MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

APPLICATION OF META-HEURISTICS ON ATM CASH WITHDRAWAL FORECASTING

ESMA DANACI

MASTER OF SCIENCE THESIS WORK

Computer Engineering Department

SUPERVISOR

Assoc. Prof. Ali Fuat ALKAYA

ISTANBUL TURKEY, 2019

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

APPLICATION OF META-HEURISTICS ON ATM CASH WITHDRAWAL FORECASTING

ESMA DANACI

524113006

MASTER OF SCIENCE THESIS WORK

Computer Engineering Department

SUPERVISOR

Assoc. Prof. Ali Fuat ALKAYA

ISTANBUL TURKEY, 2019

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Esma DANACI, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended her thesis entitled "**Application of Meta-heuristics on ATM Cash Withdrawal Forecasting**", on September 3, 2019 and has been found to be satisfactory by the jury members.

Jury Members

Assoc. Prof. Ali Fuat Alkaya (Advisor)
Marmara University .....(SIGN) .....

Asst. Prof. Fatma Corut Ergin (Jury Member)
Marmara University.....(SIGN) .....

Assoc. Prof. Hasan Sözer (Jury Member)
Özyeğin University.....(SIGN) .....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Esma DANACI be granted the degree of Master of Science in Department of Computer Engineering on 18.09 2019. 2019/19-02

Director of the Institute
Prof. Dr. Bülent EKİCİ

PREFACE

I would like to thank my supervisor Assoc. Prof. Ali Fuat ALKAYA for his help, patience, support and guidance during development and completion of my master of science thesis study.

September, 2019 **ESMA DANACI**

TABLE OF CONTENTS

ÖZET

BANKA ATM' LERİNDEN NAKİT PARA ÇEKİMİ TAHMİNLEMESİ İÇİN METASEZGİSELLERİN UYGULANMASI

Bu tez çalışmasında, büyük bir Türk bankası için üstlenilen gerçek yaşam projesinden ilham almıştır. Banka ATM'lerinin ne zaman ziyaret edileceğini ve ne kadar para yükleneceğini etkin bir şekilde belirlemek, gerçek hayatta tüm bankalar için sorun haline gelmiştir. Bu nedenle, sorunun amacı önümüzdeki günlerde ATM'den ne kadar para çekileceğini tahmin etmektir. Bu tahmin görevi, kolay bir iş değildir çünkü geçmiş para çekme kalıpları çok değişkendir. Temelde, klasik zaman serisi yöntemleri verilerimiz üzerinde iyi performans göstermeyebilir. Sonuç olarak, çok sayıda başarılı akıllı sürü tekniğini uygulamaya karar verdik çünkü bunlar zaten geniş bir yelpazedeki sorunlara başarıyla uygulanmakta olan iyi performans göstericileri olarak biliniyor.

Bu tezde, bu problemin üstesinden gelmek için yeni meta-sezgisel temelli çözüm yaklaşımları geliştirilmiştir. Spesifik olarak, bu çalışmada kullanılan teknikler şunlardır; yapay arı kolonisi, diferansiyel evrim, göçmen kuşlar optimizasyon algoritması, parçacık sürüsü optimizasyonu ve benzetilmiş tavlama. Bu çalışmada, söz konusu algoritmalar öncelikle sorunun yapısını göz önünde bulundurarak algoritma operatörlerini tasarlayarak uygulanır. Ardından, algoritmaların parametreleri hesaplama testleri ile ayarlanmıştır. Son aşamada, tüm meta-sezgiseller problem örneklerine en iyi performans gösteren parametre değerleri ile uygulanmakta ve kapsamlı hesaplama deneyleri ile karşılaştırılmaktadır. Karşılaştırma sonucunda diferansiyel evrim ve göç eden kuşlar optimizasyon algoritmasının birbirine yakın ve en iyi değerleri verdiği gözlemlenmiştir.

ABSTRACT

APPLICATION OF META-HEURISTICS ON ATM CASH WITHDRAWAL FORECASTING

This thesis work is inspired from real life project which had been undertaken for a large Turkish bank. Determining when to visit and how much to load to each ATM of the bank in an efficient way has become a problem for all banks in real life. Therefore, objective of the problem turns out to be forecasting how much money will be withdrawn from the ATM in the next days. This forecasting task is no way an easy task to cope with since the past withdrawal patterns are too volatile. In essence, the classical time series methods may not perform well on our data. As a result, we decided to implement several swarm intelligence techniques since they are known as well performing optimizers being implemented to broad range of problems successfully already.

In this thesis, novel meta-heuristic based solution approaches are developed to cope with this problem. Specifically, the techniques utilized in this study are; artificial bee colony, differential evolution, migrating birds optimization, particle swarm optimization and simulated annealing. In this study, aforementioned algorithms are implemented first by designing the operators of the algorithms by considering the nature of the problem. Then, parameters of the algorithms are fine-tuned with computational tests. In the last phase, all meta-heuristics are applied to the problem instances with their best performing parameter values and compared through extensive computational experiments. As a result of the comparison, it has been observed that the differential evolution and migrating birds optimization algorithm give close and best values.

ABBREVIATIONS

ATM : Automated Teller Machine

MAPE : mean Absolute Percentage Error

SMAPE : Symmetrical Mean Absolute Percentage Error

MAD : Mean Absolute Deviation

ABC : Artificial Bee Colony

SA : Simulated Annealing

DE : Differential Evolution

ARIMA : Auto Regressive Integrated Moving Average

SARIMA : Seasonal Auto Regressive Integrated Moving Average

MBO : Migrating Birds Optimization

PSO : Particle Swarm Optimization

ACO : Ant Colony Optimization

LM : Levenberg–Marquardt Algorithm

ANN : Artificial Neural Network

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

cost*(X):* cost of solution X

dayData.min: minimum withdrawal amount on a day

datData.max: maximum withdrawal amount on a day

atmDayData.amount: withdrawal amount on a specific day

fsp: fitness selection parameter

R_i*:* real value of ith solution

E_i : estimated value of ith solution

N: number of solutions

A: number of attributes

D_{max}*:* maximum amount on each day

D_{min}*:* minimum amount on each day

X_j*:* weight of jth attribute

D_{ij}*:* weight of jth attribute of ith solution

noi: number of instances

runtime: parameter of ABC algorithm

nos: number of solutions

lb: lower bound

ub: upper bound

r: randomly generated number

X_i*:* ith solution

X_i*':* neighbor of ith solution

prob(): probability function

fitness(): fitness function

a: parameter of ABC algorithm

b: parameter of ABC algorithm

limit: parameter of ABC algorithm

maxCycle: parameter of ABC algorithm

non: number of neighbors, parameter of MBO algorithm

olf: overlap factor, parameter of MBO algorithm

nob: number of birds, parameter of MBO algorithm

bound: value bound

radius: value distance limit

nsp: neighbor selection parameter

nof: number of flapping, parameter of MBO algorithm

P: population

Q: population

CR: crossover probability, parameter of DE algorithm

F: mutation factor, parameter of DE algorithm

$X_{M:}$ mutant solution

$X_{M,i:}$ weight of ith attribute of mutant solution

msp: mutation selection parameter of DE algorithm

$P_{r1,\,i:}$ solution at index r1 of population P

$P_{best:}$ best solutions population

G: global best solution of PSO

P': newly created population of neighbor solutions

omega: parameter of PSO algorithm

φ1, φ2: parameters of PSO algorithm

vMax: maximum velocity, parameter of PSO algorithm

vsp: velocity selection parameter

random: random number between [0,1]

pMax: maximum position, parameter of PSO

psp: position selection parameter

$X_{T:}$ trail solution

nog: number of generations, parameter of PSO algorithm

R: parameter of SA algorithm, number of iteration

maxR: maximum value of parameter R.

totalR: sum of R values in iteration of SA algorithm

T: temperature, parameter of SA algorithm

a: decrease ratio of T, parameter of SA algorithm

b: increase ratio of T, parameter of SA algorithm

ns: neighbor soluti

1. INTRODUCTION

Although the use of mobile & online banking and electronics money transfers have been increased in the last decades, cash is still the most widely used payment practice and the number of ATMs have a rising trend as the ATMs are cost effective when compared with bank branches. The worldwide number of ATMs increased to 3.6 million in 2017 from 3.5 million in 2015 and estimated to be 4 million by the end of 2021 [1].

Optimum cash management and cash service availability are outstanding factors in the ATM network services business. Banks can prevent ATMs running out of cash money and keep their system available in continuously varying environment with help of using cash load optimization and efficient cash management. More recently, becoming an important issue of achieving higher efficiency in managing cash at ATMs has turn banks attention to ATM withdrawal forecasting [2].

Although, forecasting is a very popular topic which has been studied widely so far, the number of studies made particularly on ATM forecasting is rather limited. The competition made on a number of ATMs from UK has made it become popular. The aim was to predict the cash withdrawal amounts of the next four weeks using the withdrawal data of past two years [3]. The performance measure was SMAPE (symmetrical mean absolute percentage error). Andrawis et al. took the first place in the competition with their study which uses forecast combination [4].

On the other hand, meta-heuristics are used to solve the optimization problems and they often result good solutions in fair times. Therefore, using meta-heuristics for forecasting money withdrawals from ATMs is a novel idea. Objective of this thesis work is to predict ATM cash withdrawal amounts by using meta-heuristics and to compare results of each meta-heuristics. Specifically, the meta-heuristics exploited in this study are artificial bee colony (ABC), migrating bird optimization (MBO), differential evolution (DE), particle swarm optimization (PSO) and simulated annealing (SA). The aforementioned methods are customized to perform best for the ATM forecasting problem.

All of the algorithms use MAD (mean absolute deviation), MAPE (mean absolute percentage error) and SMAPE (symmetric mean absolute percentage error) as the fitness function. Although, many researchers used MAPE or SMAPE to measure their forecasting

performances, in our study we preferred to use also MAD since MAPE or SMAPE are unstable when the withdrawal amounts are small and we have a lot of such figures in the dataset.

The computational experiments are performed on real data having numerous characteristics. The study is comprised of four steps: (1) we worked on data; we normalized and scaled amounts for each ATM, (2) the forecasting is performed on volatile & non-volatile ATM data, (3) we analyzed results and figured out best performing ones, then all tests run again for each meta-heuristic with its best performing parameter set, (4) we compared five different meta-heuristics with extensive computational experiments.

The contribution of this study is that it is first time, all aforementioned meta-heuristics are applied to ATM cash withdrawal forecasting problem. Contribution of this thesis to literature is as follows:

- Esma Danacı, Ali Fuat Alkaya, Onur Gürkan Gültekin. An Empirical Analysis of Swarm Intelligence Techniques on ATM Cash Withdrawal Forecasting. INFUS 2019, AISC 1029, pp. 1235–1242, 2020

This thesis is organized into five main sections including introduction. In the next section we provide a literature survey regarding the forecasting ATM withdrawal and meta-heuristics implementations of forecasting problems. Section 2 gives the descriptions of the meta-heuristics and related literature. In section 3, work done including the characteristics of used real data and all details of application of meta-heuristics to related problem are showed. Section 4 presents the extensive experimental setup, the best parameter values of the algorithms, experiment results and discussions about the results. Section 5 includes conclusion of the thesis by a summary of the contributions of it and possible future works.

2. SURVEY OF RELATED LITERATURE

Background information about related the studies on money withdrawal forecasting is provided and then studies about applications of meta-heuristics for forecasting problems are shown in this section. Later on, short definitions of the artificial bee colony, differential evolution, migrating bird optimization, particle swarm optimization and simulated annealing are given.

2.1. Money Withdrawal Forecasting and Implementations of Meta-heuristics on Forecasting Problems

In this subsection, first the studies performed on the money withdrawal forecasting is presented and contributions of this study is shown. Thereafter, we present our survey results on the studies related with the implementations of meta-heuristics on forecasting problems since same techniques in this study may be applied to that forecasting problems.

2.1.1. Previous Studies on Money Withdrawal Forecasting

In literature, techniques implemented on money withdrawal forecasting can be generally grouped into three [2, 5-8]:

- Time series method which estimates future money necessity based on the previous values of money withdrawal. If there is not varying factors that affect the withdrawal pattern or a sudden change in the environment, techniques in time series method work well in most cases.
- Factor analysis method, that is based on the specification of many variables affecting the money demand pattern and revealing their relation with real money withdrawal pattern. Purpose of factor analysis method is to clarify functional model of this independent factors and implementing this function to predict future values of the dependent variables. Unfortunately, revealing this functional model is not an easy task when there exists a non-linear relationship between variables.

- Neural networks correlates relationships between several variables having effect on the money withdrawal and the real money withdrawal. Criticism of this money withdrawal forecasting method has been implemented by using standard measures in

statistics such as percentage errors MAPE and SMAPE. Artificial neural networks have better steadiness; strong self-learning ability and nonlinear mapping ability. However, they likely converge to a local minimum and they have slow convergence.

Other methods applied to cash withdrawal forecasting, are commonly based on linear regression models using seasonal coefficients adjusted for each ATM. It differs for various ATMs and it is complicated to develop such models.

Venkatesh et al combine clustering techniques with their different artificial neural network models to forecast ATM withdrawals for the 111 ATM data from the NN5 competition [5]. In a recent study, Crone and Kourentzes propose a whole data based on estimation that unions filter and wrapper techniques for feature selection, automatic evaluation, construction and transformation of features [3].

On the other hand, regarding the multiple linear regression models, Andrawis et al. revealed diversity to their model by combining nonlinear and linear models such as multiple linear regression for the NN5 competition [4]. Simutis et al. made a prediction getting 3 years' data of a bank in Lithuania. For training the ANN, Levenberg-Marquard optimization method was used and their performance measure was MAPE [6].

Baker et al. developed a new data-oriented technic for a stochastic inventory problem of a large financial institution having a lot of ATMs in USA [7]. Opposite to stochastic inventory control models for money loading used in practice, they did not consider that the forecasting errors are normally distributed for a long time period with constant parameters. Rather, they got the best-fitting forecast error distribution periodically for the newer data.

To sum up, according to relevant studies, we may easily say that the performance meta-heuristics are not fully performed. Hence, our contributions in this study is development and comparison meta-heuristic algorithms, artificial bee colony optimization, differential evolution, migrating bird optimization particle swarm optimization and simulated annealing algorithms, that can solve the ATM withdrawal forecasting problem.

In next section, we demonstrate literature survey of the studies focused on the forecasting by using meta-heuristics.

2.1.2. Previous Work on Forecasting by Using Meta-heuristics

Exploitation of meta-heuristics on forecasting became popular in recent years. In a recent study, energy requirement of Turkey is estimated using PSO particle swarm optimization and ACO ant colony optimization in a hybrid model. Proposed hybrid method is performed to predict demand of energy by using export, import, population and GDP: gross domestic product. Future energy demand is forecasted with different cases and scenarios. In order to demonstrate performance of the algorithms PSO and ACO developed for the same problem are compared. With analysis of performance comparison results, it is shown that the hybrid method outperforms others [9].

Another hybridization of PSO and ACO takes place in where the authors try to forecast the energy output of a real wind farm. The empirical wind power output for 364 days is used to train then test the proposed model where temperature and wind speed are assumed as inputs to the model. They show that the hybridization of two algorithms bring higher quality results with a faster convergence [10].

Another usage of meta-heuristics in the forecasting literature is due to the training of the artificial neural networks, ANN. In one of recent studies, Piotrowski et al implemented a detailed performance comparison of nature-inspired optimization algorithms and Levenberg–Marquardt algorithm in ANN training, depending on the forecasting study of water temperature in a natural river. Such as PSO particle swarm optimization, evolution strategies, direct search methods, multi algorithms and 50 variants of 22 various meta-heuristics, including differential evolution, are compared with Levenberg-Marquardt algorithm to train ANNs. Results show that differential evolution algorithms are competitive to LM algorithm [11].

Another study proposes a new time-series forecasting methods based on nonlinear regression. The authors have used a hybrid meta-heuristic comprising two components: one PSO particle swarm optimization and other one SA simulated annealing to estimate the model parameters. Evaluation of the proposed method's performance is done by using standardized testing problems and compared with performances of related models in literature. SARIMA and ARIMA models reveal that the proposed method yields lower errors for results of datasets in solving on 11 problems with different structure [12].

2.2. Artificial Bee Colony

Artificial bee colony algorithm was introduced by Baştürk and Karaboğa. In original study of ABC is used for numerical multi-variable function optimization [13]. Being influenced by intelligent behavior of honey bees, It gets inspired from nature and based on population of organisms. ABC is an optimization tool providing a nature inspired, population-based search function in which solutions called food's locations are changed by the bees within time and the bee's duty is to finding better locations of food sources with high nectar amount. ABC uses just general controlling parameters such as colony size and maximum cycle number.

In artificial bee colony ABC algorithm, bees fly in a multi-dimensional search space and some employed and onlooker bees choose food sources based on the experience of themselves and other bee's experiences. Scout bees start to fly and pick the food source randomly. If the nectar amount of new food source is higher than that of the previous one in their memory, they memorize the new food source location and forget the previous one. Therefore, artificial bee colony system joins local searching that is performed by employed and onlooker bees, with global search methods performed by onlookers and scouts. Their attempt is to balance exploitation and exploration processes [14].

ABC algorithm is widely used for forecasting problems in different areas. Awan at al. combine ABC algorithm with artificial neural network for electric load forecasting [15]. ABC is modified and compared with conventional form on gold price forecasting by Li at al. [16]. Supreetha at al. also developed an innovative hybrid ABC algorithm based on PSO algorithm to forecast ground water levels [17].

2.3. Differential Evolution

DE is global optimization and stochastic direct search algorithm. It is one of the popular evolutionary algorithms proposed by Storn and Price in 1997 [2] DE maintains a population of candidate solutions with iterations of recombination, evaluation, and selection. First step, the recombination includes generation of new candidate solutions depends on the weight's difference between two randomly picked solutions added to a third one. The difference can be calculated by using the mutation factor (F).

The mutated solution's parameters are then mixed with the parameters of another predetermined solution, the target solution, to get the trial solution. Elements of the mutated solution enter the trial solution with a probability CR. So, each solution has to serve once as the target solution and who will survive to the next generation is based on fitness function value of the target and trial solutions [2]. This perturbs solutions comparative to the propagation of the larger population [19].

Differential evolution algorithm has several applications on prediction of time series data, one of them is study of Mandal showing DE implementation with MAPE is better than neural network prediction results with different time series data sets such as airline bookings and university enrollments [20].

2.4. Migrating Birds Optimization

Migrating bird optimization algorithm is recently introduced, population-based neighborhood search algorithm. MBO simulates the V form flight of the migrating birds [21]. In this meta-heuristic, initial solutions represent the flock and leader bird of flock is assumed first solution. Hence remaining solutions is grouped into two wings as left side and right side of V shape.

MBO has common control parameters such as number of solutions, number of flapping and number of neighbors. Each solution creates a number of neighbor solutions and takes a specific number of unused neighbor solutions from its preceding solution on the V shape form. Number of neighbor parameter has effect on determination of exploration when this value gets high, the flock explores its environment in detail so the number of neighbors indicates the speed of the flock Every solution in flock is tried to be improved by comparing its created neighbor solutions and borrowed overlapped neighbor solution of preceding solution. if neighbor solutions performance is better than one of current solution, it is replaced with the neighbor solution otherwise the solution is compared with overlapped (best unused) neighbor solution of preceding and this process goes on the lines towards the V form. Evaluation on unused best neighbors of the previous solution indicates one of the important aspects of the MBO algorithm is the benefit mechanism.

This process repeats iteratively in a predetermined number of times when all solutions are tried to be improved. Then first solution positioned at the end of flock, and one of the nearest

solutions becomes first and loop goes on. Stopping criteria of MBO is the parameter *noi*; number of iterations and number of total solutions created.

Migrating birds optimization algorithm is applied to wide range of problems such as credit card fraud detection by Duman at al. [22]. Migrating birds optimization algorithm is also applied to obstacle neutralization problem and medical diagnosis by using breast cancer data [23,24]. However, there is not forecasting implementation of it in the literature.

2.5. Particle Swarm Optimization

PSO, particle swam optimization algorithm simulates social behavior patterns of organisms interacting and living within large groups [25]. In particle swarm optimization algorithm, a predefined number of particles are located in the problem search space, and each one calculates the objective function value at its current place.

Thereafter for each particle next movement is determined through the search space by joining accordingly history of its own current and best locations and location histories of other members in the swarm, by using some random perturbations. After all particles have been moved or tried to moved, next iteration starts. In the end, whole swarm collectively, is likely to move close to an optimal value of the fitness function [26].

Particle Swarm Optimization is applied to several forecasting problems by combining neural network to optimize its performance or by comparing artificial neural networks. PSO with center of mass technique is applied training linear combiner to form a new stock market prediction model by Seidy at al. [27]. They observed proposed technique is superior than the other PSO based models according to the prediction accuracy. PSO is also used to determine the parameters of an autoregressive model for time series prediction by Islam at al. The model is applied for annual rainfall prediction and they conclude a fairly good result in comparison to ARIMA model [28].

2.6. Simulated Annealing

Simulated Annealing is a probabilistic method for finding the global minimum of a cost function that may process several local minima. Unlike any other hill climbing algorithm,

simulated annealing occasionally accepts worse solutions in other words, it works slightly different. This different characteristic of simulated annealing helps jumping out of any local optimums. SA works by imitating the physical process that is a solid is slowly cooled so that eventually it gets frozen at minimum energy configuration [29].

The main idea is to allow doing moves producing solutions with worse quality than the current solution in order to run away from local minimum. During searching probability of doing such a move is reduced. SA gets start by generating a randomly initial solution, *X*, and by initializing the so-called temperature parameter *T*. In each cycle, then a solution *X'* in *N(X)* is randomly created and it is considered as new current solution depending on cost*(X), cost(X')* and *T*. s*'* replaces *X* if cost*(X') < cost(X)* or cost*(X') ≥ cost(X),* with a probability that is a function of *T* and cost*(X')-cost(X)* [30]. Then temperature T decreased and cycles goes on.

Simulated annealing algorithm is applied to wide range of forecasting problems by combing artificial neural networks. Some of them are municipal solid waste generation prediction by Song at al. [31] and financial time series forecasting by Mancha at al. [32]. Mancha used a hybrid SA and compared hybrid algorithm with ARIMA model.

In the next section, detailed information about how aforementioned meta-heuristics are implemented to ATM cash withdrawal forecasting problem is provided.

3. WORK DONE

This section includes description of how the meta-heuristics are adapted to forecasting problem to get efficient results for forecasting ATM cash withdrawal. Firstly, we describe our data in the next subsection. Then, we describe the details of fitness value calculations. After that, the details of ABC, MBO, DE, PSO and SA are given which are again customized for the problem.

3.1. About Data

For each ATM we are given a number of days together with its withdrawal amount and a set of 59 attributes describing the day. The attributes of a day are represented by binary values. As is weekday, is weekend, is holiday, is [1st, …, 4^{th}] week of the month, is [1st, …, 30th] day of the month, is [1st, …, 7th] day number of week, is 1^{st} weekday of month, is last weekday of the month, is middle of the month etc. and also there are 3 different peak day features showing in which days, withdrawal amounts increase extremely. So, we had 59 different binary features to describe the transaction date that can be seen in Table 3.1.

Table 3.11. Binary Input Feature

All Extracted Input Feature Columns		
Day_1	Day_2	Day_3
Day_4	Day_5	Day_6
Day_7	Day_8	Day_9
Day_10	Day_11	Day_12
Day_13	Day_14	Day_15
Day_16	Day_17	Day_18
Day_19	Day_20	Day_21
Day_22	Day_23	Day_24
Day_25	Day_26	Day_27
Day_28	Day_29	Day_30
Day_31	Sunday	Monday
Tuesday	Wednesday	Thursday
Friday	Saturday	Weekday
Holiday	January	February
March	April	May
Jun	July	August
September	October	November
December	Special_Day	Peak_A
Peak_A_A1	Peak_B	Peak_B_A1
Peak_C	Peak_C_A1	

The ATM data we used in computational experiments is the withdrawal data of 10 ATMs obtained from a bank where the date interval is 31.07.2012-01.08.2013. Each algorithm uses the 366 days data for training whereas the test data is the following 30 days in the calendar (02.08.2013-31.08.2013).

Because ATM attribute features are in binary format, we prefer to scale withdrawal amounts. Variance of the features should be in the same range to prevent from domination of feature having highly variance over other features. For our experiments before applying any algorithms in preprocessing step, we found minimum and maximum withdrawal amounts of each day and we scaled amount data according to ratio; difference of amount and minimum amount over difference value of maximum amount and minimum amount.

$$
\begin{gathered}
min = dayData.min \\
max = dayData.max \\
delta = max - min \\
foreachATM \\
atmDaydata.amount = (atmDaydata.amount - min)/delta
\end{gathered} \tag{1}
$$

Therefore, for an ATM the given information can be stored in a list of complex objects including ATM name, transaction date, withdrawal amount and an array list of attributes. The object represents one day data and ATM data file represented in list of it. We defined solution object storing list of weights that have been calculated in a different way for each algorithm.

3.2. Fitness Calculations with MAD, MAPE, SMAPE

We used same fitness function for all meta-heuristics. This function has a parameter to select in which way the fitness value will be calculated. If this fitness selection parameter so-called fsp equals 1, MAPE is used, if it is 2, MAD is used and if it is 3 SMAPE is used. We run our tests for three of them with same parameters and same ATM data files.

Mean absolute deviation (MAD) is a way to describe variation in a data set. Mean absolute deviation of a data set is the average distance between each data value and the mean. Mean absolute deviation helps us get a sense of how spread out the values in a data set are.

$$
MAD = \frac{\frac{\sum_{i=1}^{N} |R_i - E_i|}{N}}{\frac{\sum_{i=1}^{N} R_i}{N}} \tag{2}
$$

where N is the number of days, R_i is the real withdrawn amount and E_i is the estimated withdrawal amount.

MAPE, mean absolute percentage error is known also as mean absolute percentage deviation. In statistics MAPE is a measure of prediction accuracy of a forecasting method. It is defined as below formula and it is clearly seen that expression of accuracy is in percentage format.

$$
MAPE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{R_i - E_i}{R_i} \right| \tag{3}
$$

SMAPE standing for symmetric mean absolute percentage error is a modified form of the MAPE. It is formulated by including average of the absolute value of the real and the absolute value of the estimate in the denominator. In several forecasting studies SMAPE has preference to the MAPE as an accuracy measure.

$$SMAPE = \frac{2}{N}\sum_{i=1}^{N}\left|\frac{R_i - E_i}{R_i + E_i}\right| \quad (4)$$

We calculated estimated value by sum of current day's minimum withdrawal amount and current day's min. max. withdrawal amount difference multiplied by a value obtained by sum of all attribute values multiplied by corresponding weight. Our estimation function is as below where D_{max}, D_{min} are maximum and minimum withdrawal amounts of each day, X is solution vector which stores the weight of attributes and A is the size of attributes.

$$E_i = D_{min} + (D_{max} - D_{min}) * \sum_{j=1}^{A} D_{ij}\, X_j \quad (5)$$

3.3.Artificial Bee Colony Implementation

The steps of the artificial bee colony algorithm are on Figure 3.1.

1. Initialize solutions randomly.
2. Send employee bees to find new solutions.
3. Send onlooker bees and find new better solutions by using probability calculations of employee bees.
4. Send the scout bees to generate new solutions.
5. Memorize the best solution so far
6. Repeat step 2 to 5 until the stopping criteria are met.

Figure 3.31. Steps of ABC Algorithm

Stopping criteria of ABC is either a predefined parametric number of iterations, *runtime* and a predefined number of created solutions, named *noi*, number of instances.

At first step of algorithm, a number of food sources, food number i.e. number of solutions *nos*, are initialized randomly by using below function, with lower bound, *lb***,** and upper bound, *ub***,** parameters and r is randomly generated values.

$$X_j = r * (ub - lb) + lb \tag{6}$$

For every newly created solution's trial parameter (related to nectar amount) is set to zero and fitness of each food source is obtained in first step. This task is performed by the initial scout bees supporting exploration.

At second step, employee bees find new food sources, new solutions are created by using below function where *p* and *k* are random number between [1, A], A is the size of attributes and *r* is randomly generated number.

$$X_i' = X_{ip} + \left(X_{ip} - X_{kp}\right) * r \tag{7}$$

A randomly chosen solution X_k is used in producing a mutant solution and the parameter to be changed determined randomly too. if generated parameter value is out of boundaries (*lb, ub*), it is shifted onto the boundaries. Then a greedy selection is done between the current solution *i* and its mutant, if the mutant solution is better than the current solution *i*, replace the solution with the mutant.

In step three, probabilities of generated solutions are calculated. A food source is chosen using probability value which is proportional to its quality. Probability values are evaluated using fitness values and normalized form of it by dividing maximum fitness value within below equation.

$$\begin{gathered} prob(X_i) = a * fitness\,(X_i)/maxFitness + b \\ a = 0.9, b = 0.1 \end{gathered} \tag{8}$$

Then onlooker bees sent to find better food sources/solutions. Onlooker bees chose a food source depending on its probability to be chosen. Chosen solution is tried to be improved by randomly generated mutant solution. if it is improved then trial is increased. One solution can be improved until its trail count arrives nectar amount limit. At step five, best solution having minimum cost is memorized. And iteration goes on from step two to step five until total cycles reach *maxCycle* parameter.

3.4. Migrating Birds Optimization Implementation

Pseudo-code of our proposed MBO algorithm can be seen on Figure 3.2.

```
Generate nob initial solutions randomly.
Locate solutions in a hypothetical V-form.
while stopping criteria not met
        for nof times
                Create neighbors of leader soluiton.
                Try to improve leader solution by comparing with its neighbors.
                for each solution evaluates "non-olf" number of created own neighbors and
                olf unused best neighbors of preceding solution.
                end for
        end for
        Set leader solution as last solution in flock.
        Make closest follower solution leader.
end while
return the best solution of flock.
```

Figure 3.42. Pseudocode of MBO Algorithm

Termination condition of migrating birds optimization algorithm is a *noi* that is predefined parametric number of iterations.

Firstly, we create a population/list of solutions, flock, including a predefined number of solutions named number of bird *nob* parameter. Each solution's weights are set randomly generated numbers and fitness values of each solution is calculated.

Then flock flies until iteration count arrives at number of flapping *nof* parameter value. At each flapping, a number of neighbor solutions *non* are created for leader bird/solution by using a value *bound*, *radius* and a neighbor selection parameter *nsp* having values 1 to 5.

We implemented five different neighbor selection by using a randomly generated number *r*, value bound and radius parameters that is new neighbor's distance to current solution. New solutions are mutant of current solution obtained by changing the weights of it. If *nsp* equals 1, neighbor solution value is currents solution's weight increased by a random value depending on *bound* and *radius*. If *nsp* equals 2, it is increased by precision of that random value, so new value is between [0,1]. If *nsp* equals 3, new weights calculated same as case 1 and maximum of new weights found, if max weight is greater than bound parameter, all weights recalculated as their current value over maximum value. In case 4, new weights calculated same as case 1 but values are restricted between [-*bound*, *bound*] and if *nsp* equals 5, new values restricted between [0, *bound*].

nsp=1, $X_j' = X_j + (-bound + r * 2 * bound * radius)$ (9)

nsp=2, $X_j' = \Big(X_j + (-bound + r * 2 * bound * radius)\Big) remainder 1$ (10)

nsp=3, $X_j' = X_j + (-bound + r * 2 * bound * radius)\ if\ X_j' max > bound\ then\ X_j' = X_j'/X_j'\ max$ (11)

nsp=4,
$$X_j' = X_j + (-bound + r * 2 * bound * radius)$$
$$if X_j' > bound\ then\ X_j' = bound$$
$$elseif X_j' \leftarrow bound\ then\ X_j' = -bound$$
(12)

nsp=5,
$$X_j' = X_j + (-bound + r * bound * radius)$$
$$if X_j' > bound\ then\ X_j' = bound$$
$$elseif X_j' < 0\ then\ X_j' = 0$$
(13)

After neighbors created and sorted leader tries to improve itself by using its best neighbor solution. If solution's best neighbor is better than itself, it is replaced. A number of unused neighbors, overlap factor *olf*, are sent to the following bird/solution. Unfortunately, because of in original flow of MBO algorithm, parameter *non* must be or greater than or equal to value calculated with formula 14.

$$2 * olf + 1 \quad (14)$$

Follower solution gets a number of overlapped neighbors and generates its own neighbors than tries to improve itself by using best neighbor of all neighbors of it. This goes on until last bird at flock tries to improve itself then it replaced by leader solution.

3.5. Differential Evolution Implementation

Differential evolution algorithm pseudocode is on Figure 3.3.

Generate population $P = (X_1, X_2, \ldots X_{nos})$ and initialize solutions randomly.
repeat
    for $i = 1$ **to** nos **do**
        Generate $r1, r2, r3$ randomly and $r1, r2, r3$ and
        i are not equal each other and $r1, r2, r3 < nos$
        Compute a mutant solution M by using $r1, r2, r3$ and mutation factor F.
        Create trail solution by the crossover of current solution X_i, mutant solution
        X_M and CR crossover probability.
        if fitness value of trail solution X_T is better than current solution X_i
            then insert trail solution X_T into new population Q
        else insert current solution X_i into Q
        endif;
    endfor;
$P = Q$;
until stopping condition;

Figure 3.53. Pseudocode o DE Algorithm

Termination condition of DE algorithm is a *noi* that is predefined parametric number of iterations.

.

First step is generation of initial population with *nos* number of solutions by setting weights of each solution randomly and fitness values are calculated. Then for each solution *r1*, *r2*, *r3* are randomly generated with constraints *r1*, *r2*, *r3*, index of current solution *i* are not equal each other and *r1*, *r2*, *r3* are numbers between [1, *nos*].

Then we generate a mutant solution by using solutions at *r1*, *r2*, *r3* indexes of population and *F* mutation factor parameter of algorithm. Mutant solutions weights are calculated by using these randomly chosen three different solutions weights of attributes. Our mutant solution creation function has also a parameter *msp* having values 1 to 4 to decide the way of mutation. There are four different mutation implementations as below. If *msp* equals 1, mutation solution generated by using weights of P_{r1}, P_{r2}, P_{r3} and *F*. If *msp* equals 2, weights calculated same as case 1 but precision values are set to mutation solution, limit is [0,1]. In case 3, new weights calculated same as in case 1, and maximum weight found if max. weight is greater than 1, all weights set as currents weight over max. For *msp* equals 4, new weights restricted between [0,1].

msp=1, $X_{M,i} = P_{r1,i} + F * \left(P_{r2,i} - P_{r3,i}\right)$ (15)

msp=2, $\left(X_{M,i} = P_{r1,i} + F * \left(P_{r2,i} - P_{r3,i}\right)\right) remainder\ 1$ (16)

msp=3,
$$X_{M,i} = P_{r1,i} + F * \left(P_{r2,i} - P_{r3,i}\right)$$
$$if\ max\left(X_{M,i}\right) > 1\ then$$
$$X_{M,i} = \left(P_{r1,i} + F * \left(P_{r2,i} - P_{r3,i}\right)\right) / max\left(X_{M,i}\right)$$ (17)

msp=4,
$$X_{M,i} = P_{r1,i} + F * \left(P_{r2,i} - P_{r3,i}\right)$$
$$if\ X_{M,i} < 0\ then\ X_{M,i} = 0$$
$$if\ X_{M,i} > 1\ then\ X_{M,i} = 1$$ (18)

After creating mutant solution, crossover phase starts by using current solution, its mutant solution and *CR* crossover probability parameter of DE algorithm. We obtain trail solution by changing weight of attributes with our crossover function where *r* is randomly generated value between [0,1] and *k* is randomly generated number between [1, A], A is size of attributes. If *r* is less than *CR* or *i* equals *k* then trail solution is mutant solution else trail solution is current solution.

In selection phase, fitness value of trail solution is compared with fitness value of current solution, the better one is added to newly created population. After each solution is compared with its trail solution in the initial population, it is set to newly created populations of accepted trail solutions until the stopping criteria is met.

3.6. Particle Swarm Optimization Implementation

Our PSO implementation steps are as follows on Figure 3.4.

1. Initialize randomly a population of solutions with a number of *nos* parameter.
2. Set velocity vector of each solution and add all solutions to newly created population that will hold best solutions.
3. For each solution calculate new velocities.
4. Then create trail solution by changing position of current solution with new velocity.
5. If trail solution is better than the solution at same index on population of best solutions then replace it. If trail is also better than global best solution thereafter set global best solution to trail solution.
6. Add trail solution to newly created population.
7. After processing last solution, set population to newly created population.
8. Repeat 2 to 6 until the stopping criteria are met.

Figure 3.6. Steps of PSO Algorithm

The algorithm starts with creation of *nos* number of solutions with randomly generated weights. Then for each solution velocity vector is generated with random numbers in range of [0,1]. We create a new population P_{best} to hold best solutions and initially each solution's best solution is itself. Then global best G is set to the minimum cost solution in bests population. Before starting iteration on solutions, a new population P' is created.

For each solution we create a velocity vector by using *omega*, $\varphi 1$, $\varphi 2$, *vMax*, global best solution that is minimum cost solution of population of best solution and local best solution

that is solution at P_{best} at same index with current solution. Changing velocity function also has a velocity selection parameter *vsp*. The parameter *vsp* can have five different values to calculate velocities in five different way. Change velocity function implementation can be seen on equations 20, 21, 22, 23 and 24. Same strategy is used in MBO neighbor creation function. In case 1, a random value used and in case 2, precision of that random value is used i.e. value is restricted in between [0,1]. For case 3, random value over maximum of it used. In case 4 values restricted between [-*vMax*, *vMax*] and in case 5 limit is [0, *vMax*].

vsp=1, $$V'_j = \left(V_j * omega\right) + \left(P_{best,ij} - P_{ij}\right) * r1 + \left(G_j - P_{ij}\right) * r2 \quad (19)$$

vsp=2, $$V'_j = \Big(\left(V_j * omega\right) + \left(P_{best,ij} - P_{ij}\right) * r1 + \left(G_j - P_{ij}\right) * r2\Big) remainder\ 1 \quad (20)$$

vsp=3, $$\begin{gathered} V'_j = \left(V_j * omega\right) + \left(P_{best,ij} - P_{ij}\right) * r1 + \left(G_j - P_{ij}\right) * r2 \\ if\ max(V') > vMax \\ then\ V'_j = V'_j / max\ (V') \end{gathered} \quad (21)$$

vsp=4, $$\begin{gathered} V'_j = \left(V_j * omega\right) + \left(P_{best,ij} - P_{ij}\right) * r1 + \left(G_j - P_{ij}\right) * r2 \\ if\ V'_j < -vMax \\ then\ V'_j = -vMax \\ if\ V'_j > vMax \\ then\ V'_j = vMax \end{gathered} \quad (22)$$

vsp=5, $$\begin{gathered} V'_j = \left(V_j * omega\right) + \left(P_{best,ij} - P_{ij}\right) * r1 + \left(G_j - P_{ij}\right) * r2 \\ if\ V'_j < 0 \\ then\ V'_j = 0 \\ if\ V'_j > vMax \\ then\ V'_j = vMax \end{gathered} \quad (23)$$

where $r1 = \varphi 1 * random, r2 = \varphi 2 * random$

After obtaining the vector of new velocities, trial solution is created by changing current solutions position using changed velocity vector and maximum position *pMax* parameter. Changing velocity function also has position selection parameter *psp* to calculate new position in different strategies same as velocity selection.

psp=1, $X_{ij} = X_{ij} + V'_j$ (24)

psp=2, $\left(X_{ij} = X_{ij} + V'_j\right) remainder\ 1$ (25)

$$X_{ij} = X_{ij} + V'_j$$
$$\text{psp=3, } if\ max(X_{ij}) > 1\ then\ X_{ij} = X_{ij}/1 \quad (26)$$

$$X_{ij} = X_{ij} + V'_j$$
$$\text{psp=4, } if\ X_{ij} < -pMax\ then\ X_{ij} = -pMax \quad (27)$$
$$if\ X_{ij} > pMax\ then\ X_{ij} = pMax$$

$$X_{ij} = X_{ij} + V'_j$$
$$\text{psp=5, } if\ X_{ij} < 0\ then\ X_{ij} = 0 \quad (28)$$
$$if\ X_{ij} > pMax\ then\ X_{ij} = pMax$$

If trail solution is better than current solution's local best then replace them. In addition, if trail solution is better than global best solution then set global best solution to trail solution.

$$if\ f(X_T) < f(P_{best,i})\ then\ P_{best,i} = X_T$$
$$if\ f(X_T) < f(G)\ then\ G = X_T \quad (29)$$

Then trail solution is added to new population *P'*, after processing of all solution on population *P*, *P* is set to *P'*. And we decrease value of *omega* parameter for next generation by using number of generation parameter *nog*.

$$omega = omega - 0.5/nog \quad (30)$$

Termination condition of PSO algorithm is a *noi* that is predefined parametric number of iterations.

3.7. Simulated Annealing Implementation

Steps of simulated annealing algorithm is on Figure 3.5.

1. Start with randomly generated solutions and initialize T with very high temperature.
2. Perturb the placement through a defined move.
3. Calculate the change of fitness value due to the move made.
4. Based on the change of fitness, accept or reject the move. Probability of acceptance related with the current temperature.
5. Decrease temperature value by decrease ratio.
6. Repeat 2-5 until the stopping criteria are met.

Figure 3.7. Steps of SA Algorithm

In our implementation of simulated annealing, algorithm starts with creation of current solution with random weights and best solution set to current solution. We used *maxR* constant 100000 to limit number of solutions parameter *R*.

Then neighbor solution of current solution is created by using value bound parameters *bound*, *radius* and with available 5 different values, neighbor selection parameter *nsp*. Our SA implementation uses exactly same neighbor creation function with MBO.

After creation of neighbor solution, *delta* calculated between fitness values of neighbor solution and best solution.

$$delta = f(X'_i) - f(X_i) \tag{31}$$

If cost of neighbor solution is less then cost of current solution, current solution is set to neighbor solution otherwise probability of acceptance is calculated as formula 33.

$$\begin{gathered} if\left(r < exp^{(|delta|/T)}\right) \\ then\ X_i = X'_i \\ where\ r\ is\ random\ number\ between \\ [0,1] \end{gathered} \tag{32}$$

If neighbor solution is better than best solution then best solution is set to neighbor solution. Creation of new neighbors repeats until variable *i* reaches parameter *R*. Then *T* decreased by parameter *a*, *R* increased by parameter *b* and *totalR* increased by *R*.

```
Generate cs, current solution randomly, cs = random()
Set best solution to current solution, bs = cs
maxR = 10000
while T > 0 and totalR < maxR
        for R times do
                ns = neighbor(bound, nsp, radius)
                delta = fitness value of ns – fitness value of cs
                if fitness value of ns > fitness value of cs then
                        cs = ns
                else if random() < exp^( |delta| / T)  then
                        cs = ns
                if fitness value of ns > fitness value of bs then
                        bs = ns
                end if
        end for
        T =T / a
        R =R * b
        totalR = totalR + R
end while
```

Figure 3.7. Implementation of Our SA Algorithm

Termination condition of SA algorithm is either temperature *T* parameter is 0 or number of solution parameter *totalR* reaches maximum limit of it *maxR*.

4. EXPERIMENTS AND RESULT ANALYSIS

In this thesis work, we worked on problem of how much money will be loaded each ATM of a bank i.e. forecasting of ATM cash withdrawal by using dataset obtained from a large Turkish bank. We implemented ABC, MBO, DE, PSO and SA for the problem in Java environment. Details of development environment and experimental setup explained in section 4.1 and parameter fine tuning details explained under section 4.

4.1.Experimental Environment

We implemented algorithms by using programming language Java and Java runtime environment JRE 1.8. Development IDE is Eclipse Oxygen.3 Release (4.7.3).

All meta-heuristics implemented are in an object-oriented design. We modeled objects like solution, population, ATM day data and meta-heuristic itself as Java classes.

Test server operating system is Windows 7 Professional 64-bit OS. Processor of machine is Intel ® Zeon ® CPU E5-1620 @ 3.60 GHz. And number of cores of processor is 4, number of threads of processor is 8. Installed memory RAM is 32 GB.

We run each test as the only running processor on the server.

4.2.Parameter Fine-Tuning

Best performing parameters of our implementations of the meta-heuristics are determined after a wide range of computational experiments. Each parameter set is run 5 times and their average costs are computed to determine best parameter set. All parameters can be seen on Table 4.1.

Table 4.21. Performing Parameter Tuples

ABC	*runtime* = {5,10} *maxCycle* = {5,10} *limi t*= {10,100} *FoodNumber* = {50,100,1000} fsp = {1,2,3}
MBO	*nob* = {5, 11, 21, 51} *nof = {5,10}* *non* = {3, 5, 7} *olf* = {1,2,3} *radius = {0.05, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1}* fsp = {1,2,3}
DE	*nog* = {50, 100, 200} *nos* = {50, 100, 200} *CR* = {0.1,0.5, 0.9} *F*= {0.1, 0.5, 1, 2} fsp = {1, 2, 3}
PSO	*nog* = {20000} nos = {50, 100} φ1 = {0.1, 0.5, 1, 2} φ2 = {0.1, 0.5, 1, 2} omega = {0.1, 0.5, 1} fsp = {1,2,3}
SA	*T* = {1000, 10000} *R* = {5,20} a = {1.1, 1.5} *b* = {1.1, 1.5} radius = {0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1} fsp = {1,2,3}

Computational tests are done with best parameters only for each *fsp*; MAPE, MAD and SMAPE with *noi* is 100000 thereafter best parameter sets are determined for ABC, MBO, DE,

PSO and SA. Comparison is made between results and each other based on their cost values given in next section. Best parameter values can be seen in Table 4.2.

Table 4.22. Best Performing Parameter Tuples.

	MAD	*MAPE*	*SMAPE*
ABC	*runtime* = 10 *maxCycle* = 5 *limit* = 100 *FoodNumber* = 1000	*runtime*= 10 *maxCycle* = 5 *limit*= 100 *FoodNumbe r* = 1000	*runtime* = 10 *maxCycle* = 5 *limit* = 100 *FoodNumbe r* = 1000
MBO	*nob* = 5 *nof = 5* *non* = 3 *olf* = 1 *radius* = 0.05 *bound* = 1 *nsp* = 5	*nob* = 5 *nof = 5* *non* = 3 *olf* = 1 *radius = 0.05* *bound* = 1 *nsp* = 5	*nob* = 5 *nof = 5* *non* = 3 *olf* = 1 *radius = 0.1* *bound* = 1 *nsp* = 5
DE	*nog* = 50 *nos* = 100 *CR*= 0.1 *F*= 0.5 *bound* = 1 *msp* = 5	*nog* = 200 *nos* = 100 *CR*= 0.1 *F*= 2 *bound* = 1 *msp* = 5	*nog* = 50 *nos* = 100 *CR*= 0.1 *F*= 0.5 *bound* = 1 *msp* = 5
PSO	*nog*= 20000 nos= 100 φ1 = 2 φ2= 0.5 *omega* = 1 *psp, vMax* = 1 *vsp, psp* = 1	*nog*= 20000 nos= 100 φ1 = 0.5 φ2= 0.5 *omega* = 0.5 *psp, vMax* = 1 *vsp, psp* = 1	*nog*= 20000 nos= 100 φ1 = 0.1 φ2= 2 *omega* = 1 *psp, vMax* = 1 *vsp, psp* = 1

SA	$T = 10000$ $R = 5$ $a = 1.1$ $b = 1.5$ $radius = 0.05$ $bound = 1$ $nsp = 5$	$T = 1000$ $R = 20$ $a = 1.1$ $b = 1.1$ $radius = 0.05$ $bound = 1$ $nsp = 5$	$T = 1000$ $R = 5$ $a = 1.1$ $b = 1.1$ $radius = 0.5$ $bound = 1$ $nsp = 5$

4.3.Results and Analysis

After finding best performing parameters of each meta-heuristic and for each fitness value calculation strategy, we run all algorithms with 10 ATM data and 100K number of instance and best performing parameters of each fitness strategy. Hence, we saw that how cost varies in same meta-heuristic by changing fitness calculation. Cost values can be seen on Table 4.3. and lower values indicate better forecast.

Table 4.33. Average Cost of Meta-heuristics with Different Measurements

Fitness Function	Avg. cost ABC	Avg. cost DE	Avg. cost MBO	Avg. cost PSO	Avg. cost SA
MAD	9,32	**0,54**	0,61	8,24	1,07
MAPE	113,51	**1,25**	15,73	136,83	38,60
SMAPE	1,48	**0,72**	0,97	1,51	1,45

We observed that with all MAD, MAPE and SMAPE calculations DE is the best performing algorithm and MBO is the second best with very near results. ABC is the worst for with MAD and PSO is the worst with MAPE and SMAPE.

In DE a population of solution vectors is repeatedly updated by addition, subtraction and component swapping until the population converges to an optimum. Moves in the search space are random and capacity of local search is high. Therefore, DE is suitable for problems

that are not continuous and change over time. Clearly DE outperforms other algorithms in our problem with our all ATM data including too many volatile day data.

MBO has strong feature such as benefit mechanism between successive solutions. If optimum solution of search space is near to bad solutions, MBO can catch optimum solution accepting bad solutions by using benefit mechanism. Some of generated neighbor solutions worse than current solution are shared with following solution. Following solution uses these bad solutions to improve itself so bad solutions are not lost. This mechanism gives MBO high search capacity like DE and helps to escape local minima. As we can see from our test results MBO performs good as well as DE.

However, ABC and PSO suffer lack of such benefit mechanism. Both algorithms usually use good solutions to generate new solutions. They can converge prematurely and be trapped into a local minimum especially with complex problems.

Thereafter we look through small number of *noi* results to see how algorithms converge. For same ATM data with MAD, best parameters and *noi* 1K, 10K, 20K, 50K, 100K costs are on Figure 4.1. DE started with a bad cost but it arrived best cost after 50K instances. MBO and SA started with nearly same costs and MBO outperformed SA after 10K instances and continued to improve until 50K as well as DE. As previously discussed, ABC and PSO started with bad costs and continues with same results. By nature of their implementation, they are not suitable for our problem and dataset.

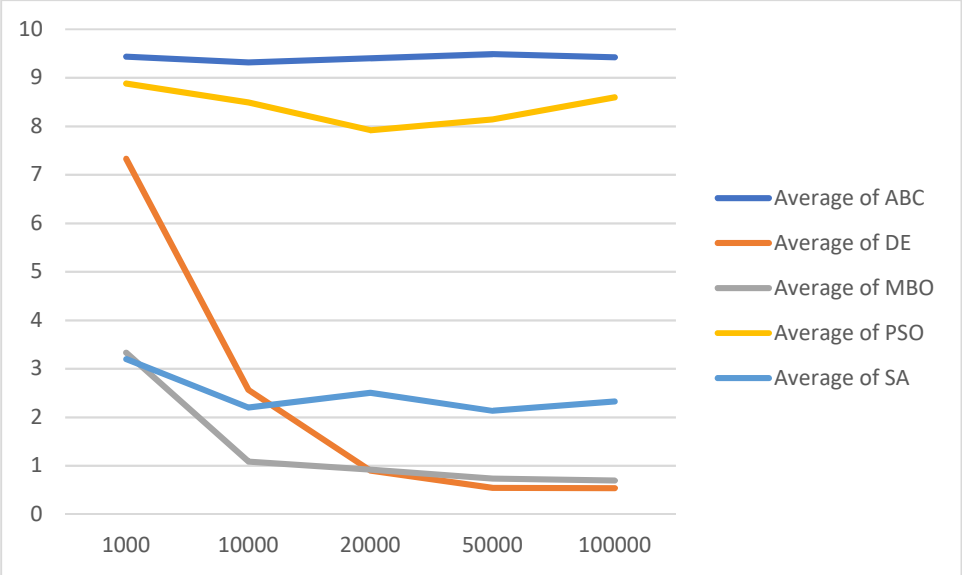

Figure 4.3 Comparison of Convergence

We made more experiments on same ATM data by increasing maximum number of instances allowed, with MAD as fitness function and best parameters of each algorithm for MAD. Comparison of meta-heuristics average cost values while *noi* increasing from 50K to 500K can be observed on Figure 4.2 and Table 4.4.

Table 4.34. Average Costs at Increasing Instances with MAD

noi	Avg. of ABC	Avg. of DE	Avg. of MBO	Avg. of PSO	Avg. of SA
50000	9,20	0,41	0,50	8,06	3,07
100000	8,87	0,40	0,47	7,49	2,89
150000	8,73	0,40	0,45	7,46	2,93
200000	9,02	0,40	0,45	7,64	2,86
250000	9,00	0,40	0,46	7,96	3,16
300000	9,04	0,40	0,44	7,78	2,84
350000	9,08	0,40	0,44	7,72	2,79
400000	8,93	0,40	0,46	7,82	3,14
450000	8,90	0,40	0,45	7,20	2,81
500000	8,76	0,40	0,45	7,95	3,08
Grand Total	8,95	0,40	0,46	7,71	2,96

DE and MBO outperform other algorithms again for every number of instances we tried. We expect that search algorithms perform good results if they iterate a lot but there is a threshold and all algorithms that we implemented arrived their threshold before 50K. When the iteration number increases, indeed we cannot say the cost decrease or increase for all algorithms. There is no clear change on cost values.

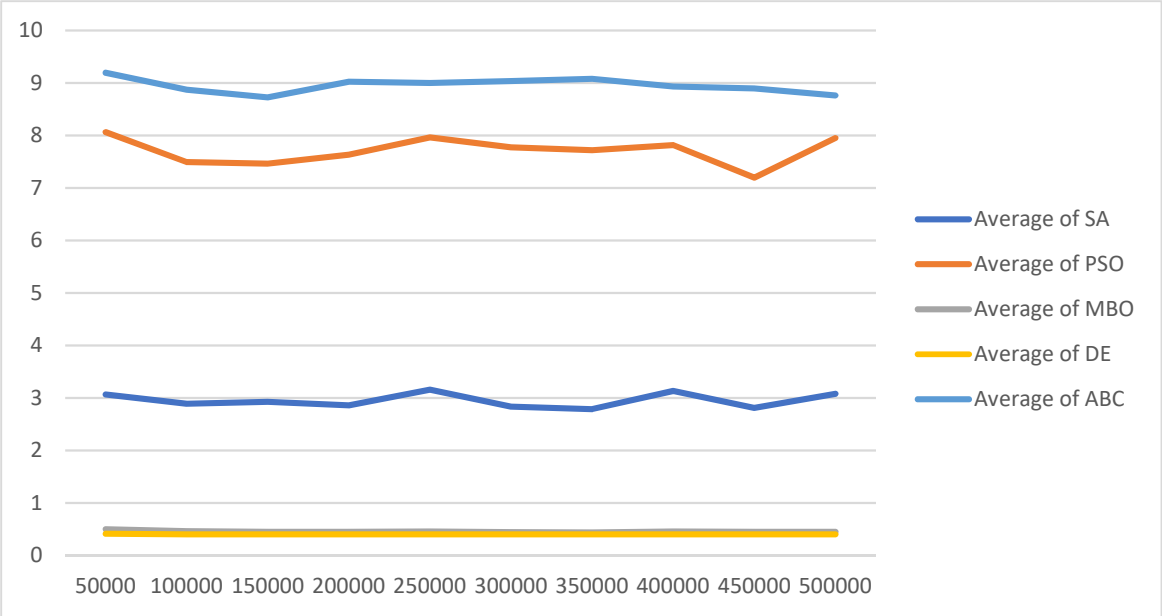

Figure 4.3 Comparison of Cost

Then we decided to separate our ATM data into two groups volatile and non-volatile ATMs. Tests run for 5 non-volatile ATM data with best parameters and *noi* 1K, 10K, 20K, 50K, 100K and MAD as fitness value calculation. Results can be seen on Table 4.5.

Table 4.35. Average Cost with Non-volatile Data

noi	Avg. ABC	Avg. DE	Avg. MBO	Avg. PSO	Avg. SA
1000	4,09	**3,54**	**1,83**	3,97	1,57
10000	4,18	**1,29**	**0,68**	4,12	1,05
20000	4,11	**0,66**	**0,64**	4,18	1,17
50000	4,11	**0,58**	**0,62**	3,83	1,18
100000	4,13	**0,58**	**0,61**	4,25	1,16

Tests run for 5 volatile ATM data with best parameters and *noi* 1K, 10K, 20K, 50K, 100K and MAD as fitness value calculation. Results can be seen on Table 4.6.

Table 4.36. Average Cost with Volatile Data

noi	Avg. ABC	Avg. DE	Avg. MBO	Avg. PSO	Avg. SA
1000	14,78	**11,12**	**4,83**	13,80	4,83
10000	14,45	**3,84**	**1,49**	12,86	3,34
20000	14,70	**1,13**	**1,20**	11,66	3,85
50000	14,86	**0,51**	**0,85**	12,45	3,09
100000	14,71	**0,50**	**0,78**	12,94	3,49

We observed that still DE and MBO are best performing algorithms for both volatile and non-volatile data. As we expected all algorithms start with better values for non-volatile data comparing with volatile data results.

For ABC and PSO there is huge difference between volatile results and non-volatile results. After 100K average cost for ABC is 14,71 with volatile data and it is 4,13 with non-volatile data. Similarly, after 100K solutions PSO average is 12,94 with volatile data and it is 4,25 with non-volatile data. Results get nearly 3 times worse with volatile data for ABC and PSO.

On the contrary, DE and MBO have nearly same good results for volatile data too. After 100K solutions DE has average 0,50 for volatile and 0,58 for non-volatile data, MBO has average 0,78 for volatile and 0,61 for non-volatile data.

SA has better results than comparing with ABC and PSO but not good as DE and MBO. SA also is capable of escaping from local minimum by allowing hill-climbing moves. Improving solutions are always accepted, while a number of non-improving solutions is accepted with a probability depends on a decreasing temperature parameter. However, for our data SA cannot compete with DE or MBO.

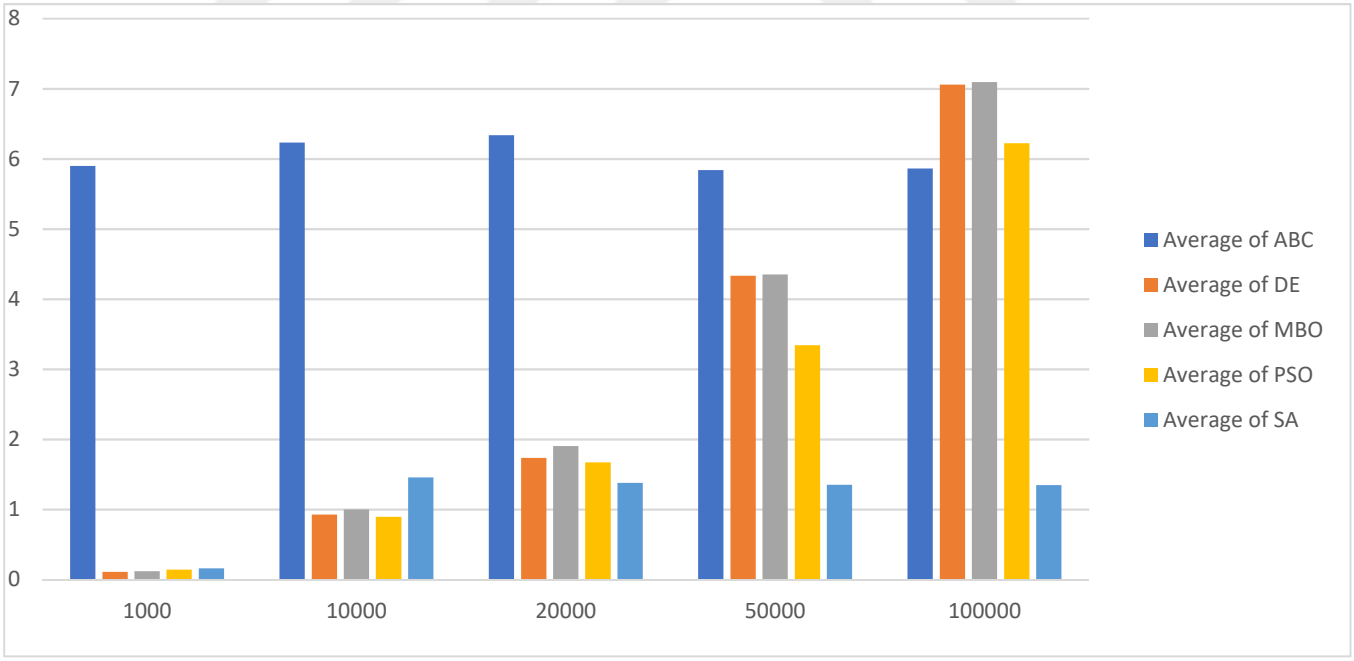

Figure 4.3. Comparison of Average Execution Time

If we compare runtime duration of algorithm executions SA outperforms all algorithms for each 1K, 10K, 20K, 50K, 100K solutions and the worst one, the slowest one is ABC algorithm on average.

However, while number of solutions are increasing runtime of MBO and DE are also increasing and they arrive the maximum execution time at 100K solutions. Comparison graphic can be seen on Figure 4.3. For 100K solutions, SA has 1.35 seconds execution time on average while DE and MBO have average 7 seconds execution time. Details are on Table 4.7. If we have constraint on execution time, SA may be a good choice with average estimate accuracy and fast runtime.

Table 4.7. Average Execution Time in seconds

noi	Avg. ABC	Avg. DE	Avg. MBO	Avg. PSO	Avg. SA
1000	5,90	0,11	**0,12**	0,14	**0,16**
10000	6,23	0,93	**1,00**	0,90	**1,46**
20000	6,34	1,74	**1,90**	1,67	**1,38**
50000	5,84	4,33	**4,35**	3,34	**1,35**
100000	5,86	7,06	**7,10**	6,22	**1,35**

5. CONCLUSION

In this study, we compared the performance of five algorithms with each other in the ATM forecasting problem. The problem emerges from the need of banks to determine when to visit and how much to load on each ATM. The primary question that needs to be answered turns out to be forecasting how much money will be withdrawn from the ATMs in the next days.

We implemented ABC, MBO, PSO, DE and SA algorithms to be used for forecasting. In this study, firstly the aforementioned algorithms are implemented by designing the operators of the algorithms by considering the nature of the problem. We implemented a number of different solution generation methods. Then, parameters of the algorithms are fine-tuned with computational tests by using three different error measurements MAD, MAPE and SMAPE.

In the last phase, all meta-heuristics are applied to the problem instances with their best performing parameter values and compared through extensive computational experiments. Results show that MBO and DE provide comparable best costs for forecasting using volatile ATM data. As a future work, the comparison study may include some machine learning techniques such artificial neural networks, recurrent neural networks and a hybrid of all of these.

6. REFERENCES

[1] Global ATM installed base to reach 4M by 2021. https://www.atmmarketplace.com/news/global-atm-installed-base-to-reach-4m-by-2021/ (accessed on 06.04.2018

[2] Saad M. Darwish, A Methodology to Improve Cash Demand Forecasting for ATM Network. International Journal of Computer and Electrical Engineering, Vol. 5, No. 4, August 2013.

[3] S. Crone, Time Series Forecasting Competition for Computational Intelligence, http://www.neural-forecastingcompetition.com (2008).

[4] R.R. Andrawis, A.F. Atiya, H. El-Shishiny, Forecast Combinations of Computational Intelligence and Linear Models for the NN5 Time Series Forecasting Competition, International Journal of Forecasting, 27, (2011) 672–688.

[5] K. Venkatesh, R. Vadlamani, A. Prinzie, D. Van den Poel, Cash demand forecasting in ATMs by clustering and neural networks, European Journal of Operational Research 232 (2) (2014) 383-392.

[6] R. Simitus, D. Dilijonas, L. Bastina, J. Friman, "A Flexible Neural Network for ATM Cash Demand Forecasting", 6th WSEAS Int. Conference on Computational Intelligence, Man-Machine Systems and Cybernetics, Tenerife, Spain, December 14-16, 2007.

[7] T. Baker, V. Jayaraman, N. Ashley, A Data-Driven Inventory Control Policy for Cash Logistics Operations: An Exploratory Case Study Application at a Financial Institution, Decision Sciences, 44 (1) (2012) 205-226.

[8] P. Kumar and E. Walia, "Cash Forecasting: An Introduction of Artificial Neural Networks in Finance," International Journal of Computer Sciences and Applications, vol. 3, no. 1, pp. 61-77, 2006

[9] M.S. Kıran, E. Özceylan, M. Gündüz, T. Paksoy, A novel hybrid approach based on particle swarm optimization and ant colony algorithm to forecast energy demand of Turkey. Energy conversion and management, 53 (1) (2012) 75-83.

[10] R. Rahmani, R. Yusof, M. Seyedmahmoudian, S. Mekhilef, Hybrid technique of ant colony and particle swarm optimization for short term wind energy forecasting, Journal of Wind Engineering and Industrial Aerodynamics 123 (2013) 163-170.

[11] A.P. Piotrowski, M. Osuch, M.J. Napiorkowski, P.M. Rowinski, J.J. Napiorkowski. Comparing large number of metaheuristics for artificial neural networks training to predict water temperature in a natural river, Computers & Geosciences, 64 2014 136-151.

[12] J. Behnamian, S.M.T. Fatemi Ghomi, Development of a PSO–SA hybrid metaheuristic for a new comprehensive regression model to time-series forecasting, Experts Systems with Applications, 37 2010 974-984.

[13] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony algorithm, Journal of global optimization 39 (3) (2007) 459-471.

[14] ABC Algorithm Homepage, https://abc.erciyes.edu.tr/, last accessed 2019/04/13.

[15] Awan, Shahid & Aslam, Muhammad & Khan, Zubair & Saeed, Hassan. (2014). An efficient model based on artificial bee colony optimization algorithm with Neural Networks for electric load forecasting. Neural Computing and Applications. 25. 1967-1978. 10.1007/s00521-014-1685-y.

[16] Bai Li, "Research on WNN Modeling for Gold Price Forecasting Based on Improved Artificial Bee Colony Algorithm," Computational Intelligence and Neuroscience, vol. 2014, Article ID 270658, 10 pages, 2014.

[17] Supreetha, B. S. and Nayak, Prabhakar K. and Shenoy, Narayan K. (2019) Hybrid Artificial Intelligence Based ABC-PSO System for Ground Water Level Forecasting in Udupi Region. Journal of Engineering Science and Technology, 14 (2). pp. 797-809.

[18] R. Storn, K. Price, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Space," Journal of Global Optimization, 11 (1997) 341-359.

[19] Jason Brownlee, Clever Algorithms: Nature-Inspired Programming Recipes, Evolutionary Algorithms, Differential Evolution (2011).

[20] Satyendra Nath Mandal, Jit Saha. Application of Differential Evolution Algorithm in Prediction of Time Series Data. International Journal of Innovative Research in Science, Engineering and Technology, Vol. 5, Special Issue 13, October 2016.

[21] E. Duman, M. Uysal,A.F. Alkaya, Migrating Birds Optimization: A New Metaheuristic Approach and Its Performance on Quadratic Assignment Problem, Information Sciences, 217 (2012) 65-77.

[22] Duman E., Elikucuk I. (2013) Applying Migrating Birds Optimization to Credit Card Fraud Detection. In: Li J. et al. (eds) Trends and Applications in Knowledge Discovery and Data Mining. PAKDD 2013. Lecture Notes in Computer Science, vol 7867. Springer, Berlin, Heidelberg.

[23] Alkaya, A. F., & Algin, R. (2015). Metaheuristic based solution approaches for the obstacle neutralization problem. Expert Systems with Applications, 42(3), 1094-1105. (SCI Expanded)

[24] Muyesaier AIHAITI, Ali Fuat ALKAYA and Ramazan ALGIN, An Empirical Comparison of Data Mining Tools and Migrating Birds Optimization Algorithm on Medical Diagnosis. International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES'18), May 11-13, 2018 Safranbolu, Turkey.

[25] J. Kennedy, R.C. Eberhard, Particle swarm optimization. Proc. of IEEE International Con-ference on Neural Networks. Piscataway, NJ, USA, (1995) 1942-1948

[26] R. Poli, J.Kennedy, T. Blackwell, Particle swarm optimization, An overview. Swarm Intell (2007) 1: 33–57 DOI 10.1007/s11721-007-0002-0.

[27] Essam El. Seidy, A New Particle Swarm Optimization Based Stock Market Prediction Technique, (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 7, No. 4, 2016

[28] Sirajul Islam, Bipul Talukdar, Performance improvement of a Rainfall Prediction Model using Particle Swarm Optimization, International Journal of Computational Engineering Research (IJCER), Volume 06, Issue 07, July 2016.

[29] D. Bertsimas, J. Tsitsiklis, Simulated Annealing, Statistical Science 1993, Vol. 8, No. 1, 10-15.

[30] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 35, 268–308.

[31] Song, Jingwei & He, Jiaying & Zhu, Menghua & Tan, Debao & Zhang, Yu & Ye, Song & Shen, Dingtao & Zou, Pengfei. (2014). Simulated Annealing Based Hybrid Forecast for Improving Daily Municipal Solid Waste Generation Prediction. The Scientific World Journal. 2014. 834357. 10.1155/2014/834357.

[32] González-Mancha, J., Frausto-Solís, J., Castilla Valdez, G., Terán-Villanueva, J., & González Barbosa, J. (2018). Financial time series forecasting using Simulated Annealing and Support Vector Regression. International Journal of Combinatorial Optimization Problems and Informatics, 8(2), 10-18.