Int j simul model 18 (2019) 2, 205-216
ISSN 1726-4529 Original scientific paper

ECONOMIC LOT-SIZE USING MACHINE LEARNING, PARALLELISM, METAHEURISTIC AND SIMULATION

Sousa Junior, W. T. de[*,#]; Montevechi, J. A. B.[**]; Miranda, R. de C.[**]; Rocha, F.[**] & Vilela, F. F.[**]

[*] Mechanical Engineering Department, Federal University of São João del Rei, São João del Rei, Brazil
[**] Institute of Production Engineering and Management, Federal University of Itajubá, Itajubá, Brazil
E-Mail: wilson.trigueiro@ufsj.edu.br, montevechi@unifei.edu.br, rafael.miranda@unifei.edu.br, fernandarocha@unifei.edu.br, ffvconsultor@gmail.com ([#] Corresponding author)

Abstract
The use of discrete event simulation optimisation methods is a tool commonly used as a decision-making support system in industrial problems, concerning management and resource allocation in order to maximise a set of values regarding costs, revenues and other enterprise interests. The present study has proposed and tested an optimisation algorithm developed on Python, with different wall clock time reduction strategies including parallelism, the Greedy Randomized Adaptive Search Procedure (GRASP) population-based metaheuristic, and ten machine learning methods. With the selected best machine learning method (Decision Trees Regressor) 6 optimisation scenarios were generated and then applied to an economic lot-size problem for a theoretical shop floor. The results showed improvements in the reduction of the processing time of 95.0 % comparing the serial GRASP with the parallel machine learning GRASP, obtaining a solution of 94.0 % of the best local optimum.
(Received in September 2018, accepted in March 2019. This paper was with the authors 2 months for 1 revision.)

Key Words: Optimisation, Economic Lot-Size, Machine Learning, Parallelism, Metaheuristic, Discrete Event Simulation

1. INTRODUCTION

One of the primary goals of any company in order to obtain stability and growth in the market is to find the best alternatives for investment regarding production costs and overall related services [1]. In many cases, the use of analytic tools helps in the process to find opportunities considering a large number of variables and scenarios [2-4].

A well established and used set of practices are related to the operations research area, regarding discrete event simulation and optimisation methodologies applied on data treatment, capture patterns and scenarios evaluation to aid in the decision-making process [5, 6]. In that sense, the metaheuristic class of optimisation techniques has been used to solve industrial problems in terms of production, logistics, related services and other issues [7-9].

With the advent and development of industry 4.0, a wide range of data is being generated and stored with cost savings related to capturing, transmitting, storing and processing data. In light of the last issue, parallel processing is a technique that has been used, considering the latest advances in the hardware matter [8, 10, 11]. For this volume of data to generate a direct benefit for the industries, techniques are needed for aid in their analysis. This analysis must be done in order to find patterns and transform them into useful information [12, 13]. As an example of such techniques, it is possible to identify the machine learning (ML) as sufficiently robust methodology on data treatment, capture patterns and evaluate the scenarios to aid in the decision-making process [14].

The present study contributes to the stochastics optimization research field proposing and testing a new open source framework developed in Python 3.6. The framework was applied in an industrial matter of economic lot-size definition, putting together the concepts of parallel computing, metaheuristics and machine learning, benefiting from all of these concepts to generate a novel approach tool to help the decision-making on an economic lot-size problem,

and discussing their construction and evaluation. At the moment, papers were identified that work with these concepts separately but not together, *e. g.* [8, 15, 16].

The remainder of the paper is organised as follows: Section 2 presents methods used for the optimisation algorithm, Section 3 describes the experimental work to test the methodology, in Section 4 are results and discussion, and Section 5 gives conclusions with future directions.

2. METHODS USED

The methodology used in the present study consists of the selection of the most suitable ML techniques for selection, test and ranking inside of a metaheuristic optimisation with parallel evaluations. In that way, the following steps were used for selection of the tested methods.

2.1 Machine learning selection, test and ranking

A search in the Web of Science and Scopus databases did not find any significant results regarding metaheuristic with machine learning applied to economic lot-size problems. Therefore, no studies were found as a base for which would be the most suitable machine learning in this case.

To evaluate the best machine learning techniques, a set of 10 methods were selected, representing a sample for the most used in the literature [17, 18]. Table I shows the considered ML families and the tested methods.

Table I: Machine learning methods tested.

Family	Method	Parameters used
Generalized Linear Models	Orthogonal Matching Pursuit (OMP)	n_nonzero_coefs = 3
	Polynomial Regression (PR)	('poly', PolynomialFeatures(degree = 3)), ('linear', LinearRegression (fit_intercept = False))
Linear and Quadratic Discriminant Analysis	Linear Discriminant Analysis (LDA)	solver = "svd", store_covariance = True
Naive Bayes	Multinomial Naive Bayes (MNB)	----------
Decision Trees	Decision Trees Classifier (DTC)	max_depth = None, min_samples_split = 2, random_state = 0
	Decision Trees Regressor (DTR)	----------
Ensemble methods	Random Forest Classifier (RFC)	n_estimators = 10
	Random Forest Regressor (RFR)	max_depth = 2, random_state = 0
	Extra Trees Classifier (ETC)	n_estimators = 10, max_depth = None, min_samples_split = 2, random_state = 0
	Gradient Tree Boosting Regressor (GTBR)	n_estimators = 100, learning_rate = 1.0, max_depth = 1, random_state = 0

The methods presented in Table I, which demand parameters to be specified by the user, were selected as the default ones presented in the Python package SciKit-learn. The use of these default values is due to the primary objective of the present study to identify the most suitable methods to be used and to give good insight regarding the search direction in the solution space, not to adjust a specific method to the best parameters that will set it for a better prediction. A methodology to adjust specific ML was chosen to be presented in future work.

The following steps were applied to generate, treat and do analyses of data to the proposed problem in the first phase in order to determine the ML technique to be used:

1) Setting a data frame. The problem's solution space of the problem is defined by 40,585,181 possible solutions, presented in Section 3.1. To determine a representative sample, 2,000 random solutions without repetition (0.005 % of the solution space) were selected to represent the variability of the problem and they can be generated in a reasonable time (1.5 hours) in the presented optimisation algorithm.
2) Generating data to train the method. To apply an ML method, it is necessary to have data specified into classes to make the relative frequency calculation. For the proposed problem, as in general for all discrete event simulation problems, the results are discrete and they are not evenly distributed. It originally generates many classes that prohibit a direct use of ML. To work around this issue, the results were classified as multiples of 300,000 as a reference that any solution that is inside this interval would have the same attractiveness for the decision-making agent [19].
3) Prediction and method ranking. With the data frame composed by the 2,000 solutions, 80 % (1,600) were set for the training, and 20 % (400) were separated for the evaluation of the predictions/classifications against the real values. To compare the 10 methods, the following criteria were used: confidence interval (*CI* with α = 5.0 %) for the difference between the real value and the predicted value; the minimum and maximum values generated for the error between the predicted and real ones; the Mean Absolute Error (*MAE*), the Mean Squared Error (*MSE*); and the wall clock time necessary for the training and prediction. In order to rank the best results, the values of *MSE* and *CI* were considered since the fact that the most desirable method should have the minimum total error on prediction.

2.2 Selection of optimisation techniques set

In operations research, many discrete event simulation optimisation methods were used successfully. Examples of this include heuristics, metaheuristics, meta-models (surrogate models), ranking and selection, complete enumeration and black-box (proprietary software) [20-24]. The selection of a method is related to the solution space size and knowledge of how the variables interact with each other. In our case study, searching the entire solution space is not practical due to the necessity of at least three years of processing time on the tested hardware.

With that in mind, the metaheuristic optimization methodology was chosen because, according to the aforementioned authors, these optimisation methods consider stop criteria that involves some restriction on its fundamental elements. That is true, for instance, with the total wall clock time or iterations without improvement. Another element is the generation of a number of solutions by each iteration of the search algorithm that is oriented by the method proposed. These two characteristics comply with the desirable characteristic of the present study in order to walk on the solution space in a parallel way.

Modern computers (from the year 2003 on) can process more than one instruction in parallel with the advent of multi-core processors on personal computers [25]. However, not all of the programs can take advantage of this method but rather this depends on the previous knowledge and the possibility of distributing a study with no or low dependency on previous data from the program that justifies the use of parallelism [26]. On the optimisation way, it is possible to use the parallelism [27] to evaluate multiple scenarios at the same time [8]. For this purpose, the Greedy Randomized Adaptive Search Procedure (GRASP) was chosen as the metaheuristic that uses population search for the next solution generation phase and walks in the solution space [28].

3. EXPERIMENTAL WORK

The experimental section was divided into the explanation of the study object, the set of the computer environment and the description of the optimisation algorithm, used for the present study.

3.1 The study object

The problem used for testing the optimisation framework is a theoretical variety for the economic lot-size presented by [29, 30]. Orders arrive electronically and wait following a FIFO (first in first out) order for one of the two employees to attend. Each order requests a number of products and, if it is available, the stock is depleted, and the order ended. If there is not enough products in stock, the order waits until the end of the replenishment process. A third employee is responsible for checking the stock between a defined period and generating a replenishment according to a defined economic lot-size for one year to end the simulation and ten replications to minimise the variance of the results. The authors decided the values for the variables to have a stochastic distribution, defined in Table II, and present the data characteristic of been Identical and Independently Distributed because these were characteristics more suitable to be resolved by discrete event simulation body of knowledge.

Table II: Statistical distribution for the stochastic variables on the study object.

Activity	Probability distribution (seconds)
Order arrival	432 + Normal (mean = 36, std deviation = 10) – Exponential (lambda = 2)
Order demand	6 + Normal (mean = 4, std deviation = 1)
Service time	(Order demand) / (3 + Normal (mean = 2, std deviation = 1))
Replenish time	271 + Exponential (lambda = 10) – Exponential (lambda = 2)) × lot_{rep}

The answer to the problem consists of determining the economic lot-size, the security level that triggers a new order and the review period in order to check the *lot* level. All of these parameters are a part of the objective function (OF) and restrictions presented in Eqs. (1) to (4), to maximise the net revenue. The parameter $wait_{time}$ is given by the sum of total waiting time that the current solution generates for the order between the arrival and exit from the simulation.

$$\max F(lot_i, sec_j, rev_k) = \left[-1{,}149{,}557 - 1{,}621 \times lot_i - 154 \times \sum wait_{time} + 1.8 \times sec_j - 3.9 \times {rev_k}^2 + 83.4 \times lot_i \times \sum wait_{time} - \left[\left((0.22 \times lot_i)^{\left(\frac{3}{\sum wait_{time}}\right)}\right) + 1\right] - \left(0.0109 \times sec_j\right)^2 - 221{,}000{,}000 \times (lot_i)^2\right] / 45{,}683{,}000 \quad (1)$$

s.t.

$$lot_i = \{100; 101; \ldots; 5{,}000\}: \forall i \in lot \quad (2)$$

$$sec_j = \{5; 6; \ldots; 95\}: \forall j \in sec \quad (3)$$

$$rev_k = \{1{,}800; 2{,}100; 2{,}400; \ldots; 28{,}800\}: \forall k \in rev \quad (4)$$

Eq. (1) summarizes all of the expenses and revenues related to sales, acquisition, holding and generation of a replenish order and stock. According to Eqs. (2) to (4), all of the variables are integer, with lot_i and sec_j related to the economic lot-size (measured in units) and security level (in percentage) respectively, being evaluated with the addition of one element and rev_k the time between the checks on the stock level, with a difference of 300 seconds between solutions. With that presented, the problem has a solution space of $4{,}901 \times 91 \times 91 = 40{,}585{,}181$ possible scenarios.

With the distributions in Table II and Eqs. (1) to (4), it was possible to generate a space solution were the best local optimum is located in a central region. That fact permits to test if the optimisation algorithm has enough energy to pass and scape radial regions with local points of minimum, maximum and saddle.

3.2 The programming environment

The simulation and optimisation algorithm used open source packages, found in the official repositories constructed using the following:

- Programming environment in Python 3.6.5 x64, which has a built-in MapReduce function for parallel processing of the simulation scenarios;
- For the discrete event simulation, the SimPy 3.0.10 environment was selected, which allows the problem parameters to modify from the code that is generated in the optimisation that has already been tested and used in engineering problems [31, 32].
- To make the machine learning data frame, training and prediction possible, the following packages were used: NumPy 1.14.3, SciKit-Learn 0.19.1, Pandas 0.22.0 and SciPy 1.0.1. These packages are used in studies for machine learning using Python language and already have ML available for testing [18, 33]; and
- The graphical representation of the results with MatPlotLib 2.2.2.

In our approach to program the optimisation, two algorithms were constructed of which the first was used to build the initial populations and the primary data gathering for the ML data frame, analysis and training and the last one for serial, parallel and ML optimisation.

3.3 The optimisation algorithm

The optimisation algorithm was built in two different phases. The first is related to the initial generation of the population for the purpose of training the selected ML and generating individuals for the optimisation, presented in Algorithm 1.

Algorithm 1: Initial population and data frame for ML training

1 $\boldsymbol{economicLotSizeSimul} \leftarrow (rand.seed, lot_i, sec_j, rev_k)$
2 **while** $t \leq T_{max}$ **do**
3 $procesTime \leftarrow rand.distribution$ % for each stochastic variable according to Table II
4 $\sum (wait_{time}\ at\ time\ t)$
5 $t \leftarrow t + timeAdvanceNextEvent$
End
return Objective Function according to Eq. (1)
% Phase 2: Construction of the initial population
6 **for each** $vecPop$ **do**
7 $vecPop \leftarrow rand(H/L\_lot_i) + rand(H/L\_sec_j) + rand(H/L\_rev_k)$
End
% Phase 3: Evaluation function for the initial population
8 **for each** $vecPop$ **do**
9 $resultPop \leftarrow economicLotSizeSimul(vecPop)$
End
% Phase 4: Data transformation for initial machine learning training
10 **for each** $vecPop$ **do**
11 $dataFrameML \leftarrow resultPop\ MOD$ 300,000 % Transform data into classes
12 $markFrameML \leftarrow vecPop$
End
% Phase 5: Sort the population and set stop criteria
13 $vecPop \leftarrow sortMaxToMin(vecPop, resultPop)$
14 $iterMax \leftarrow 16$
15 $iter \leftarrow 0$

In Algorithm 1, the first phase requires building a discrete event simulation model (lines 1-5) that can recursively be called to construct the initial population and evaluate the selected scenarios for the next generation. In particular, the simulation model was developed to run scenarios on the serial or parallel ways. During the second phase (lines 6-7), an initial population was set to generate 600 random solutions respecting a combination of low and high levels for the variables of the problem without repetition. This arrangement was set in order to give the ML method an initial data frame that has a good representation of the solution space with a more even, sparse distance between the generated solutions. As a consequence, the ML has a better chance of generating predictions near the real values.

The third phase is the solutions' evaluation (lines 8-9). This evaluation can be done through either serial or parallel ways, according to the previous setting. The simulation call is the only part deliberately executed in this manner because this is the most computer power demanding program function, as a time consequence, in order to be processed. The fourth phase is the construction and training for the ML method (lines 10-12). To reduce the variance of the evaluation function returned by the simulation call before it enters the data frame, a data transformation is applied in which the result of the simulation is substituted by the quotient of the division by 300,000. The chosen value is determined considering a difference smaller than that value between results has the same attractiveness from a decision-making point of view. This data transformation is necessary to create classes and that makes it possible to use the selected ML techniques.

The Algorithm 1 last phase is sorting the solution vector for the initial population according to the corresponding values stored in the results vector, which decreases the value of the results (line 13). The last two functions (lines 14, 15) set the optimization process stop criteria parameters. The optimization search was set to stop when 16 consecutive iterations are generated without improvement of the best solution. With the stop criteria defined, the performance evaluation of the algorithms is related to the comparison of time and quality of the solution according to the best result found in Algorithm 2.

For the GRASP optimisation, the pre-processing phase is considered in Algorithm 1, in which the evaluation function is created and applied within the population. In Algorithm 2, the first phase, being constructive, requires building the restricted candidate list for selection. For this purpose, an $\alpha = 0.8$ was chosen to represent that the first 80 % of the individuals sorted in the initial population generated at the end of Algorithm 1 are candidates that make up the restricted list. Thus, three were selected to generate a new solution with the first variable of the first selection with the second variable of the second selection and so on. With the offspring in phase one, a neighbourhood is generated that evaluates five levels for each variable (+2, +1, 0, -1, -2) with a total of 125 different combinations or less when a level reaches the limit range of a variable. This is presented in Eqs. (2) to (4). In this case, the solution receives the value of its limit.

The third phase is the evaluation of the neighbourhood, with and without ML constituting the six scenarios tested. When the evaluation is made without ML, it can be a serial or parallel call of the simulation, Phase 3.1 in Algorithm 1, forming scenario 1 and 2. When done with ML, 4 levels were tested with 125 (×1), 1,250 (×10), 12,500 (×100) and 125,000 (×1,000) predictions in order to select the best 40 with further evaluations using the parallel simulation function (scenarios 3 to 6). After the evaluation, the data frame is updated with the simulated values, and the ML is adjusted for better learning and prediction in the next generation. The fourth phase is the selection of the best solution and stored on a list to compare it with the best current solution, and finally, count if any improvement is generated compared to the best solution and if the stop criteria is reached to end the optimisation.

Algorithm 2: GRASP with and without machine learning or parallel processing

```
1   while iter ≤ iterMax do
2       for each numbOfIterations do
            % Phase 1: Constructive
3           a, b, c ← restrictList(vecPop)
4           offspr1 ← buildVec(a, b, c)
            % Phase 2: Build a partial solution
5           for each numbOffspr do
6               tempVec ← tempVec + neighborhood(offspr1)
            end
        end
        % Phase 3.1: Evaluation of partial solution without machine learning
7       for each tempVec do
8           tempRes ← economicLotSizeSimul(tempVec)
        end
        % Phase 3.2: Evaluation and next generation with machine learning
9       for each tempVec do
10          tempResult ← machLearnPredict(tempVec)
        end
        % Phase 3.2.1: Data transformation to improve machine learning prediction
11      for each tempResult(best) do
12          dataFrameML ←
            dataFrameML + economicLotSizeSimul[tempResult(best)]MOD 300,000
13          markFrameML ← markFrameML + tempVec
        end
        % Phase 4: Find the best solution and iteration count
14      starResultBefore ← tempResult(best)
15      vecPop ← sortMaxToMin(tempVec, tempResult)
16      starResultAfter ← tempResult(best)
17      bestResList ← bestResList + tempResult
18      if starResultAfter ≥ starResultBefore do
19          iter ← iter + 1
        end
    end
```

The hardware chosen for the experiments was a Supermicro X9DAi with dual Intel Xeon E5-2620 running a full load on 2.29 GHz for all cores and 2.48 GHz on a single core with a total of 12 cores, 24 threads and 96 GB of RAM.

4. RESULTS AND DISCUSSION

In Table III, the results for the application of the 10 ML methods applied to the problem present in item 3.1 following the steps shown on item 2.1, are presented along with the corresponding evaluation parameters.

According to Table III, the results are presented respecting the decreasing order for preference in the quality of the predictions, mainly as evaluated by the Mean Squared Error and Confidence Intervals for the error on the predictions. The best method for prediction in the evaluated problem was the DTR (Decision Trees Regressor) with the lowest levels in all parameters except the maximum error (Max), in particular, related to *MSE* and *CI* that are the main evaluation points for ranking the ML methods for the present study. Compared with the second-best ML, the DTR had values of 94.7 %, -17.8 %, 45.9 %, 68.4 %, 0.0 %, 89.5 % and 0.0 % for Min, Max, *MAE*, *MSE*, *T-Train* and *T-Predict* respectively.

Fig. 1 illustrates the boxplot set for the error and the prediction data on the ten methods presented in Table III.

Table III: Results for the 10 ML techniques test.

Method	**Min**	**Max**	***MAE***	***MSE***	R^2	***T-Train***	***T-Predict***	***CI* (95 %)**
DTR	-350,000	4,830,000	76,400	9,047,867	0.99	0.02	< 0.00	(-1.91 ; 4.93)
DTC	-6,550,000	4,100,000	141,300	28,649,633	0.99	0.19	< 0.00	(-7.47 ; 4.71)
ETC	-410,000	7,430,000	194,367	57,637,833	0.98	0.83	0.03	(6.32 ; 23.27)
RFC	-9,750,000	9,390,000	225,033	80,217,967	0.97	0.92	0.02	(-1.08 ; 19.20)
GTBR	-3,605,023	1,561,735	292,795	27,074,264	0.99	0.05	< 0.00	(-11.41 ; 0.37)
PR	-3,065,231	1,086,772	337,690	24,519,506	0.99	0.02	< 0.00	(-5.01 ; 6.26)
LDA	-4,080,000	5,910,000	518,733	85,233,667	0.97	0.13	< 0.00	(4.42 ; 25.16)
MNB	-6,440,000	6,550,000	724,233	144,621,367	0.95	0.05	<0.00	(26.78 ; 52.62)
RFR	-9,096,776	3,242,486	872,532	161,065,033	0.95	0.02	< 0.00	(-9.38 ; 19.49)
OMP	-6,748,712	3,992,626	2,166,732	627,046,043	0.81	< 0.00	< 0.00	(-30.47 ; 26.52)

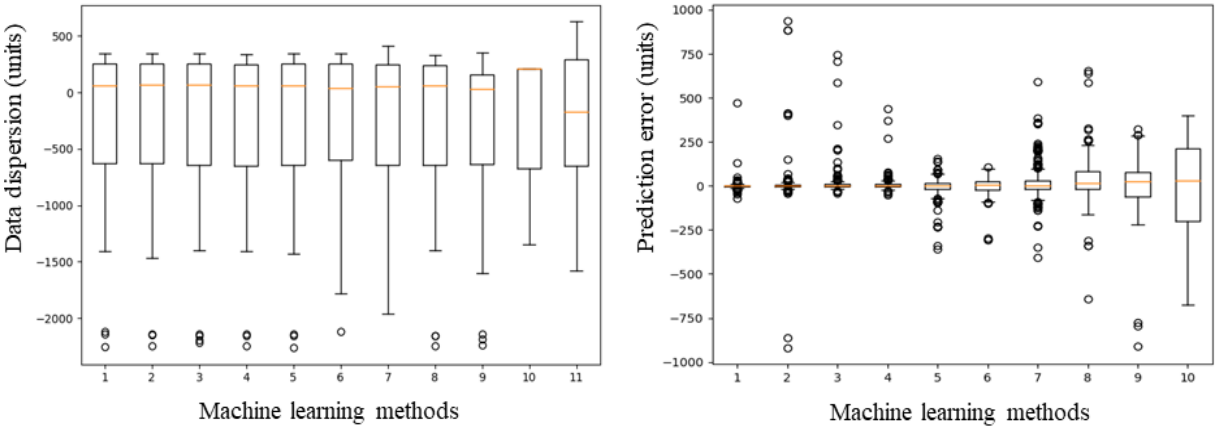

Figure 1: Boxplot for the data dispersion and confidence interval for the error.

Analysing Fig. 1 with the ML ordered according to Table III, the first boxplot set (on the left) compares the distribution of the prediction of the 10 MLs with the boxplot presented in number "1" for the real values that should be predicted, presenting a proper fitting for all of the methods. The second boxplot set shows that the better the ML prediction, the closer the error is to be a value of zero. This is a good sign that the selected ML for optimisation (DTR) has the desirable ability to make useful predictions with an expected error that is near to the real value, but the evaluation of the data distribution can only help as a tiebreaker.

For a comparison issue, after the test and selection of the ML, a random search was implemented considering the space solution defined in Eqs. (2) to (4) and tested over 268 hours to find a good result for reference and comparison on the solutions found in the optimisation process. A total of 242,600 solutions were evaluated, representing 0.6 % of all possible combinations, with the best value resulting in 10,778,862.

Fig. 2 presents the results for the application of the scenarios presented in item 3.3 respecting the value of the current best solution on the iteration and the time for the end of each iteration. The only two scenarios that were not represented in full were the serial and ML×1000 that demanded an exponential time to be evaluated (Table IV) and interfered on the scale of the representation for the other methods, which only presented the initial three values.

According to Fig. 2, the best solutions optimization scenarios generated are the parallel ones, or the ones that had machine learning with 100 and 1,000 iterations before the simulation evaluation. This examination can lead to an earlier conclusion without the analysis of the related time. For a final ranking, the methods were scored for the best values of

objective function found and the time spent with the same weight for both variables. The results are presented in Table IV.

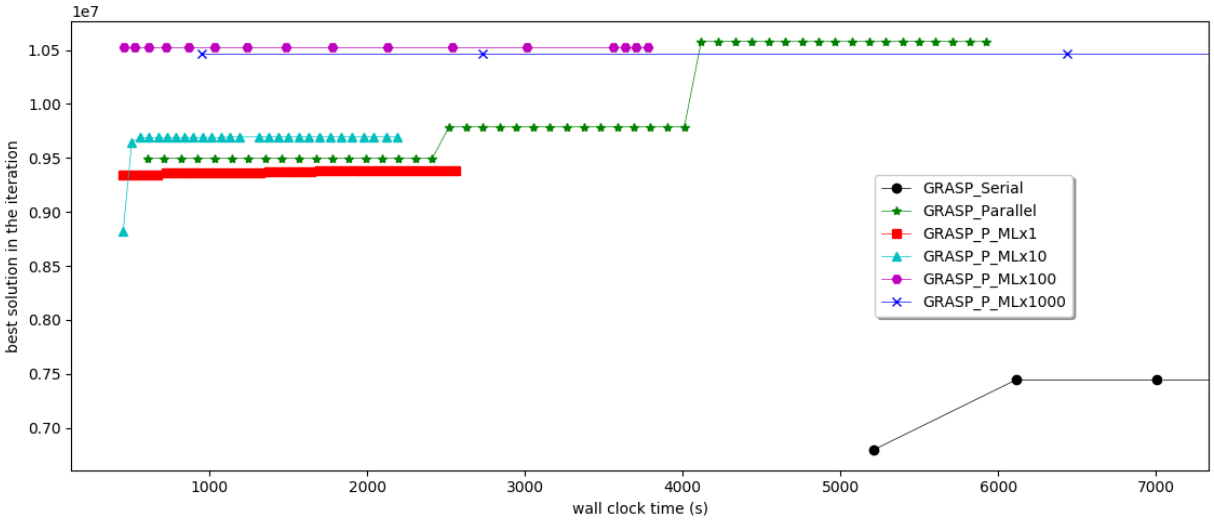

Figure 2: Results for GRASP and variances relating objective function values and time.

Table IV: Ranking of the optimisation combinations.

Scenario	**Final solution**			**Time** (s)		**Final score**
	Solution (*lot*; *sec*; *rev*)	**Objective function**	**Score**	**Value**	**Score**	
4: GRASP P ML ×10	1,927; 85; 4,800	9,695,122	0.899	2,193	1.000	0.899
3: GRASP P ML ×1	1,598; 82; 2,700	9,385,339	0.871	2,559	0.857	0.746
5: GRASP P ML ×100	1,568; 07; 3,000	10,528,161	0.977	3,778	0.580	0.567
2: GRASP Parallel	1,475; 69; 5,400	10,578,676	0.981	5,925	0.370	0.363
6: GRASP P ML ×1000	1,384; 12; 6,000	10,465,319	0.971	30,656	0.072	0.069
1: GRASP Serial	1,455; 57; 7,500	10,162,864	0.943	43,399	0.051	0.048
Reference: Random search	1,570; 12; 2,100	10,778,862	1.000	964,800	0.002	0.002

Table IV presents the final scores and ranking of the scenarios tested, defining that the use of the GRASP along with ten iterations and evaluation using the DTR before the call of the parallel simulation generates the best optimisation set evaluating only the OF and time. As a result, the use of 10 iterations to generate neighbours to be evaluated using the predictions of the ML DTR is 17.0 %, 37.0 % and 92.3 % better than the use of 1, 100 and 1,000 iterations, respectively. This effect can be explained by the breakeven generated by the trade-off between the time and OF value.

For that reason, a small number of predictions, iteration = 1, generated only neighbours for the current solutions that have OF values that are very similar to one another. This fact has a direct impact on the computational time because it generated a higher number of predictions with similar values that demanded a parallel evaluation at the end of the iteration. Oppositely, the use of a large number of iterations (1,000) before simulation evaluation demanded a higher amount of time (at least 2,700 seconds) at each iteration.

Evaluating the quality of the scenarios response in a management way, the scenario "GRASP P ML ×100" can configure a good solution because it lowers the size of the economic lot and the stock level that configures an increase in the stock turnover with the intensification of 37.5 % during the revision period. To find this solution, 42.0 % more time was necessary which represents only 26.4 minutes more than the previous best solution with a 7.9 % better OF.

Regarding the use of parallelism, the gain was 86.3 % and 86.8 % related to time and the final score of the serial programming (scenario 1 vs 2). In comparison, the 3 best solution scores were on average 50.7 % better than the parallel ones (scenarios 3-5 vs 2). Therefore, the use of both methods generated an average improvement of 93.5 %, on the final score, comparing a more traditional approach of only using the GRASP metaheuristic in a serial way for the proposed object study (scenarios 3-5 vs 1).

5. CONCLUSION

According to the results presented in Section 4, there is strong evidence corroborating with the main idea of the present study that the use of the presented novel optimization algorithm with machine learning method and parallelism within a metaheuristic to search in a large solution space compound an efficient and fast decision support tool for the economic lot-size problem. This is done by integrating good practices related to hardware and software optimisation and it can contribute to a significant improvement in the searching time. It can additionally find a good value of the intended objective function. The proposed optimisation algorithm can be tested on other industrial problems and for academic studies to verify the effectiveness in problems like logistics and healthcare.

The use of the GRASP metaheuristic with the DTR machine learning metamodeling to evaluate the fit function (objective function) of the responses was proved to reduce searching time and walk on the solution space with enough power to find a solution near the best local optimum on the study object. The use of parallel evaluations generated 86.3 % of improvement in time and the increment of a machine learning an additional average of 52.0 % for time reduction, with 94.0 % on average (6 scenarios) for the OF, comparing the best local optimum found with the referenced random search.

The proposed algorithm in Python performed most of all the desired issues for discrete event simulation optimisation on a single platform. These were namely: simulation modelling and data gathering, different machine learning training and testing, data storage, manipulation and statistical inference, parallelisation of the desired simulation scenarios and implementation of a specific metaheuristic with the connection of all the aforementioned issues. Such flexibility was only possible to be performed on an open source environment, without being seen yet on traditional proprietary simulation software with closed optimisation packages.

For future studies, the authors are willing to use a different set of machine learning methods on a distributed framework with direct data collection on a real study case, performing an online approach for real-time simulation and prediction throughout the ongoing process.

ACKNOWLEDGEMENTS

The authors are grateful to Brazil's institutions FAPEMIG (APQ-01188-16), CAPES and CNPq (305545/2017-5 and 431758/2016-6) for funding this work, and the anonymous referees for suggestions that contributed to substantial improvements in the paper.

REFERENCES

[1] Salam, M. A.; Khan, S. A. (2016). Simulation based decision support system for optimization: A case of Thai logistics service provider, *Industrial Management & Data Systems*, Vol. 116, No. 2, 236-254, doi:10.1108/IMDS-05-2015-0192

[2] Dorigatti, M.; Guarnaschelli, A.; Chiotti, O.; Salomone, H. E. (2016). A service-oriented framework for agent-based simulations of collaborative supply chains, *Computers in Industry*, Vol. 83, 92-107, doi:10.1016/j.compind.2016.09.005

[3] Lopes, H. dos S.; Lima, R. da S.; Leal, F.; Nelson, A. de C. (2017). Scenario analysis of Brazilian soybean exports via discrete event simulation applied to soybean transportation: The case of Mato Grosso state, *Research in Transportation Business & Management*, Vol. 25, 66-75, doi:10.1016/j.rtbm.2017.09.002

[4] Landa, P.; Sonnessa, M.; Tànfani, E.; Testi, A. (2016). Multiobjective bed management considering emergency and elective patient flows, *International Transactions in Operational Research*, Vol. 25, No. 1, 91-110, doi:10.1111/itor.12360

[5] Fu, M. C. (2015). *Handbook of Simulation Optimization*, Springer, New York, doi:10.1007/978-1-4939-1384-8

[6] Li, M.; Yang, F.; Uzsoy, R.; Xu, J. (2016). A metamodel-based Monte Carlo simulation approach for responsive production planning of manufacturing systems, *Journal of Manufacturing Systems*, Vol. 38, 114-133, doi:10.1016/j.jmsy.2015.11.004

[7] Yousefi, M.; Yousefi, M.; Ferreira, R. P. M.; Kim, J. H.; Fogliatto, F. S. (2018). Chaotic genetic algorithm and Adaboost ensemble metamodeling approach for optimum resource planning in emergency departments, *Artificial Intelligence in Medicine*, Vol. 84, 23-33, doi:10.1016/j.artmed.2017.10.002

[8] Saviniec, L.; Santos, M. O.; Costa, A. M. (2018). Parallel local search algorithms for high school timetabling problems, *European Journal of Operational Research*, Vol. 265, No. 1, 81-98, doi:10.1016/j.ejor.2017.07.029

[9] Aringhieri, R.; Landa, P.; Soriano, P.; Tànfani, E.; Testi, A. (2015). A two level metaheuristic for the operating room scheduling and assignment problem, *Computers & Operations Research*, Vol. 54, 21-34, doi:10.1016/j.cor.2014.08.014

[10] Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithm*, John Wiley & Sons, Hoboken

[11] Alba, E.; Luque, G.; Nesmachnow, S. (2012). Parallel metaheuristics: recent advances and new trends, *International Transactions in Operational Research*, Vol. 20, No. 1, 1-48, doi:10.1111/j.1475-3995.2012.00862.x

[12] Wang, S.; Wan, J.; Zhang, D.; Li, D.; Zhang, C. (2016). Towards smart factory for Industry 4.0: a self-organized multi-agent system with big data based feedback and coordination, *Computer Networks*, Vol. 101, 158-168, doi:10.1016/j.comnet.2015.12.017

[13] Zúñiga, E. R.; Moris, M. U.; Syberfeldt, A. (2017). Integrating simulation-based optimization, lean, and the concepts of Industry 4.0, *Proceedings of the 2017 Winter Simulation Conference*, 3828-3839, doi:10.1109/WSC.2017.8248094

[14] Calvet, L.; de Armas, J.; Masip, D.; Juan, A. A. (2017). Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs, *Open Mathematics*, Vol. 15, No. 1, 261-280, doi:10.1515/math-2017-0029

[15] Amouzgar, K.; Bandaru, S.; Ng, A. H. C. (2018). Radial basis functions with a priori bias as surrogate models: A comparative study, *Engineering Applications of Artificial Intelligence*, Vol. 71, 28-44, doi:10.1016/j.engappai.2018.02.006

[16] Bandaru, S.; Ng, A. H. C. (2015). On the scalability of meta-models in simulation-based optimization of production systems, *Proceedings of the 2015 Winter Simulation Conference*, 3644-3655, doi:10.1109/WSC.2015.7408523

[17] Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; Layton, R.; Vanderplas, J.; Joly, A.; Holt, B.; Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, Workshop: Languages for Data Mining and Machine Learning*, 108-122

[18] Müller, A. C.; Guido, S. (2017). *Introduction to Machine Learning with Python*, O'Reilly Media, Sebastopol

[19] Kubat, M. (2017). *An Introduction to Machine Learning*, 2nd edition, Springer, Cham, doi:10.1007/978-3-319-63913-0

[20] Negahban, A.; Smith, J. S. (2014). Simulation for manufacturing system design and operation: Literature review and analysis, *Journal of Manufacturing Systems*, Vol. 33, No. 2, 241-261, doi:10.1016/j.jmsy.2013.12.007

[21] Kleijnen, J. P. C. (2017). Regression and Kriging metamodels with their experimental designs in simulation: A review, *European Journal of Operational Research*, Vol. 256, No. 1, 1-16, doi:10.1016/j.ejor.2016.06.041
[22] Juan, A. A.; Faulin, J.; Grasman, S. E.; Rabe, M.; Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems, *Operations Research Perspectives*, Vol. 2, 62-72, doi:10.1016/j.orp.2015.03.001
[23] Xu, J.; Huang, E.; Chen, C.-H.; Lee, L. H. (2015). Simulation optimization: A review and exploration in the new era of cloud computing and big data, *Asia-Pacific Journal of Operational Research*, Vol. 32, No. 3, Paper 1550019, 34 pages, doi:10.1142/S0217595915500190
[24] Gruler, A.; Panadero, J.; de Armas, J.; Pérez, J. A. M.; Juan, A. A. (2018). A variable neighborhood search simheuristic for the multiperiod inventory routing problem with stochastic demands, *International Transactions in Operational Research*, 22 pages (in press), doi:10.1111/itor.12540
[25] Kirk, D. B.; Hwu, W.-M. W. (2016). *Programming Massively Parallel Processors: A Hands-on Approach*, 3rd edition, Elsevier, Amsterdam
[26] Choi, C.; Seo, K.-M.; Kim, T. G. (2014). DEXSim: an experimental environment for distributed execution of replicated simulators using a concept of single simulation multiple scenarios, *Simulation*, Vol. 90, No. 4, 355-376, doi:10.1177/0037549713520251
[27] Katsios, D.; Xanthopoulos, A. S.; Koulouriotis, D. E.; Kiatipis, A. (2018). A simulation optimisation tool and its production/inventory control application, *International Journal of Simulation Modelling*, Vol. 17, No. 2, 257-270, doi:10.2507/IJSIMM17(2)425
[28] Resende, M. G. C.; Ribeiro, C. C. (2016). *Optimization by GRASP*, Springer, New York, doi:10.1007/978-1-4939-6530-4
[29] Hillier, F. S.; Lieberman, G. J. (2015). *Introduction to Operational Research*, 10th edition, McGraw-Hill, New York
[30] Sifaleras, A.; Konstantaras, I. (2017). Variable neighborhood descent heuristic for solving reverse logistics multi-item dynamic lot-sizing problems, *Computers & Operations Research*, Vol. 78, 385-392, doi:10.1016/j.cor.2015.10.004
[31] Véjar, A.; Charpentier, P. (2012). Generation of an adaptive simulation driven by product trajectories, *Journal of Intelligent Manufacturing*, Vol. 23, No. 6, 2667-2679, doi:10.1007/s10845-011-0504-x
[32] Hadjsaid, N.; Tranchita, C.; Rozel, B.; Viziteu, M.; Caire, R. (2009). Modeling cyber and physical interdependencies − Application in ICT and power grids, *Proceedings of the 2009 IEEE/PES Power Systems Conference and Exposition*, 6 pages, doi:10.1109/PSCE.2009.4840183
[33] Raschka, S.; Julian, D.; Hearty, J. (2016). *Python: Deeper Insights into Machine Learning*, Packt Publishing Ltd., Birmingham