158359.82	1499999.82
29/03/2019	7350613386	1027423.95	1499999.95
30/03/2019	7351404450	1175828.08	1000000.08
31/03/2019	7352195514	1162658.21	1000000.21

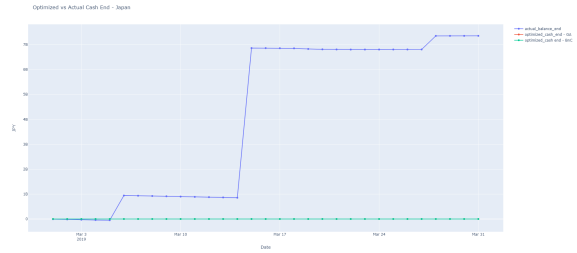
TABLE 7. Japan's cash end

date	cash_end baseline	cash_end - GA	cash end - BnC
01/03/2019	29991370.16	10951370.16	10000000.16
02/03/2019	-16416848.33	10540872.67	10000000.67
03/03/2019	-12825066.82	12130375.18	10000000.18
04/03/2019	-9110831.31	12842331.69	10000000.69
05/03/2019	-5478971.19	13471912.81	10000000.81
06/03/2019	-1874353.74	13074251.26	10000001.26
07/03/2019	1723456.66	11669782.66	10000000.66
08/03/2019	681064450.8	11311244.83	10000000.83
09/03/2019	684924505.2	10050199.22	10000001.22
10/03/2019	686265659.6	10392153.61	10000001.61
11/03/2019	687612685.1	10145840.12	10090677.12
12/03/2019	687522009.3	10011575.34	10000001.34
13/03/2019	688863163.7	10452033.73	10000001.73
14/03/2019	690205400.9	11575214.94	10000000.94
15/03/2019	691577179.3	10046993.27	10000000.27
16/03/2019	692121108	12800409.95	10000000.95
17/03/2019	697560396.6	14062530.63	14402489.63
18/03/2019	698157906.6	10382872.59	14999999.59
19/03/2019	574760116.1	12982250.09	10000000.09
20/03/2019	590188825.7	12033791.69	10000000.69
21/03/2019	595714517.3	13642744.33	10000000.33
22/03/2019	629007735.6	14458794.61	14999999.61
23/03/2019	625676802.7	10139663.68	11669066.68
24/03/2019	622345869.8	14808739.75	10000000.75
25/03/2019	619022939.2	10775810.24	10000001.24
26/03/2019	585717363	13046903.98	10000001.98
27/03/2019	582386813.4	13038406.38	10000001.38
28/03/2019	549143002	10911162.96	10000000.96
29/03/2019	543831380.8	14521541.81	10000000.81
30/03/2019	510522057.9	14452318.88	10000000.88
31/03/2019	507191125	10871112.95	10000000.95

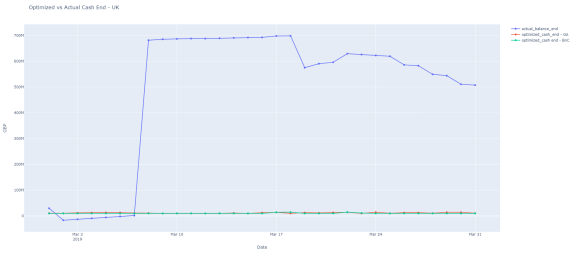
TABLE 8. UK's cash end



(A) India



(B) Japan



(C) UK

FIGURE 2. Optimized Cash End - India, Japan, UK

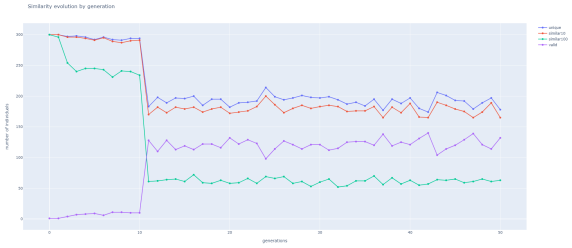
The next Figure 2 graphically illustrate the amount of cash reduced to optimal values when applying an optimization algorithm for India, Japan, UK offices. The blue line represents the actual (baseline) cash end without any optimization approach. The red line represents the optimized cash end with genetic algorithm and the green line is the optimized cash end with Branch-and-Cut algorithm. As for Australia and German offices, the optimal cash end from both algorithms is very similar.

3. GA results

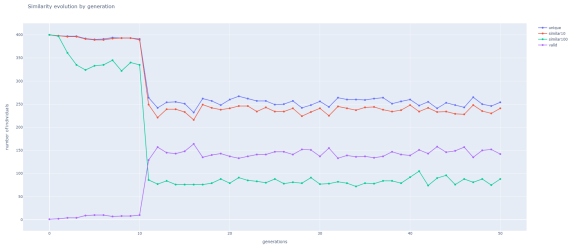
As we can see in subplots in Figure 3, for all countries, after generation 10, the number of unique individuals is approximately around 200. The blue line refers to the number of different individuals that exist for a given generation. The red line counts the number of unique individuals in tens and the green line counts the number of unique individuals in hundreds. Last, the purple line counts the number of feasible individuals that exist in a given generation.

The subplots in Figure 4 for each country shows us that genetic algorithm is converging correctly to a local optimum since the net earnings increases as the number of generations increases. The blue line indicates the best individual found in each generation, hence, it does not guarantee that this individual is feasible. For example, the Subplot 4e for UK office, there were no feasible individuals in generations 2 and 4. Therefore, the best individuals in these generations are those with negative fitness score, but with fitness scores closest to zero. Whereas for the other branches, the best individual in each generation has a positive fitness score because in each generation there is at least one feasible individual.

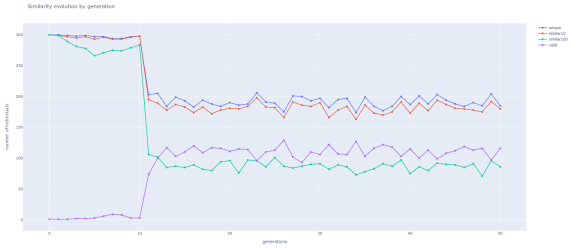
Figure 5 shows the best individual found so far (best individual kept in Hall of Fame) for each country. In Subplot 5b for Germany office, the predefined individual was the best individual found so far for the first 27 generations. Then, in generation 28 the algorithm found an individual with a better score than the predefined individual and it was kept in the Hall of Fame until generation 37. Finally, the algorithm found an even better individual in generation 38 and it was kept in the Hall of Fame until generation 50. Therefore, the individual from generation 38 is the local optimum for Germany office.



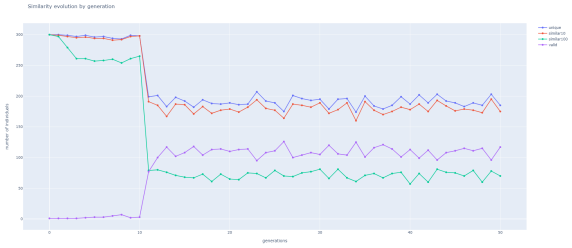
(A) Australia



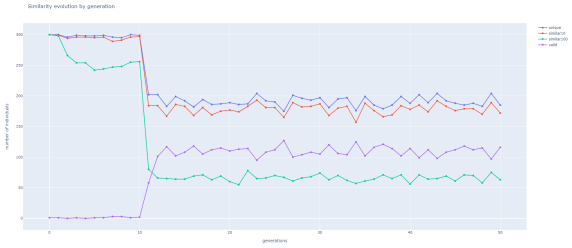
(B) Germany



(C) India

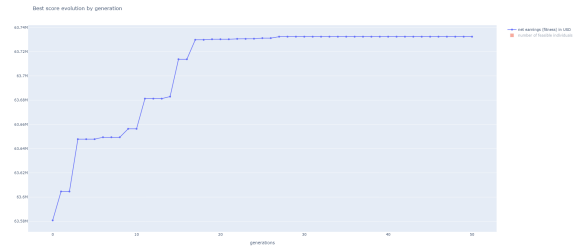


(D) Japan

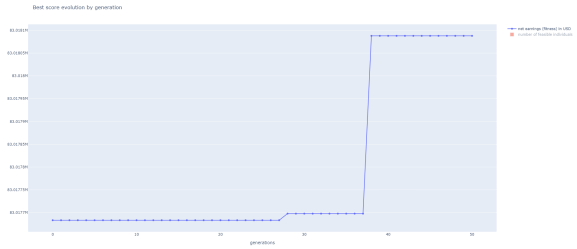


(E) UK

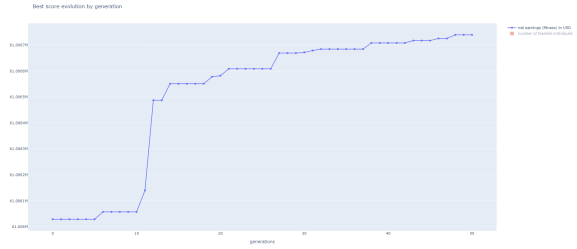
FIGURE 3. Diversity Plot by Country



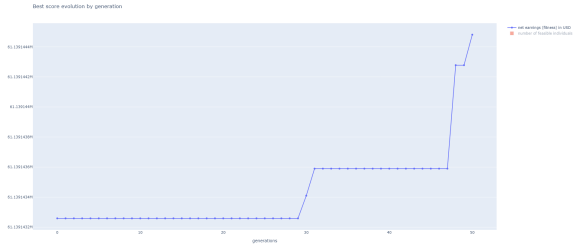
(A) Australia



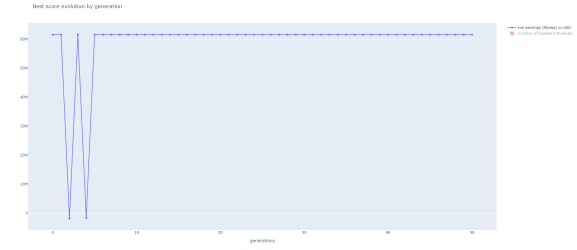
(B) Germany



(C) India

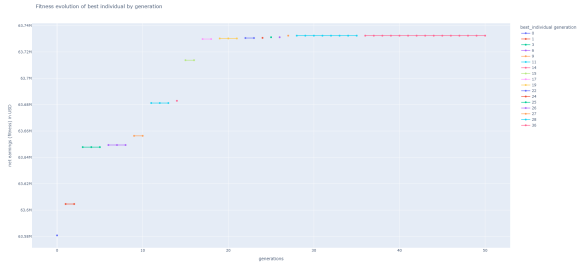


(D) Japan

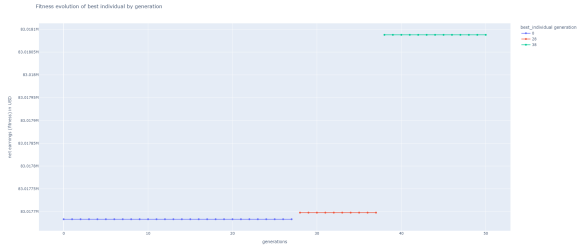


(E) UK

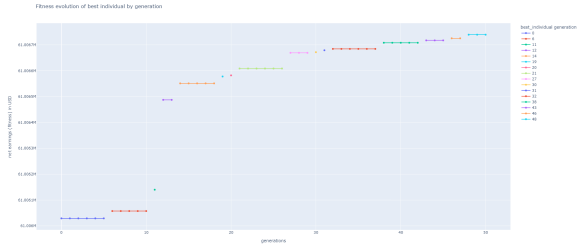
FIGURE 4. Best Individual by Generation and by Country



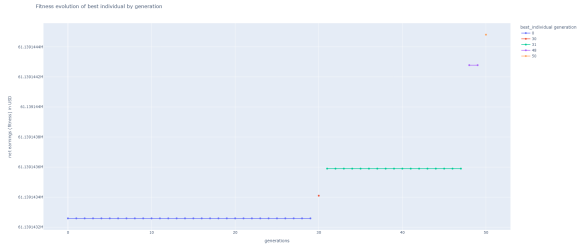
(A) Australia



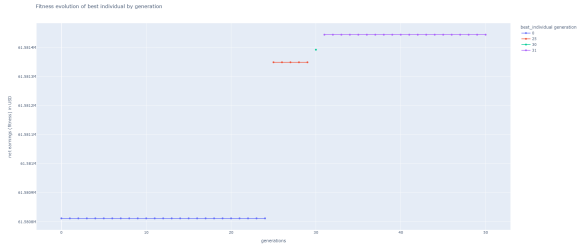
(B) Germany



(C) India



(D) Japan



(E) UK

FIGURE 5. Hall of Fame Evolution by Country

3.1. Germany with Population Size 300. In Figure 6 we see that genetic algorithm was not able to find a local optimum different from the predefined solution for the first 31 generations. However, when the population size is 400, there is more diversity in the population, genetic algorithm is able to find a local optimum different from the predefined solution (Subplot 5b)

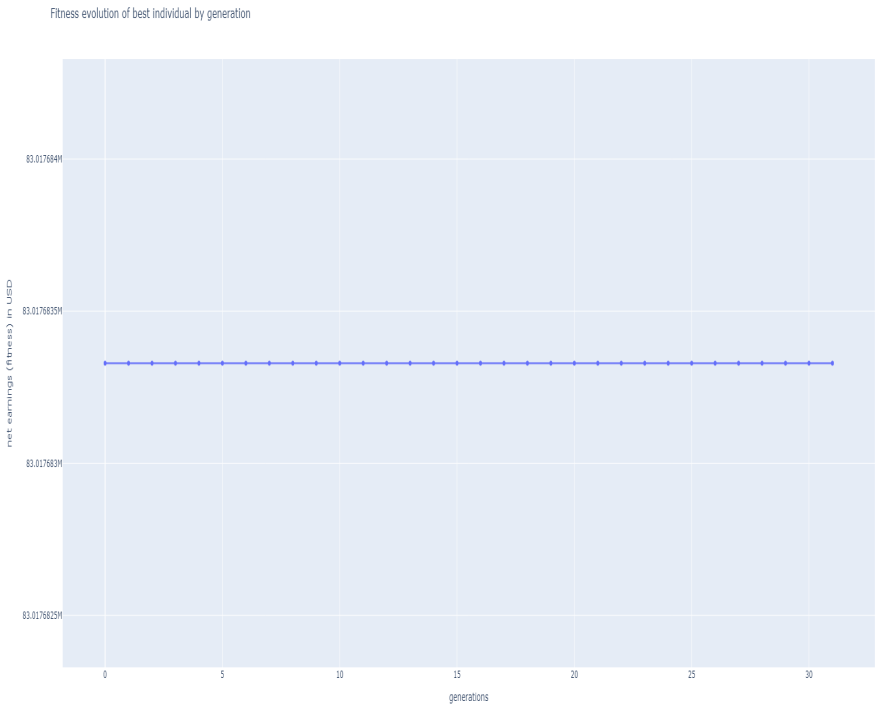
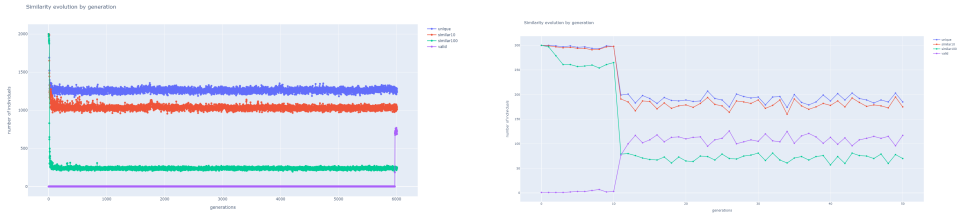


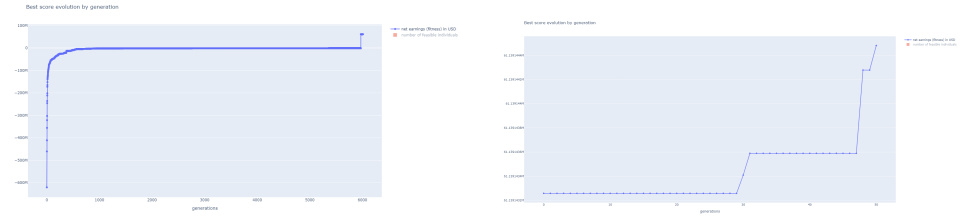
FIGURE 6. Hall of Fame Evolution - Germany with Population Size 300

3.2. GA Execution without predefined solution. In this section we illustrate the results of executing genetic algorithm with multiprocessing module for Japan office without the predefined solution. In this case, we set the parameters "generations", "force_end_generations", "convergence_generations" to 6000, 6010, 30 respectively. The algorithm stop at iteration 6010 with an execution time of 22 hours and net earnings of 61139154.62. From Table 9, we see that the optimized funds with and without the predefined solution are very similar and, therefore, the final net earnings are also roughly the same. The net earnings for the case without predefined solution is 10USD more than for the case with the predefined solution.



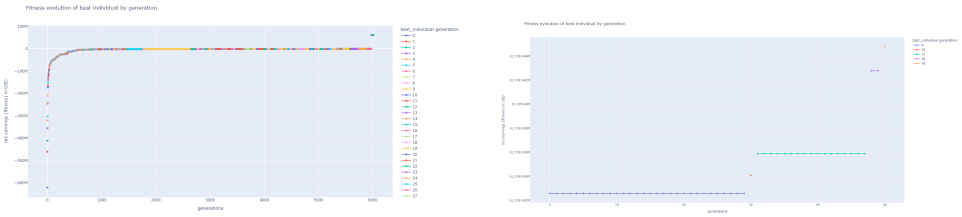
(A) Diversity Plot - Without (B) Diversity Plot - With

FIGURE 7. Diversity Plot - Japan



(A) Best Individuals - Without (B) Best Individuals - With

FIGURE 8. Best Individual Results - Japan



(A) Hall of Fame Evolution - Without (B) Hall of Fame Evolution - With

FIGURE 9. Hall of Fame Evolution - Japan

date	optimized funds - without	optimized funds - with
01/03/2019	7946220	7940000
02/03/2019	7597053	8002279
03/03/2019	8393139	8002279
04/03/2019	7929690	8002279
05/03/2019	7839772	7993357
06/03/2019	8010650	8002279
07/03/2019	8412105	8002279
08/03/2019	11280623	11700079
09/03/2019	11864814	11602200
10/03/2019	11598043	11599200
11/03/2019	11424180	11593339
12/03/2019	11644833	11243589
13/03/2019	11179990	11900696
14/03/2019	11659527	11219056
15/03/2019	-6419757	-6900000
16/03/2019	1768563	2209488
17/03/2019	4146722	4177168
18/03/2019	4662285	4277168
19/03/2019	3813188	4477168
20/03/2019	4661174	4377168
21/03/2019	4177259	3777168
22/03/2019	3785624	4477168
23/03/2019	-494673	-988198
24/03/2019	-1065794	-900009
25/03/2019	-776247	-709999
26/03/2019	-681744	-857667
27/03/2019	-680314	-322052
28/03/2019	-1005583	-1116568
29/03/2019	-834518	-922000
30/03/2019	-794783	-642660
31/03/2019	-322858	-804234

TABLE 9. Optimized funds with and without predefined solution

References

[1] Alam, T., & Qamar, S., & Dixit, A., & Benaida, M. (2020). Genetic Algorithm: Reviews, Implementations, and Applications. *International Journal of Engineering Pedagogy (iJEP)*, 10(6), 57-77. https://doi.org/10.48550/arXiv.2007.12673

[2] Aliber, R. Z., & Chowdhry, B., & Yan, S. (2000). Transactions Costs in the Foreign Exchange Market. UCLA: Finance. https://escholarship.org/uc/item/4qw3p6rp

[3] Allais, M. (1952). Economie et Intérêt. Présentation nouvelle des problèmes fondamentaux relatifs au rôle économique du taux de intérêt et de leur solutions. *Revue économique*, 3(6), 892-895. www.persee.fr/doc/reco_0035-2764_1952_num_3_6_406950_t1_0892_0000_001

[4] Alvarez, F., & Lippi, F. (2009). Financial Innovation and the Transactions Demand for Cash. *Econometrica*, 77(2), 363-402. http://www.jstor.org/stable/40263869

[5] Armenise, R., & Birtolo, C., & Sangianantoni, E., & Troiano, L. (2011). Optimizing ATM Cash Management by Genetic Algorithms.

[6] Baumol, W.J. (1952). The transaction Demand for the Cash: An Inventory Theoretic Approach. *The Quarterly Journal of Economics*, 66(4), 545-556. https://doi.org/10.2307/1882104

[7] Bilir, C., & Doseyen, A. (2018). Optimization Of ATM And Branch Cash Operations Using An Integrated Cash Requirement Forecasting And Cash Optimization Model. *Business & Management Studies: An International Journal*, 6(1), 237-255. https://doi.org/10.15295/bmij.v6i1.219

[8] Chaouch, B.A. (2018). Analysis of the stochastic cash balance problem using a level crossing technique. *Ann Oper Res* **271**, 429-444. https://doi.org/10.1007/s10479-018-2822-2

[9] Chen, X., & Simchi-Levi, D. (2009). A NEW APPROACH FOR THE STOCHASTIC CASH BALANCE PROBLEM WITH FIXED COSTS. *Probability in the Engineering and Informational Sciences*, 23(4), 545-562. https://doi-org.recursos.biblioteca.upc.edu/10.1017/S0269964809000242

[10] Dondeti, Venkateswara R.. *Application of Optimization Techniques in Corporate Cash Management*. PhD Dissertation, Old Dominion University. https://www.doi.org/10.25777/d2w4-4y45

[11] Eppen, G. D., & Fama, E. F. (1968). Solutions for Cash-Balance and Simple Dynamic-Portfolio Problems. *The Journal of Business*, 41(1), 94-112. http://www.jstor.org/stable/2351960

[12] Eppen, G. D., & Fama, E. F. (1969). Cash Balance and Simple Dynamic Portfolio Problems with Proportional Costs. *International Economic Review*, 10(2), 119-133. https://doi.org/10.2307/2525547

[13] Cabello, J.G. (2013). Cash efficiency for bank branches. *SpringerPlus* **2**, 334. https://doi.org/10.1186/2193-1801-2-334

[14] Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. *Addison-Wesley Professional*

[15] Katoch, S., & Chauhan, S.S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimed Tools Appl* ***80***, 8091-8126. https://doi.org/10.1007/s11042-020-10139-6

[16] Kwak, N.K., & Renfro, M.D. (1989). An inventory model for optimal cash management for financial institutions: a case study. *RAIRO. Recherche Opérationnelle*, 23(1),99-112. http://www.numdam.org/item/RO_1989__23_1_99_0

[17] Lázaro, J.L., & Jiménez, A.B., & Takeda, A. (2018). Improving cash logistics in bank branches by coupling machine learning and robust optimization. *Expert Systems with Applications*, 92, 236-255. https://doi.org/10.1016/j.eswa.2017.09.043.

[18] Liu, S., & Tang, P. (2014). The Stochastic Cash Balance Problem with Fixed Costs: The Risk-averse Case. *Journal of Systems Science and Information*, 2(6), 520-531. https://doi.org/10.1515/JSSI-2014-0520

[19] Levi, D.M. (2009). International Finance. Routledge.

[20] Miller, M.H., & Orr, D. (1966). A Model of the Demand for Money by Firms. *The Quarterly Journal of Economics*, 80(3), 413-435. https://doi.org/10.2307/1880728

[21] Moraes, M., & Nagano, M. (2013). Cash Management Policies By Evolutionary Models: A Comparison Using The MILLER-ORR Model. *Journal of Information Systems and Technology Management*, 10(3), 561-576. https://doi.org/10.4301/S1807-17752013000300006

[22] Osorio, A.F., & Toro, H.H. (2012). An MIP model to optimize a Colombian cash supply chain. *International Transactions in Operational Research*, 19(5), 659-673. https://doi.org/10.1111/j.1475-3995.2011.00850.x

[23] Pacheco, M., & Noronha, M., & Vellasco, M., & Lopes, C. (2000). Cash Flow Planning And Optimization Through Genetic Algorithms. *Society for Computational Economics, Computing in Economics and Finance 2000*, 333.

[24] Paté-Cornell, M.E., & Tagaras, G., & Eisenhardt, K. (1990). Dynamic optimization of cash flow management decisions: A stochastic model. *Engineering Management, IEEE Transactions*, 37(3), 203 - 212. https://www.doi.org/10.1109/17.104290

[25] Salas-Molina, F., & Pla-Santamaria, D., & Garcia-Bernabeu, A., & Mayor-Vitoria, F. (2020). Multiple criteria cash management policies with particular liquidity terms, *IMA Journal of Management Mathematics*, 31(2), 217-231, https://doi.org/10.1093/imaman/dpz010

[26] Salas-Molina, F., & Rodriguez-Aguilar, J., & Pla-Santamaria, D. (2020). A stochastic goal programming model to derive stable cash management policies. *Journal of Global Optimization*, 76(2), 333-346. https://doi.org/10.1007/s10898-019-00770-5

[27] Sethi, S., & Thompson, G. (1970). Applications of Mathematical Control Theory to Finance: Modeling Simple Dynamic Cash Balance Problems. *Journal of Financial and Quantitative Analysis*, 5(4-5), 381-394. https://www.doi.org/10.2307/2330038

[28] Simon, D. (2013). Evolutionary optimization algorithms: biologically-Inspired and population-based approaches to computer intelligence. *John Wiley & Sons*.

[29] Tobin, J. (1956). The Interest-Elasticity of Transactions Demand For Cash. *The Review of Economics and Statistics*, 38(3), 241-247. https://doi.org/10.2307/1925776

Glossary

BAT: Baumol-Allais-Tobin model. 3
BnC: Branch-and-Cut. iii, 2, 17, 27–32, 52, 55–59

CFS: Cash Flow Financial Statement. 1
CMP: Cash Management Problem. 2, 4, 5, 9, 32

EA: Evolutionary Algorithm. 5

GA: Genetic Algorithm. 2, 5–9, 20, 27–29, 31, 32

LCY: Local Currency. 12, 14, 22, 27

MILP: Mixed Integer Linear Programming. 15, 31, 32