Machine learning techniques applied to improve parallel discrete event simulation optimization

Wilson Trigueiro de Sousa Junior[a], José Arnaldo Barra Montevechi[b] and Rafael de Carvalho Miranda[b]

[a]*Mechanical Engineering Department, Federal University of São João del Rei, São João del Rei, Brazil*

[b]*Institute of Production Engineering and Management, Federal University of Itajubá, Itajubá, Brazil*

E-mail: wilson.trigueiro@ufsj.edu.br [Sousa Junior]; montevechi@unifei.edu.br[Montevechi]; rafael.miranda@unifei.edu.br[Miranda]

Abstract

The use of discrete event simulation optimization methods is a tool commonly used as a decision making support systems, on industrial problems regarding the resource allocation to maximize some set of cost and revenue values. The present work proposed and tested an open source framework developed on Python, integrating different strategies for optimization including multicore parallelism, metaheuristic and machine learning applied to a resource allocation on a theoretical shop floor problem. The results showed optimization improvements on processing time from 88.5% to 95.2%, obtaining a solution 95,3% near the global optimum.

Keywords: discrete event, simulation, optimization, parallelism, machine learning, metaheuristic and industry.

1. Introduction

In the seek to maintain the company market competitiveness, the satisfaction of constant costumers demands lies in the improvement of goods and services quality regarding to costs and final price (Salam and Khan, 2016). There for, the management of production systems demands, in many cases, the use of analytic tools to help in the process of identifying opportunities to improve the overall quality in terms of production, logistics, related services and other issues (Dorigatti *et al.*, 2016; Shahi, Mehdipour and Amiri, 2016; Lopes *et al.*, 2017; Landa *et al.*, 2018).

With the advent of industry 4.0, a wide range of data is being generated and stored on data centers with cost savings related to capture, transmit and data storage. To this volume of data generate direct benefit on the industries, techniques are needed that aid in their analysis, in such way that patterns are identified and transformed into information. As an example of such techniques, is possible to identify the discrete event simulation as sufficiently robust methodology on data treatment, capture patterns and the evaluation of scenarios to aid in the decision making process (Wang *et al.*, 2015; Xu *et al.*, 2016; Zúñiga, Moris and Syberfeldt, 2017; Zúñiga, Syberfeldt and Moris, 2017). In the same idea, other techniques classified as Machine Learning, use from data mine expertise and statistical methods to make predictions using existing data frames (Jahangirian *et al.*, 2010; Calvet *et al.*, 2017).

To improve a production system based on collected data, resource allocation is a common optimization problem addressed on discrete event simulation projects (Li, Jia and Wang, 2012; Zschieschang *et al.*, 2014; Lucidi *et al.*, 2016). These studies search for the best configuration of a given set of variables to approximate the goal of minimize costs and/or maximize the profit on a given situation. How the set of possible solutions in general demands an exponential time to evaluate, these problems are considered NP-hard to solve. In this sense, special algorithms where developed to obtain good results in a reasonable amount of time (wall clock). For this purpose, is possible to name the classes of heuristics, metaheuristics and hyperheuristics that represent a great number of the actual optimization research field (Azimi and Charmchi, 2012; Maashi, Kendall and Özcan, 2015; Raska and Ulrych, 2015; Resende and Ribeiro, 2016; Li, Özcan and John, 2017).

The present work contributes with the optimization area proposing and testing an open source framework developed on Python 3.0 and applied on a theoretical instance, putting together the concepts of parallel computing, metaheuristics and machine learning, benefiting from all these concepts to generate a novel approach tool to help the decision making on industrial resource allocation problem.

The remainder of the paper is organized as follows: Section 2 literature review on the main concepts related to the techniques applied on the framework methodology, Section 3 research method, Section 4 findings and discussion, Section 5 conclusions and future directions, and Section 6 references.

2. Literature review

In this section, the main concepts involved on the development of the modeling, simulation and optimization framework are presented, mainly but not restricted to discrete event simulation, optimization, machine learning techniques and parallel computing.

2.1. Discrete event simulation

The simulation in this paper refers to a selection of technics and practices to imitate a specific behavior from a real or ideal system, using resources (time, knowledge, dedicate people) to answer questions made for feasible changes on the studied object, when real experiments are too costly or impossible to be performed for one or more possible solutions. This can be used in a variety of fields, industries and applications, that mainly consist of gathering initial information, develop concepts and models from the important identified variables, analysis of the data collected from experiments and the help of computer software's (Law and Kelton, 1991; Banks *et al.*, 2010; Kelton, Sadowski and Swets, 2010; Hillier and Lieberman, 2015).

The questions made are often related to the optimization of specific characteristics that represent "what if" scenarios to the proposed system. Simulation is a body of knowledge and methods that is recommended to be used when the studied system involve the relation among variables that the collected data can be considered stochastic with none or minimal correlation, in an acceptable level, and considered independent and identically distributed (IID) to other variables, for example time (Bianchi et al., 2009).

The choose of the simulation package relies on two fundamental categories: simulation language or application-oriented simulator. The first offer a more flexible environment to the agent modeler express complex behavior with the drawback that requires previous knowledge and expertise on computer programming. The second category, in general adopts a "drag-and-drop" functionality that makes easier to people with none programmers skill learn to model, and commonly off-the-shelf simulation packages have built-in animation resource for a better presentation. In the middle of these two extreme categories are factors related but not restricted to (Maria, 1997; Gupta, 2014):

- Model development;
- Simulation of experiments;
- Animation;
- Integration with other tools;
- Analysis of results;
- Optimization;
- Efficiency related to quality of the results and time to process the model;
- Software update and maintenance;
- The goal of simulation project and its members (i.e., simulation model developers and model users).

On the operational research science there is a variety of papers relating the discrete event simulation with optimization. To cite only literature review papers about the theme were found 17 on the Web of Science and Scopus databases. To reference only for the last 4 years is possible to cite the following 6 articles. Negahban and Smith did a literature review considering the aspects to simulation manufacturing systems design, operations and language/package development from 2002 to 2013 (Negahban and Smith, 2014). Xu *et al.* reviewed based in 6 categories: ranking and selection, black-box search, meta-model based, gradient-based methods, sample path, stochastic constraints and multi-objective, explaining each category, with 4 random examples in total (Xu *et al.*, 2015). In 2015, Bierlaire related some issues related to the use of simulation optimization in transportation (Bierlaire, 2015). Alrabghi studied the simulation optimization applied to the maintenance problem (Alrabghi and Tiwari, 2015).

On specific reviews related to simulation optimization, is possible to cite Juan that talked about metaheuristics applied in simulation and stochastic combinatorial problems, and the differences about these two fields (Juan *et al.*, 2015). Kleijnen reviewed low-order polynomial regression and Kriging metamodeling in simulation, with no specific application (Kleijnen, 2017).

The use of metaheuristics on a parallel perspective, is more discussed from the year 2005 with the advent of multicore personal computers (Alba, 2005; Alba, Luque and Nesmachnow, 2013). The combination of ML and metaheuristic (learnheuristics) have a good potential to be used on industrial problems but it is yet not fully used (Calvet *et al.*, 2017). On data mining, Djenouri used parallelism on a GPU environment for the development of the ML training (Djenouri *et al.*, 2018). To relate some work where is identified applications of ML on industrial problems, is possible to cite quality prediction (Rostami, Dantan and Homri, 2015), the estimation of project duration (Pospieszny, Czarnacka-Chrobot and Kobylinski, 2018) and predict on the reliability of components (Alsina *et al.*, 2017). On the cited ML works, the error on prediction is considered but not a matter that affect the result. Considering the use of parallelism, metaheuristic and machine learning all together, were not identified any published work yet.

3. Methodology

In this section are presented the steps took on the construction and test for the proposed framework with focus on the implementation and analysis (Pereira *et al.*, 2015), mainly about the selection for the related components namely: study case, optimization platform, machine learning technique, metaheuristic and results evaluation.

3.1. The case study

In the present work, an inductive approach was used to test and confirm the gains in the use of parallelism, machine learn and metaheuristic. As shown by research, the use of a single case design is possible to be used in such case (Piekkari and Welch, 2004; Salam and Khan, 2016). The single study case design was used with multiple optimization techniques scenarios to find the best combination according to the time necessary to obey the stop criteria.

The presented case is adapted from previous works related to shop floor resource allocation (Azadeh, Asadzadeh and Tadayoun, 2014; Azadeh, Motevali Haghighi and Asadzadeh, 2014). Parallel machines pull resources to be processed and generate goods with a stochastic rate and probability to break. The maintenance staff repair the broken machines with a constant rate. If the number of disabled machines is bigger than the staff, it waits until someone be available. When the machine broke, the production in process is discarded. All the values used on the simulation are showed on Table 1.

Table 1. Problem distributions

Activity	**Probability distribution (min)**
Processing time	Normal (mean = 10, std deviation = 2)
Mean time to failure	Exponential (lambda = 1/300)
Maintenance time	Constant = 30

The objective function of the problem, presented on Equation 1, is used to evaluate each simulations scenario and find the maximum total gross profit.

$$\begin{aligned} Max\ F(machines_i, staff_j, time_k) \\ = (machines_i \times \$100) - \left(staff_j \times \$20 \times time_k\right) - (time_k \times 5) \end{aligned} \tag{1}$$

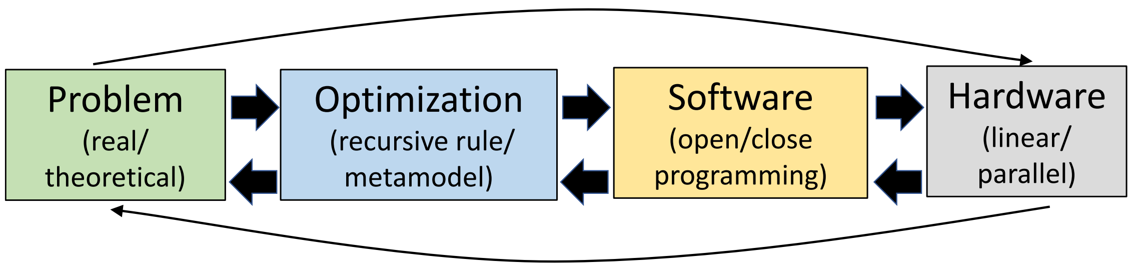
s.t.
$Machine_i = \{2, 3, \dots, 40\}: \forall i \in machines$
$Staff_j = \{2, 3, \dots, 20\}: \forall j \in staff$
$Time_k = \{30, 31, \dots, 150\}: \forall k \in time$

The variables that consist the answer of the optimization are formed by the number of machines and maintenance staff that is necessary to generate the maximum profit on a variable period of time measured on weeks. The solution space for this problem is the combination of the 82080 possible solutions. The chose for this specific problem was set because manufacture problems related to the resource allocation on the production shop-floor have a great impact on how the enterprise work regarding to investments, master planning, buffer allocation, dispatching rules, flexible production, etc. (Yang, Kuo and Cho, 2007; Geyik and Dosdoğru, 2012; Ponsignon and Mönch, 2014; Wang *et al.*, 2015).

3.2. Optimization platform

For the purpose of the present work, the characteristics presented on Figure 1 are considered in the selection of the framework components.

Figure 1. Different initial levels to consider on a DES optimization project.



Considering the problem definition and the objectives of the project, the following steps used on a simulation project are not used: real world data collection, distribution fit, conceptual model, validation, verification and animation. The only request is an environment able to run recursively the defined problem, retrieve data for analysis and retro feed the process to guide the optimization search thought solution space areas that are prone to be optimal or very near from it. To do that, were selected different strategies to search just part of the solution space (metaheuristic), run just the simulations that are more probable to be the best (forecast with machine learning) on parallel (with parallel computing), using an environment capable of handling all at the same software.

3.2.1. Selection of machine learning technique

The present paper is considered axiomatic because the main objective is to evaluate a set of optimization techniques that gives a good solution on a reasonable time (Will M. Bertrand and Fransoo, 2002). A search on the Web of Science and Scopus data bases did not found any significant result regarding to metaheuristic and machine learning applied to industrial engineering problems. As consequence, were not found any work as base for which are the most suitable machine learning for this case.

To evaluate the best machine learning techniques, a set of 33 methods were selected, representing the most used on literature (Buitinck *et al.*, 2013; Müller and Guido, 2017). Table 2 show the considered ML families and the tested methods.

Table 2. Machine learning methods tested

Family	Method	Parameters Used
Generalized Linear Models	Ordinary Least Squares (OLS)	-------------------------
	Ridge Regression (RR)	alpha = 0.5
	Ridge Regression with Cross Validation (RRCV)	alphas=[0.1, 1.0, 10.0]
	Lasso (L)	alpha = 0.1
	Elastic Net (EN)	alpha=0.1, l1_ratio=0.7
	Orthogonal Matching Pursuit (OMP)	n_nonzero_coefs=3
	Bayesian Ridge Regression (BRR)	-------------------------
	Automatic Relevance Determination Regression (ARDR)	compute_score=True

	Logistic Regression (LR)	C=100, penalty='l1', tol=0.01
	Stochastic Gradient Descent (SGD)	loss="hinge", penalty="l1",tol=0.01
	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
Kernel ridge regression	Kernel Ridge Regression (KRR)	alpha=1.0
Support Vector Machines	Support Vector Machines (SVM)	-------------------------
Nearest Neighbors	Nearest Neighbors Classification (NNC)	n_neighbors=15, weights='uniform'
	Nearest Centroid Classifier (NCC)	-------------------------
Gaussian Processes	Gaussian Process Regressor (GPR)	kernel = 1.0 * RBF(length_scale=1.0, length_scale_bounds=(1e-2, 1e3)) \ + WhiteKernel(noise_level=1e-5, noise_level_bounds=(1e-10, 1e+1)), alpha=0.0
Naive Bayes	Gaussian Naive Bayes (GNB)	-------------------------
	Multinomial Naive Bayes (MNB)	-------------------------
	Bernoulli Naive Bayes (BNB)	-------------------------
Decision Trees	Decision Trees Classifier (DTC)	max_depth=None, min_samples_split=2, random_state=0
	Decision Trees Regressor (DTR)	-------------------------
Ensemble methods	Bagging Classifier (BC)	KNeighborsClassifier(), max_samples=0.5, max_features=0.5
	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
	Ada Boost Classifier (ABC)	n_estimators=100
	Gradient Tree Boosting Regressor (GTBR)	n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0
	One-Vs-Rest Classifier (OVRC)	LinearSVC(random_state=0)
Multiclass and multilabel algorithms	One-Vs-One Classifier (OVOC)	LinearSVC(random_state=0)
	Error-Correcting Output-Codes (ECOC)	LinearSVC(random_state=0), code_size=2, random_state=0
Neural network models	Multi-layer Perceptron Classifier (MLPC)	solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15,), random_state=1
	Multi-layer Perceptron Regressor (MLPR)	solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(15,), random_state=1

All the methods presented on Table 2 that demand a set of parameters to be specified, were considered the default ones, presented on the Python package SciKit-learn, for more details on their meaning, please refer to the aforementioned program. The use of these default values is due to the primary objective of the present work to identify the most suitable methods to be used and give good insight about the direction to search on the solution space, not to adjust a specific method to the best parameters that will adjust it for a better prediction.

Other methods were identified but not tested on the present paper, because demand a different type of data presentation and manipulation that differ from the available on the studied problem, that if applied to the data frame, generate error or values with no significance to the proposition of the work, for example: Least Angle Regression, Cross Decomposition, Gradient Tree Boosting Classifier, LARS Lasso, Cross Decomposition, Multioutput Regression and Multioutput Classifier.

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

1) Set a data frame. The solution space of the problem is defined by 82080 possible solutions. Two vectors were built to help in the definition of the data frame. The first, store the sequential solution space and the second receive, in a random way, the solution from the first vector. With this procedure, the second vector is random and have no repeated values. To determine a representative sample, the 2.000 first solutions of the second vector (2,44% of the solution space) were select to represent the variability of the problem and can be generated on a reasonable time in the presented frame work.
2) Generate data to train the method. To apply a ML method, is necessary to have data specified in to 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 not evenly distributed, with originally generates many classes. To work around that issue, the results were classified as multiples of 300.000 as a reference that any solution that is inside this interval, have the same attractiveness for the decision making agent (Kubat, 2017).
3) Train and method classification. With the data frame composed by the 2.000 solutions, 80% (1600) were set to the training and 20% (400) were separated to the evaluation of the predictions against the real values. To compare the 33 methods, were used the following criteria: confidence interval ($\alpha = 5\%$) for the difference between the real value and the predicted value, k-fold with 3 levels to verify the presence of discrepant values, the minimum and maximum values generated for the error between the predicted and real, the Mean Absolute Error (MAE) and Mean Squared Error (MSE) and the wall clock time necessary for the training and prediction. To rank the best results were considered the values of MAE and MSE, as the fact that the most desirable method should have the minimum total error on prediction.

3.2.2. Selection of optimization technique

In scientific literature, many discrete event simulation optimization methods where used with success, for example heuristics, metaheuristics, metamodels (surrogate models), ranking and selection, complete enumeration and black-box (close commercial software) (Fu, 1994; Long-Fei and Le-Yuan, 2013; Riley, 2013; Negahban and Smith, 2014; Juan *et al.*, 2015; Xu *et al.*, 2015; Kleijnen, 2017; Gruler *et al.*, 2018). The selection of a specific method is related to the size of the solution space and previous knowledge how the variables interact with each other. In our case, to search the entire solution space is not made with a feasible time on a daily routine.

With that in mind, the metaheuristic optimization methodology was chosen because, according to the aforementioned authors, on its fundamental elements these optimization methods consider stop criteria that involve some restriction, for example the total wall clock time or interactions without improvement. Other element is the generation of a number of solutions by each interaction of the search algorithm, oriented by the method proposition. These two characteristics complies with the desirable characteristic to the present work to walk on the solution space in a parallel way.

Modern computers (from the year 2003) have the ability to process more than one instruction in parallel with the advent of multi-core processors on personal computers (Kirk and Hwu, 2010). However, not all the programs have the ability to take advantage of this method, that depends on the previews knowledge and the possibility to distribute a work with none or low dependency on previews data from the program that justify the use of parallelism (Choi, Seo and Kim, 2014). On the optimization way, is possible to use the parallelism to evaluate multiple scenarios on the same time (Saviniec, Santos and Costa, 2018). To this purpose, were chosen the Genetic Algorithm and the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristics that use population search on the next solution generation phase and walk on the solution space. These two optimization methods were used before on simulation, however was not found any previews work that couple parallelism, metaheuristic and machine learning (Jia *et al.*, 2015; Song, Qiu and Liu, 2015; Tiacci, 2015; Resende and Ribeiro, 2016).

The comparison of GA with ML is not a new task, for example the adjust of parameters on manufacture cell (Vosniakos, Tsifakis and Benardos, 2005) and the prediction using GA and Artificial Neural Networks (ANN), with GA been slower but with better confidence than ANN (Can and Heavey, 2012). For a collection of industrial engineering problem using GRASP is possible to cite oil distribution, inventory control and vehicle routing (Mujica Mota and Flores De La Mota, 2017) and the integration of GRASP and data mining (Ribeiro, Plastino and Martins, 2006). These two optimization methods (GA and GRASP) were select because both use population search to walk on the solution space, making this a starting point and less difficult to compare different methods.

3.3. Construction of the optimization environment

To accomplish the main objective of the present paper, two different approaches for discrete event optimization are related with software: metaheuristic and machine learning, and hardware with the use of parallelization on two different types of hardware. So, the proposed framework have these five characteristics: program a metaheuristic, program a machine learning technique, run a discrete event simulation environment, run on parallel the chosen simulation scenarios and couple on the same environment these four characteristics cited before.

Considering the software, the framework used open source packages found on the official repositories, constructed using the following:

- Programing environment on Python 3.6.5 x64, that have inbuild MapReduce function for parallel processing. This programing environment use the Pip 9.0.3 package management to download and install the necessary resources. Initially other programing environments were considered, but none have shown the five desired characteristics (Dagkakis and Heavey, 2016). An advantage of instead a compiled language (ex. Java and C++), Python is an interpreted language that make easier to a beginner programmer to develop a code and/or find mistakes (Raschka, Julian and Hearty, 2016). For the purpose of this work, the easier to reproduce the presented framework is preferable than a minor improvement that other languages can offer;
- For the discrete event simulation, was selected the SimPy 3.0.10 environment, that allows to modify the problem parameters from the code that is generated on the

optimization, already tested and used on engineering problems (Hadjsaid *et al.*, 2009; Véjar and Charpentier, 2012).

- To make possible the machine learning data frame, train and prediction, were used the packages NumPy 1.14.3, SciKit-Learn 0.19.1, Pandas 0.22.0 (with the dependencies python-dateutil-2.7.2, pytz-2018.4 and six-1.11.0) and SciPy 1.0.1. These packages are used on studies for machine learning using Python language and already have ML available for test (Raschka, Julian and Hearty, 2016; Müller and Guido, 2017); and
- The graphical representation of the results with MatPlotLib 2.2.2 (and the dependencies cycler-0.10.0, kiwisolver-1.0.1 and pyparsing-2.2.0).

On our approach to program the optimization, three algorithms were constructed where the first was used to build the initial populations and the primary data gathering for the ML data frame, analysis and training.

Algorithm 1 Generate initial population and initial data frame

```
     Inputs:
     $M_i = \{2, 3, \dots, 40\}$: Set of machines
     $S_j = \{2, 3, \dots, 20\}$: Set of maintenance staff
     $T_k = \{30, 31, \dots, 150\}$: Set of time to simulate
     % Phase 1: Generate the discrete event simulation
     $t \leftarrow 0$ % all simulations start at time 0
1    $\boldsymbol{resourceAlocSimul} \leftarrow (M_i, S_j, T_k)$
2        while $t \leq T_k$ do
3            $procesTime \leftarrow rand(Normal[10,2])$ for each $M_i$
4            $breakTime \leftarrow rand(Expo[1/300])$ for each $M_i$
5            $repairTime \leftarrow const(30)$ for each $M_i$ using a $S_j$
6            $\sum (M_i\ parts\ produced\ at\ time\ t)$
7            $t \leftarrow t + timeAdvanceNextEvent$
         end
     return OF(total of produced parts) %OF according to Equation 1
     % Phase 2: Construction of the initial population
8    for each $vecPop$ do
9        $vecPop[12{,}5\%] \leftarrow rand(H\_M_i) + rand(H\_S_j) + rand(H\_T_k)$
10       $vecPop[12{,}5\%] \leftarrow rand(L\_M_i) + rand(H\_S_j) + rand(H\_T_k)$
11       $vecPop[12{,}5\%] \leftarrow rand(H\_M_i) + rand(L\_S_j) + rand(H\_T_k)$
12       $vecPop[12{,}5\%] \leftarrow rand(H\_M_i) + rand(H\_S_j) + rand(L\_T_k)$
13       $vecPop[12{,}5\%] \leftarrow rand(L\_M_i) + rand(L\_S_j) + rand(H\_T_k)$
14       $vecPop[12{,}5\%] \leftarrow rand(L\_M_i) + rand(H\_S_j) + rand(L\_T_k)$
15       $vecPop[12{,}5\%] \leftarrow rand(H\_M_i) + rand(L\_S_j) + rand(L\_T_k)$
16       $vecPop[12{,}5\%] \leftarrow rand(L\_M_i) + rand(L\_S_j) + rand(L\_T_k)$
     end
     % Phase 3: Evaluation function for the initial population
17   for each $vecPop$ do
18       $resultPop \leftarrow resourceAlocSimul(vecPop)$
     end
     % Phase 4: Data transformation for initial machine learning training
19   for each $vecPop$ do
20       $dataML \leftarrow resultPop\ MOD300$ % Transform data into classes
21       $markML \leftarrow vecPop$
     end
     % Phase 5: Sort the population and set stop criteria
22   $vecPop \leftarrow sortMaxToMin(vecPop, resultPop)$
23   $iterMax \leftarrow 11$
```

```
24    $iter \leftarrow 0$
```

On Algorithm 1 the first phase is to build a discrete event simulation model (lines 1-7) that can be call recursively to construct the initial population and to evaluate the selected scenarios for the next generation. In special, the simulation model was developed to permit to run on sequence or parallel way. On the second phase (lines 8-16) an initial population was set to generate 120 random solutions respecting a combination of low and high levels for the variables of the problem. In this case, the variable “machines” have a range of values from 2 to 40, the low level represent values from 2 to 19 and high from 20 to 40, with the same analogy for the variables staff and time. This arrangement was set, instead of total random generation for the initial population, to give the ML method an initial data frame that have a good representation of the solution space with a more even sparse distance between the generated solutions. As consequence, have a better chance to generate predictions near the real values with low errors, for the training of the ML.

The third phase is the evaluation of the solutions (lines 17-18). This can be done by serial or parallel ways. The simulation call is the only part that is deliberately executed at this manner because is the function of the program that most demand computer power, as consequence time, to be processed and is a recursive function called each time the final value of a solution should be known. The fourth phase is the construction and training for the ML method (lines 19-21). To reduce the variance of the evaluation function returned by the simulation call, before it enters the data frame, it is applied a data transformation were the result of the simulation is substituted by the quotient of the division by 300. The chose for the value 300 is determined by the consideration that a difference less than 300 between results have the same value for decision-making point of view. This data transformation is necessary to the creation of classes and make it possible to use a more variety of ML techniques.

The last phase of the Algorithm 1 is to sort the vector of solution for the initial population according to the corresponding values stored on the results vector, decreasing the value of the results. This sorting is necessary because both optimization algorithms take advantage to their optimization process (line 22). The last two functions (lines 23-24) set the parameters for the stop criteria of the optimization process. At our programs was set that the optimization search stop the program when 11 consecutives iterations are generated without improvement on the best solution. With the defined stop criteria, the performance evaluation of the algorithms are related to the comparison of time and quality of the solution.

Algorithm 2 Genetic algorithm with and without machine learning

```
1     while $iter \leq iterMax$ do
2         for $numbOffspr$ do
3             % Phase 1: Selection and crossover
4             $a \leftarrow rand(vecPop)$
5             $b \leftarrow rand(vecPop)$
6             $crossPoint \leftarrow rand(3)$
7             if $crossPoint = 1$ do
8                 $offspr1 \leftarrow M_b + S_a + T_a$
9                 $offspr2 \leftarrow M_a + S_b + T_b$
              end
10            if $crossPoint = 2$ do
11                $offspr1 \leftarrow M_a + S_b + T_a$
12                $offspr2 \leftarrow M_b + S_a + T_b$
```

```
            end
13          else
14              $offspr1 \leftarrow M_a + S_a + T_b$
15              $offspr2 \leftarrow M_b + S_b + T_a$
            end
            % Phase 2 Mutation
16          $c \leftarrow rand(mutProb)$
17          if $mutProb \geq c$ do
18              $offspr1 \leftarrow mutatCrossov(rand(restrict))$
            end
19          $c \leftarrow rand(mutProb)$
20          if $mutProb \geq c$ do
21              $offspr2 \leftarrow mutatCrossov(rand(restrict))$
            end
        end
        % Phase 3.1: Evaluation and next generation without machine learning
22      for each $offspr$ do
23          $resultPop(rand) \leftarrow resourceAlocSimul(offspr)$
24          $vecPop(rand) \leftarrow offspr$
        end
        % Phase 3.2: Evaluation and next generation with machine learning
25      for each $offspr$ do
26          $tempResult \leftarrow machLearnPredict(offspr)$
27          $tempVec \leftarrow offspr$
        end
28      $tempResult \leftarrow sortMaxToMin(tempVec, tempResult)$
29      for each $tempResult(best)$ do
30          $resultPop(rand) \leftarrow resourceAlocSimul(tempResult)$
31          $vecPop(rand) \leftarrow tempVec$
        end
        % Phase 3.2.1: Data transformation to improve machine learning training
32      for each $tempResult(best)$ do
33          $dataML \leftarrow dataML + tempResult(best)\ MOD300$
34          $markML \leftarrow markML + tempVec$
        end
        % Phase 4: Find best solution and iteration count
35      $starResultBefore \leftarrow resultPop(first)$
36      $vecPop \leftarrow sortMaxToMin(vecPop, resultPop)$
37      $starResultAfter \leftarrow resultPop(first)$
38      if $starResultAfter \geq starResultBefore$ do
39          $iter \leftarrow 0$
        end
40      else
41          $iter \leftarrow iter + 1$
        end
    end
```

The phase one on Algorithm 2 (lines 4-15) represent the selection and crossover on the Genetic Algorithm. The selection itself is a random assortment of two individuals on the population vector. How the population is sorted (line22 on Algorithm 1), the random selected is made only on the first 80% of the population, that guarantee a variance on the fathers for the offsprings for a better walk on the solution space, to reduce the probability of fast convergence for a local optimum and enough power to scape when it happens. The crossover can happen on three ways, according to the three random point that are possible to be selected and exchange values for a new offspring. The mutation operation is a random selection with 8% probability for each offspring (lines 16-21). If an offspring is selected for mutation, one of the three parameters of the current solution is random selected and receive a random value respecting the range of the variables. In the final of the crossover iteration, 40 new solutions are generated.

The third phase is the evaluation of the fitting function for the generated offsprings. This evaluation can happen on two ways. The first is without the use of ML (lines 22-24), with a parallel or serial call of the simulation to evaluate the 40 new solutions. The second way is generating 120 and 1200 new solutions, instead of the original 40 and call the ML method to predict the value of the offsprings. After that, the offsprings are sorted and 40 of the best solutions are simulated to find their fitting (lines 25-28). How is known that the prediction using ML can generate an error (difference between real and predicted), two level for prediction on 120 and 1200 new offsprings on each generation are tested. The use of two levels is to verify if a prediction on a solution 3 or 30 times the original crossover than the original, generates a better solution respecting the fitting function and necessary time, still simulating 40 solutions on each generation.

The survival for the next generation (lines 29-31) is the random selection of individuals between the 10% and 80% related to the simulated evaluation and replace by the new offsprings. The next step when using a ML (lines 32-34) is the improve of the ML data frame and fitting, with the values evaluated and generated calling the simulation, for a better prediction on the next generation. The last phase on the Algorithm 2 (lines 35-41) is the sort of the population and count iterations with improvement or not, as the stop criteria of optimization.

Algorithm 3 GRASP with and without machine learning

```
1   while $iter \leq iterMax$ do
2       % Phase 1: Constructive
3       $a, b, c \leftarrow restrList(vecPop)$
4       $offspr1 \leftarrow buildVec(a, b, c)$
5       % Phase 2: Build partial solution
6       for each $numbOffspr$ do
7           $tempVec \leftarrow neighborhood(offspr1)$
        end
        % Phase 3.1: Evaluation of partial solution without machine learning
8       for each $tempVec$ do
9           $tempRes \leftarrow resourceAlocSimul(tempVec)$
        end
        % Phase 3.2: Evaluation and next generation with machine learning
10      for each $tempVec$ do
11          $tempResult \leftarrow machLearnPredict(tempVec)$
        end
        % Phase 3.2.1: Data transformation to improve machine learning training
12      for each $tempResult(best)$ do
13          $dataML \leftarrow dataML + tempResult(best)\ MOD300$
14          $markML \leftarrow markML + tempVec$
        end
        % Phase 4: Find best solution and iteration count
15      $starResultBefore \leftarrow tempResult(first)$
16      $vecPop \leftarrow sortMaxToMin(tempVec, tempResult)$
17      $starResultAfter \leftarrow tempResult(first)$
18      $bestVecList \leftarrow bestVecList + tempVec$
19      $bestResList \leftarrow bestResList + tempResult$
20      if $starResultAfter \geq starResultBefore$ do
21          $iter \leftarrow 0$
        end
22      else
23          $iter \leftarrow iter + 1$
        end
```

```
    end
```

For the GRASP optimization, the pre-processing phase is considered the Algorithm 1 itself, were the evaluation function is created and applied to the population. On Algorithm 3, the first phase, constructive, is to build the restrict candidate list for selection. On this purpose, was chosen a α=0,8 representing that the first 80% of the individual sorted on the initial population generated on the end of Algorithm 1, are candidates that compose the restrict list, and 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 solution generated on phase one, a neighborhood is generated evaluating 5 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, in this case the solution receives the value of the limit.

The third phase is the evaluation of the neighborhood, like the GA, with and without ML. When the evaluation is made without ML, it can be a serial or parallel call of the simulation. When it is done with ML, the optimization tested two levels with 125 and 1250 (10 neighborhoods at a time for 10 solutions) to predict and select the best 40 and call the simulation function for evaluation. This can be considered a 4% increase on the solution search on each interaction compared to the GA search, but is this case the probability of generating solutions on the range frontier compensate that effect. When the ML is used either on GA or GRASP, the evaluations are made using the parallel function. After the evaluation, the data frame is updated with the simulated values and the ML is adjusted for a better learning and prediction on the next generation. The fourth phase is the selection of the best solution and store on a list to compare with the current best solution, and finally count if any improvement is generated compared to the best solution and if the stop criteria is reached to end the optimization.

The hardware chosen for the experiments were two computers with different setups. The first is a Dell XPS 8910 with 16 GB of RAM and an Intel i7 7700 with 4 processors and 8 threads, running full load on 4,00GHz for all cores and 4,16GHz on single core single instruction processing. The second machine is a Supermicro X9DAi with dual Intel Xeon E5-2620 running full load on 2,29GHz for all cores and 2,48GHz on single core, with a total of 12 cores, 24 threads and 96 GB of RAM. This set of machines was put together to evaluate the impact of the computation power on the results of the optimization in terms of quantity of cores versus the velocity of processing. To guarantee that the same results are obtained independently from the software used, both systems are windows 10 x64 version1709 with all updates until may/01 2018 and the same random seed.

4. Results

The results presented are related to the experiment conducted for the selection of the ML technique for prediction and the application of the selected methods for the reduction of necessary iterations to find a good solution.

4.1. Result for testing the ML techniques and optimization scenarios

On Table 3 is presented the result for the application of the 33 ML methods applied to the problem present on item 3.1 and the corresponding evaluation parameter.

Table 3. Results for the 33 ML techniques

Method	CI (95%)	K-fold 3	Min	Max	MAE	MSE	R^2	T - Train	T - Predic.
GPR	(-0.19 ; 0.21)	0.99	-5.3	14.7	0.7	2.1	0.99	11.89	0.01
PR	(-0.29 ; 0.80)	0.99	-15.2	19.8	2.5	15.2	0.99	0.00	0.00
DTR	(-1.07 ; 0.29)	0.99	-21.0	25.0	3.1	25.3	0.99	0.00	0.00
RFC	(-0.11 ; 1.50)	0.15	-18.0	25.0	3.4	29.9	0.98	0.08	0.00
ETC	(-0.16 ; 1.79)	0.16	-26.0	42.0	4.3	49.8	0.97	0.08	0.00
DTC	(-0.81 ; 1.11)	0.16	-28.0	27.0	4.7	53.7	0.97	0.01	0.00
MLPR	(-0.47 ; 1.86)	0.97	-19.1	40.5	5.9	69.6	0.96	0.12	0.00
GNB	(-1.79 ; 2.23)	0.02	-40.0	42.0	11.0	206.9	0.90	0.01	0.01
LDA	(-2.20 ; 2.03)	0.05	-52.0	57.0	11.3	229.9	0.88	0.04	0.00
GTBR	(-2.88 ; 2.27)	0.84	-56.7	57.5	13.1	339.1	0.83	0.02	0.00
ARDR	(-2.56 ; 2.69)	0.84	-57.0	55.1	13.1	353.1	0.82	3.53	0.00
EN	(-2.57 ; 2.68)	0.84	-57.1	55.4	13.1	353.1	0.82	0.00	0.00
L	(-2.58 ; 2.68)	0.84	-57.1	55.4	13.1	353.3	0.82	0.00	0.00
BRR	(-2.58 ; 2.68)	0.84	-57.2	55.4	13.1	353.3	0.82	0.00	0.00
RRCV	(-2.58 ; 2.68)	0.84	-57.2	55.3	13.1	353.4	0.82	0.01	0.00
RR	(-2.58 ; 2.68)	0.84	-57.2	55.3	13.1	353.4	0.82	0.03	0.00
OLS	(-2.58 ; 2.68)	0.84	-57.2	55.3	13.1	353.4	0.82	0.03	0.00
OMP	(-2.58 ; 2.68)	0.84	-57.2	55.3	13.1	353.4	0.82	0.00	0.00
KRR	(-3.05 ; 2.23)	0.84	-58.5	56.5	13.1	358.1	0.82	0.17	0.00
NNC	(4.79 ; 10.14)	0.03	-82.0	57.0	15.2	422.8	0.79	0.00	0.01
LR	(-2.78 ; 3.49)	0.03	-49.0	57.0	17.3	483.0	0.76	0.65	0.00
RFR	(-3.19 ; 3.56)	0.70	-75.9	90.0	17.9	581.8	0.71	0.01	0.00
MNB	(1.68 ; 8.46)	0.03	-73.0	80.0	19.1	612.7	0.69	0.01	0.00
OVOC	(-0.37 ; 6.76)	0.05	-73.0	105.0	17.7	661.4	0.67	22.01	1.36
NCC	(-8.08 ; 1.64)	0.01	-141.0	122.0	23.9	1217.3	0.39	0.01	0.00
SGD	(-6.05 ; 6.55)	0.01	-118.0	113.0	28.6	1467.7	0.26	0.12	0.00
ABC	(-3.82 ; 6.91)	0.02	-118.0	135.0	28.5	1475.8	0.26	0.98	0.05
ECOC	(-6.58 ; 4.30)	0.01	-113.0	111.0	30.9	1518.3	0.23	31.37	0.01
MLPC	(-4.94 ; 7.27)	0.03	-123.0	135.0	33.9	1908.5	0.04	3.23	0.00
BNB	(-6.34 ; 6.10)	0.02	-137.0	135.0	34.7	1983.0	0.00	0.01	0.00
SVM	(-23.92 ; -11.40)	0.04	-158.0	114.0	35.9	2316.2	-0.17	0.57	0.15
BC	(6.47 ; 19.31)	0.03	-124.0	192.0	40.0	2909.7	-0.47	0.01	0.01
OVRC	(39.15 ; 53.49)	0.01	-98.0	186.0	58.9	4776.9	-1.41	2.26	0.00

According to Table 3, the results are presented respecting the decreasing order for preference on the quality of the predictions, mainly evaluated by the Mean Squared Error. The evaluation is explained on section 3.2.1. The upmost method for prediction on the evaluated problem is the GPR (Gaussian Process Regressor), with the lowest levels of MSE and small confidence interval (CI) for the error on the predictions that are the main evaluation points for ranking the ML methods. On the other parameters, the GPR got an improvement of respectively 65.3%, 25.8%, 72% and 86,2% for Min, Max, MAE and MSE.

The only parameters that GPR lose to the second ML method (PR) are the necessary times for training and prediction, by a significant margin. This fact can be a problem because a faster ML can generate worst solutions but can be preferable, considering that is only a support system that will pass to a second process (real simulation) for a final evaluation. For this reason, where selected two ML techniques (GPR e PR) for evaluation of the best combination of methods on the two selected optimization techniques (GA and GRASP).

Figure 2 show the boxplot set for the error and the prediction data on 9 first methods presented on Table 3.

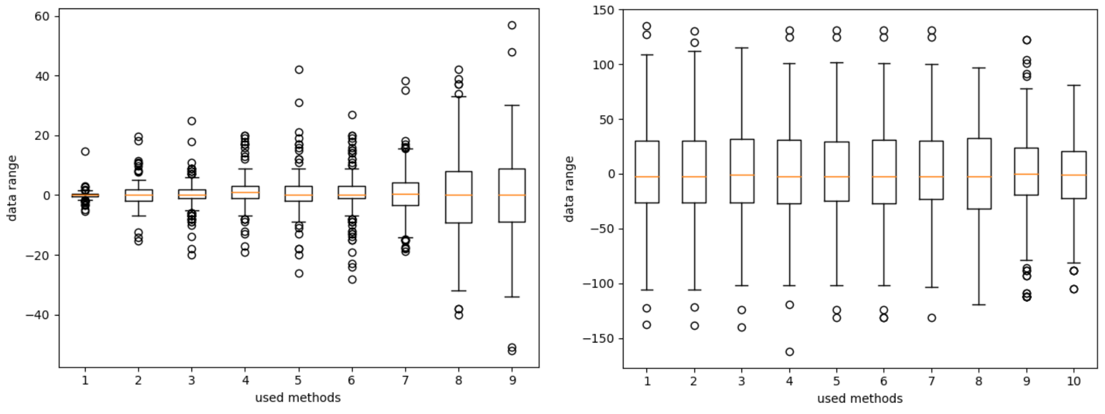


Figure 2. Boxplot for the error and data on prediction

Analyzing Figure 2, the first boxplot set shows that the better the ML prediction, the closest the error are from the value zero. That is a good sign that the selected ML for the optimization (GPR and PR) have the desirable ability to make good predictions with an expected error that is near to the true value. The second boxplot set on Figure 2 compare the distribution of the prediction of the 9 ML with the boxplot presented on number "1" for the real values that should be predicted.

According to the proposed for the optimization parameters and evaluation of the problem, were generated 12 scenarios according to Table 4.

Table 4 – Scenarios evaluated

Genetic Algorithm (GA)	Instance	GRASP	Instance
GA in serial	(GA-S)	GRASP in serial	(GRASP -S)
GA in parallel with population size of 120	(GA-120)	GA in parallel with population size of 120	(GRASP - 120)
GA in parallel with 1º ML and prediction size of 120	(GA-GPR-120)	GRASP in parallel with 1º ML and prediction size of 120	(GRASP - GPR-120)
GA in parallel with 1º ML and prediction size of 1200	(GA-GPR-1200)	GRASP in parallel with 1º ML and prediction size of 1200	(GRASP - GPR-1200)
GA in parallel with 2ª ML and prediction size of 120	(GA-PR-120)	GRASP in parallel with 2ª ML and prediction size of 120	(GRASP - GPR-120)
GA in parallel with 2ª ML and prediction size of 1200	(GA-PR-1200)	GRASP in parallel with 2ª ML and prediction size of 1200	(GRASP - GPR-1200)

4.2. Result for the application of the scenarios

A preliminary study before the application of the scenarios presented on Table 4 was conducted for the discovery of the best simulation solution. To find the global optimum, the space solution was generated and presented on Figure 3.

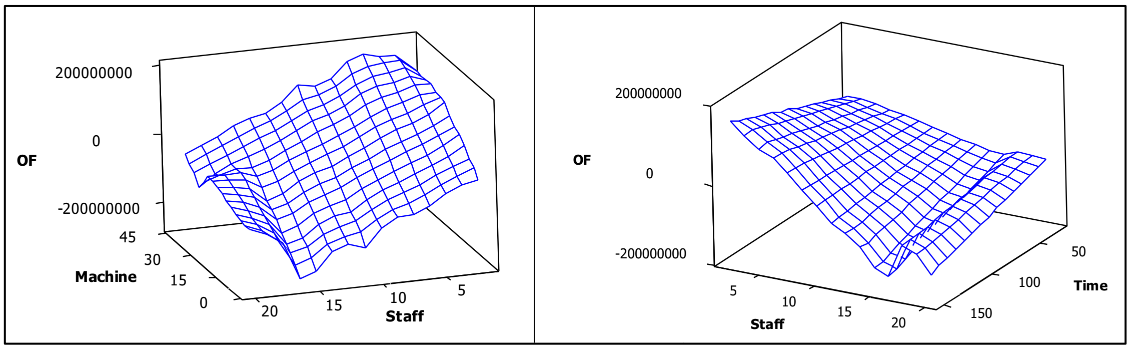


Figure 3. Solution space for the proposed problem

The space solution on Figure 3 was generated on a serial way, demanding a total of 12 days to find the global optimum with objective function of 399473800 and the solution $M_i = 40$, $S_j = 4$ and $T_k = 150$. This parameter serves as base to evaluate the solutions generated from the 12 optimization instances and know how far they are from the overall best. Analyzing the shape of the graphs on Figure 3, the presence of valleys, peaks and valleys, represent an opportunity to test if the proposed optimization instances have enough power to surpass and walk on the direction of the global optimum and in the best case find it. On Table 5 are presented the results for the application of the 12 instances on the two machines presented on section 3.3.

Table 5. Times necessary for each instance two different machines

Instance / Time (s)	Machine 1 – Xeon x2	Machine 2 – i7	Gain (%)
GRASP GPR 1200	2028	3061	33.75
GA GPR 1200	2708	2761	1.92
GRASP PR 1200	1725	1868	7.66
GA PR 1200	2982	3061	2.58
GRASP PR	2565	2677	4.18
GA GPR	3093	3143	1.59
GRASP GPR	2735	2708	-1.00
GA 120	4950	4806	-3.00
GA PR	5144	5451	5.63
GRASP 120	5336	5688	6.19
GA Serial	42410	15541	-172.89
GRASP Serial	17663	6604	-167.46
		Mean	-23.40

Analyzing the data from Table 5, it is possible to infer that in a first inference, the results shows that Machine 2 have a better overall performance. This result was a consequence for the higher capability to process single instructions with a higher clock speed for the serial model. Considering only the parallel processing instances, the result for the mean time is 5.95% less on Machine 1. For the purpose of the rest of this work, the results presented are related to the Machine 1 that have a better performance regarding the objective of the paper to use good optimization practices on the same environment.

Figure 5 present the results for the application of the instances related to the GA, respecting the value of the current best solution on the iteration and the time for the end of each iterations. The only instance that it is not represented is the serial alone that demanded a exponential time to be evaluated (Table 5) and interfere on the scale of the representation for the other methods.

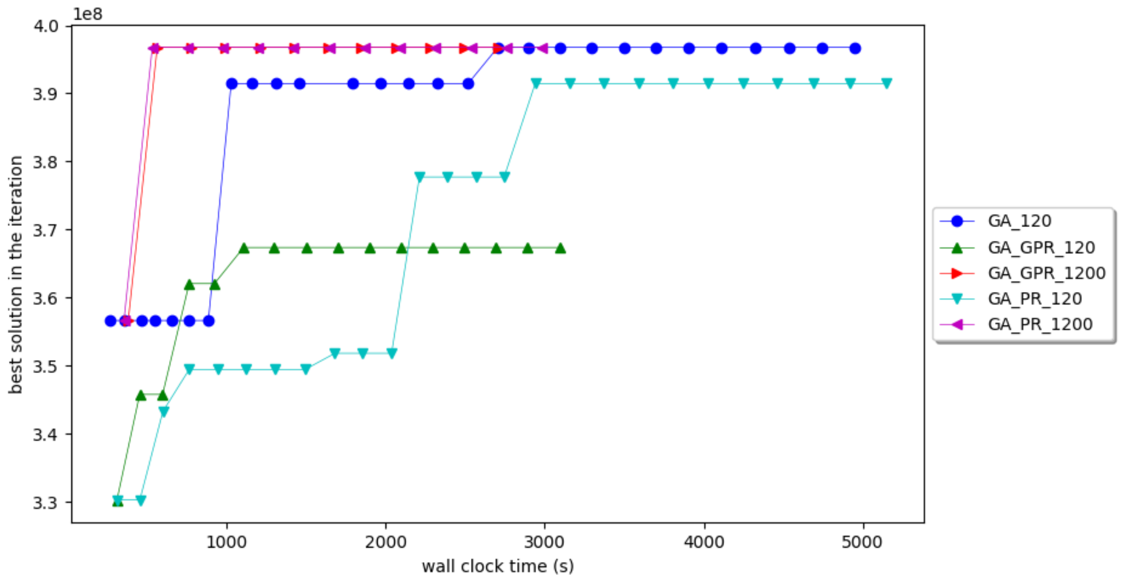


Figure 5. Results for GA and variances

According to Figure 5, the best optimization is the GA with GPR ML and the evaluation of 1200 individual by each iteration (GA_GPR_1200). This is presented by the red arrows, with a time reduction of 10,10% to the second best GA optimization method. Either methods that used a ML with the highest level of prediction (1200), found a solution 0.68% far from the global optimum on the second interaction of the algorithm. Figure 6 show the data for the instances that used the GRASP optimization and for the same reasons do not present the instance for serial optimization.

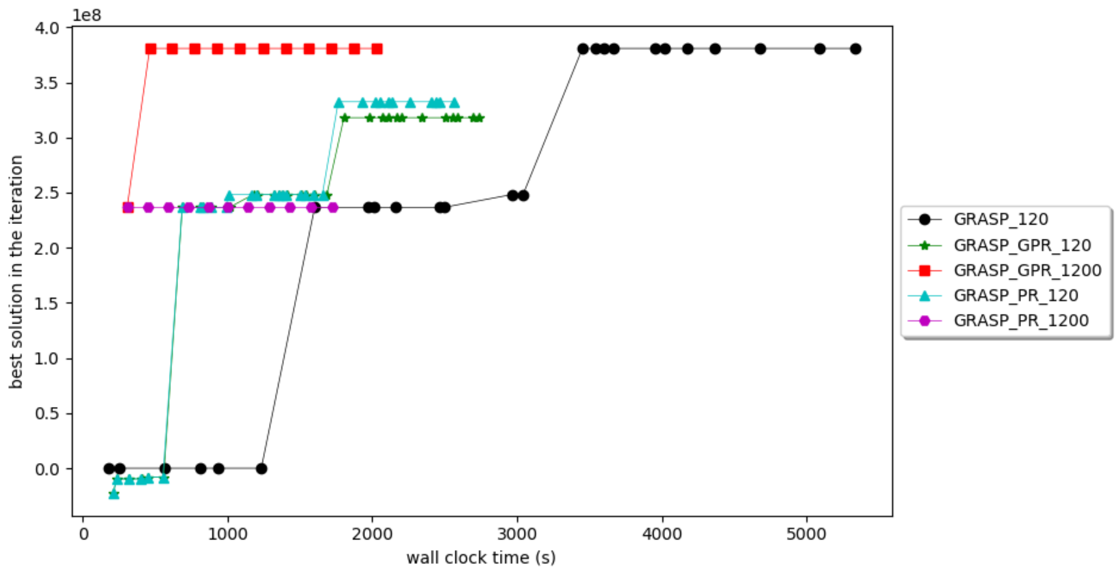


Figure 6. Results for GRASP and variances

The same that occurred with the GA, on GRASP the best optimization instances are related to the use of GPR/PR and 1200 predictions. This result couples with the same idea that the use of metaheuristic with a tuned ML have a chance to find good solutions in a faster way without the use of ML. Table 6 presents the overall results for the 12 optimization instances evaluated with a final ranking.

Table 6 - Ranking of the optimization combinations

Instance	Best Solution		Time (s)		Final
	Value	Score	Value	Score	Score
GRASP GPR 1200	380811200	0.960	2028	0.851	0.816
GA GPR 1200	396770000	1.000	2708	0.637	0.637
GRASP PR 1200	236614800	0.596	1725	1.000	0.596
GA PR 1200	396770000	1.000	2982	0.578	0.578
GRASP PR	332468900	0.838	2565	0.673	0.564
GA GPR	367363300	0.926	3093	0.558	0.516
GRASP GPR	317995100	0.801	2735	0.631	0.505
GA 120	396770000	1.000	4950	0.348	0.348
GA PR	391436700	0.987	5144	0.335	0.331
GRASP 120	380811200	0.960	5336	0.323	0.310
GA Serial	396770000	1.000	42410	0.041	0.041
GRASP Serial	94200900	0.237	17663	0.098	0.023

Considering that processing time and the value of the evaluation function have the same weight and assigning them 100% values, the final scores define that the instance GRASP GPR 1200 generated the best relation of time and generated solution, followed by the other three instances that used ML and 1200 predictions for each interaction. Compared with the global optimum, the best optimization generated a solution 4.68% less than the global one with a time near the minimum tested. In a direct comparison of the best instance that only used a metaheuristic on serial (GRASP serial), metaheuristic with parallelism (GA 120), the first level of prediction on ML (GRASP PR) and the second level of prediction (GRASP PR 1200) represent a “stair” of gains. From the base line of serial processing the gains are 72%, 85.5% and 88.5% for time and in comparison of the worst with the best scenario, a time improvement of 95.2%.

5. Conclusion

The main objective of this paper was attended with the proposition, construction and testing for a discrete event simulation optimization simulation for parameter tuning on an industrial engineering problem. This framework integrated the main concepts of parallelism, machine learning and metaheuristic, integrating issues related to advances on hardware and artificial intelligence. The discrete event simulation environment developed on Python was able to handle all these different elements on a single scope, without the dependency of closed third party programs, with all the packages been open source.

The 33 ML methods were used to identify the most suitable ones that can collaborate to a discrete event simulation optimization problem, in a way that the 2 used generated a significant interaction and result to the presented problem, serving as reference for future works. How the parameters of the ML were not an issue on the present work, other methods can be tuned to generate similar results than the proposed ones. For future works, the 33 ML methods used, and other ones not even considered from the beginning, can be tuned and generate different results than the presented.

What is strong is the idea that ML can perform a great integration with optimization methods. In this study the ML presented the benefit of been fast for processing with the drawback of the possible error on prediction. The metaheuristics were accretive on the result with an initial lack of direction to search only on good feasible regions on the solution space. The possible error on prediction was compensated and decreased with the parallel processing of the best possible predicted solutions of the interaction.

The machines used for testing represent two different sets available on the market that are able to take advantage of the parallel processing. On the point of view of optimization, the machine with more cores and less clock speed have a 5.95% better result. On the point of view for project investment, the saved amount of time on the optimization instances used do not justified alone an investment of $15,000.00 on machine 1 against the $2,000.00 on machine 2. For that, are recommended for future work the test of parallelism on a distributed environment with high-end workstations instead the use of a single new server.

This project can be compare to the efforts on combinatorial optimization for the construction of new hyper heuristics that can that advantage of ML techniques and parallelism. These efforts couple with one of the ideas of the industry 4.0 to the development of methods to deal with “big data”, identify hidden patterns and transform on useful information for decision making on the industrial environment.

6. References

Alba, E. (ed.) (2005) *PARALLEL METAHEURISTICS A New Class of Algorithm*. John Wiley & Sons, Inc.

Alba, E., Luque, G. and Nesmachnow, S. (2013) ‘Parallel metaheuristics: Recent advances and new trends’, *International Transactions in Operational Research*, 20(1), pp. 1–48. doi: 10.1111/j.1475-3995.2012.00862.x.

Alrabghi, A. and Tiwari, A. (2015) ‘State of the art in simulation-based optimisation for maintenance systems’, *Computers and Industrial Engineering*. Elsevier Ltd, 82, pp. 167–182. doi: 10.1016/j.cie.2014.12.022.

Alsina, E. F. *et al.* (2017) ‘On the use of machine learning methods to predict component reliability from data-driven industrial case studies’, *International Journal of Advanced Manufacturing Technology*, pp. 1–15. doi: 10.1007/s00170-017-1039-x.

Azadeh, A., Asadzadeh, S. M. and Tadayoun, S. (2014) ‘Optimization of operator allocation in a large multi product assembly shop through unique integration of simulation and genetic algorithm’, *International Journal of Advanced Manufacturing Technology*, 76(1–4), pp. 471–486. doi: 10.1007/s00170-014-6213-9.

Azadeh, A., Motevali Haghighi, S. and Asadzadeh, S. M. (2014) ‘A novel algorithm for layout optimization of injection process with random demands and sequence dependent setup times', *Journal of Manufacturing Systems*. The Society of Manufacturing Engineers, 33(2), pp. 287–302. doi: 10.1016/j.jmsy.2013.12.008.

Azimi, P. and Charmchi, H. R. (2012) 'A new optimization via simulation approach for dynamic facility layout problem with budget constraints', *Modelling and Simulation in Engineering*, 2012. doi: 10.1155/2012/189742.

Banks, J. *et al.* (2010) *Discrete-Event System Simulation*. 5th edn, *PrenticeHall international series in industrial and systems engineering*. 5th edn. Pearson.

Bianchi, L. *et al.* (2009) 'A survey on metaheuristics for stochastic combinatorial optimization', *Natural Computing*, 8(2), pp. 239–287. doi: 10.1007/s11047-008-9098-4.

Bierlaire, M. (2015) 'Simulation and optimization: A short review', *Transportation Research Part C: Emerging Technologies*. Elsevier Ltd, 55, pp. 4–13. doi: 10.1016/j.trc.2015.01.004.

Buitinck, L. *et al.* (2013) 'API design for machine learning software: experiences from the scikit-learn project', *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases*, pp. 1–15. Available at: http://arxiv.org/abs/1309.0238.

Calvet, L. *et al.* (2017) 'Learnheuristics: Hybridizing metaheuristics with machine learning for optimization with dynamic inputs', *Open Mathematics*, 15(1), pp. 261–280. doi: 10.1515/math-2017-0029.

Can, B. and Heavey, C. (2012) 'A comparison of genetic programming and artificial neural networks in metamodeling of discrete-event simulation models', *Computers and Operations Research*. Elsevier, 39(2), pp. 424–436. doi: 10.1016/j.cor.2011.05.004.

Choi, C., Seo, K.-M. and Kim, T. G. (2014) 'DEXSim: an experimental environment for distributed execution of replicated simulators using a concept of single simulation multiple scenarios', *Simulation*, 90(4), pp. 355–376. doi: 10.1177/0037549713520251.

Dagkakis, G. and Heavey, C. (2016) 'A review of open source discrete event simulation software for operations research', *Journal of Simulation*, 10(3), pp. 1–14. doi: 10.1057/jos.2015.9.

Djenouri, Y. *et al.* (2018) 'How to exploit high performance computing in population-based metaheuristics for solving association rule mining problem', *Distributed and Parallel Databases*. Springer US, 36(2), pp. 369–397. doi: 10.1007/s10619-018-7218-4.

Dorigatti, M. *et al.* (2016) 'A service-oriented framework for agent-based simulations of collaborative supply chains', *Computers in Industry*. Elsevier B.V., 83, pp. 92–107. doi: 10.1016/j.compind.2016.09.005.

Fu, M. (1994) 'A tutorial review of techniques for simulation optimization', *Simulation Conference Proceedings, 1994. Winter*, pp. 149–156. doi: 10.1109/WSC.1994.717096.

Geyik, F. and Dosdoğru, A. T. (2012) 'Process plan and part routing optimization in a dynamic flexible job shop scheduling environment: an optimization via simulation approach', *Neural Computing and Applications*, 23(6), pp. 1631–1641. doi: 10.1007/s00521-012-1119-7.

Gruler, A. *et al.* (2018) 'A variable neighborhood search simheuristic for the multiperiod inventory routing problem with stochastic demands', *International Transactions in Operational Research*, 00, pp. 1–22. doi: 10.1111/itor.12540.

Gupta, A. (2014) 'How to Select A Simulation Software', *International Journal of Engineering Research and Development*, 10(3), pp. 35–41. Available at: http://www.ijerd.com/paper/vol10-issue3/Version_5/F1033541.pdf.

Hadjsaid, N. *et al.* (2009) 'Modeling cyber and physical interdependencies - Application in ICT and power grids', *2009 IEEE/PES Power Systems Conference and Exposition, PSCE 2009*, pp. 1–6. doi: 10.1109/PSCE.2009.4840183.

Hillier, F. S. and Lieberman, G. J. (2015) *Introduction to Operational Research*. 10th edn. McGraw-Hill. Available at: http://highered.mheducation.com/sites/0073523453/information_center_view0/index.html.

Jahangirian, M. *et al.* (2010) 'Simulation in manufacturing and business: A review', *European Journal of Operational Research*. Elsevier B.V., 203(1), pp. 1–13. doi: 10.1016/j.ejor.2009.06.004.

Jia, S. *et al.* (2015) 'Improving performance of dispatch rules for daily scheduling of assembly and test operations', *Computers and Industrial Engineering*. Elsevier Ltd, 90, pp. 86–106. doi: 10.1016/j.cie.2015.08.016.

Juan, A. A. *et al.* (2015) 'A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems', *Operations Research Perspectives*. Elsevier Ltd, 2, pp. 62–72. doi: 10.1016/j.orp.2015.03.001.

Kelton, W. D., Sadowski, R. P. and Swets, N. B. (2010) *Simulation with Arena*. 5th edn. McGraw-Hill.

Kirk, D. and Hwu, W.-M. W. (2010) *Programming Massively Parallel Processors: A Hands-on Approach*. Second. Elsevier. doi: 10.1016/B978-0-12-381472-2.00001-5.

Kleijnen, J. P. C. (2017) 'Regression and Kriging metamodels with their experimental designs in simulation: A review', *European Journal of Operational Research*. Elsevier B.V., 256(1), pp. 1–16. doi: 10.1016/j.ejor.2016.06.041.

Kubat, M. (2017) *An Introduction to Machine Learning*. Second. Cham: Springer International Publishing. doi: 10.1007/978-3-319-63913-0.

Landa, P. *et al.* (2018) 'Multiobjective bed management considering emergency and elective patient flows', *International Transactions in Operational Research*, 25(1), pp. 91–110. doi: 10.1111/itor.12360.

Law, A. M. and Kelton, W. D. (1991) *Simulation Modeling and Analysis*. 2nd edn. New York: McGraw-Hill.

Li, S., Jia, Y. and Wang, J. (2012) 'A discrete-event simulation approach with multiple-comparison procedure for stochastic resource-constrained project scheduling', *International Journal of Advanced Manufacturing Technology*, 63(1–4), pp. 65–76. doi: 10.1007/s00170-011-3885-2.

Li, W., Özcan, E. and John, R. (2017) 'Multi-objective evolutionary algorithms and hyper-heuristics for wind farm layout optimisation', *Renewable Energy*, 105, pp. 473–482. doi: 10.1016/j.renene.2016.12.022.

Long-Fei, W. and Le-Yuan, S. (2013) ‘Simulation Optimization: A Review on Theory and Applications’, *Acta Automatica Sinica*. The Chinese Association of Automation and The Institute of Automation, Chinese Academy of Sciences, 39(11), pp. 1957–1968. doi: 10.1016/S1874-1029(13)60081-6.

Lopes, H. dos S. *et al.* (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 and Management*. Elsevier, 25(October), pp. 66–75. doi: 10.1007/978-3-319-63913-0.

Lucidi, S. *et al.* (2016) ‘A Simulation-Based Multiobjective Optimization Approach for Health Care Service Management’, *IEEE Transactions on Automation Science and Engineering*, 13(4), pp. 1480–1491. doi: 10.1109/TASE.2016.2574950.

Maashi, M., Kendall, G. and Özcan, E. (2015) ‘Choice function based hyper-heuristics for multi-objective optimization’, *Applied Soft Computing Journal*. Elsevier B.V., 28, pp. 312–326. doi: 10.1016/j.asoc.2014.12.012.

Maria, A. (1997) ‘Introduction to modelling and simulation’, *Winter Simulation Conference*, pp. 7–13. doi: 10.1109/WSC.1997.640371.

Mujica Mota, M. and Flores De La Mota, I. (eds) (2017) *Applied Simulation and Optimization 2*. Cham: Springer International Publishing. doi: 10.1007/978-3-319-55810-3.

Müller, A. C. and Guido, S. (2017) *Introduction to Machine Learning with Python*. O’Reilly. Available at: http://oreilly.com/catalog/errata.csp?isbn=9781449369415.

Negahban, A. and Smith, J. S. (2014) ‘Simulation for manufacturing system design and operation: Literature review and analysis’, *Journal of Manufacturing Systems*. The Society of Manufacturing Engineers, 33(2), pp. 241–261. doi: 10.1016/j.jmsy.2013.12.007.

Pereira, T. F. *et al.* (2015) ‘Integrating soft systems methodology to aid simulation conceptual modeling’, *International Transactions in Operational Research*, 22(2), pp. 265–285. doi: 10.1111/itor.12133.

Piekkari, R. and Welch, C. (2004) *Handbook of Qualitative Research Methods for International Business*, *Journal of International Business Studies*. doi: 10.4337/9781781954331.

Ponsignon, T. and Mönch, L. (2014) ‘Simulation-based performance assessment of master planning approaches in semiconductor manufacturing’, *Omega*. Elsevier, 46, pp. 21–35. doi: 10.1016/j.omega.2014.01.005.

Pospieszny, P., Czarnacka-Chrobot, B. and Kobylinski, A. (2018) ‘An effective approach for software project effort and duration estimation with machine learning algorithms’, *Journal of Systems and Software*, 137, pp. 184–196. doi: 10.1016/j.jss.2017.11.066.

Raschka, S., Julian, D. and Hearty, J. (2016) *Python: Deeper Insights into Machine Learning*. Packt Publishing Ltd.

Raska, P. and Ulrych, Z. (2015) ‘Comparison of modified Downhill Simplex and Differential Evolution with other selected optimization methods used for discrete event simulation models’, *Procedia Engineering*. Elsevier B.V., 100(January), pp. 807–815.

doi: 10.1016/j.proeng.2015.01.435.

Resende, M. G. C. and Ribeiro, C. C. (2016) *Optimization by GRASP*. New York, NY: Springer New York. doi: 10.1007/978-1-4939-6530-4.

Ribeiro, M. H., Plastino, A. and Martins, S. L. (2006) 'Hybridization of GRASP metaheuristic with data mining techniques', *Journal of Mathematical Modelling and Algorithms*, 5(1), pp. 23–41. doi: 10.1007/s10852-005-9030-1.

Riley, L. A. (2013) 'Discrete-event simulation optimization: A review of past approaches and propositions for future direction', *Simulation Series*, 45(11), pp. 386–393. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84880648611&partnerID=tZOtx3y1.

Rostami, H., Dantan, J.-Y. and Homri, L. (2015) 'Review of data mining applications for quality assessment in manufacturing industry: support vector machines', *International Journal of Metrology and Quality Engineering*, 6(4), p. 401. doi: 10.1051/ijmqe/2015023.

Salam, M. A. and Khan, S. A. (2016) 'Simulation based decision support system for optimization A case of Thai logistics service provider', *Industrial Management & Data Systems*, 116(2), pp. 236–254. doi: 10.1108/IMDS-05-2015-0192.

Saviniec, L., Santos, M. O. and Costa, A. M. (2018) 'Parallel local search algorithms for high school timetabling problems', *European Journal of Operational Research*. Elsevier B.V., 265(1), pp. 81–98. doi: 10.1016/j.ejor.2017.07.029.

Shahi, M. R. M., Mehdipour, E. F. and Amiri, M. (2016) 'Optimization using simulation and response surface methodology with an application on subway train scheduling', *International Transactions in Operational Research*, 23(4), pp. 797–811. doi: 10.1111/itor.12150.

Song, J., Qiu, Y. and Liu, Z. (2015) 'Integrating Optimal Simulation Budget Allocation and Genetic Algorithm to Find the Approximate Pareto Patient Flow Distribution', *IEEE Transactions on Automation Science and Engineering*, 13(1), pp. 149–159. doi: 10.1109/TASE.2015.2424975.

Tiacci, L. (2015) 'Coupling a genetic algorithm approach and a discrete event simulator to design mixed-model un-paced assembly lines with parallel workstations and stochastic task times', *International Journal of Production Economics*. Elsevier, 159, pp. 319–333. doi: 10.1016/j.ijpe.2014.05.005.

Véjar, A. and Charpentier, P. (2012) 'Generation of an adaptive simulation driven by product trajectories', *Journal of Intelligent Manufacturing*, 23(6), pp. 2667–2679. doi: 10.1007/s10845-011-0504-x.

Vosniakos, G.-C., Tsifakis, A. and Benardos, P. (2005) 'Neural network simulation metamodels and genetic algorithms in analysis and design of manufacturing cells', *The International Journal of Advanced Manufacturing Technology*, 29(5), pp. 541–550. doi: 10.1007/s00170-005-2535-y.

Wang, S. *et al.* (2015) 'Towards smart factory for Industry 4.0: A self-organized multi-agent system with big data based feedback and coordination', *Computer Networks*, 101, pp. 158–168. doi: 10.1016/j.comnet.2015.12.017.

Will M. Bertrand, J. and Fransoo, J. C. (2002) 'Operations management research methodologies using quantitative modeling', *International Journal of Operations & Production Management*, 22(2), pp. 241–264. doi: 10.1108/01443570210414338.

Xu, J. *et al.* (2015) 'Simulation Optimization: A Review and Exploration in the New Era of Cloud Computing and Big Data', *Asia-Pacific Journal of Operational Research*, 32(03), p. 1550019. doi: 10.1142/S0217595915500190.

Xu, J. *et al.* (2016) 'Simulation optimization in the era of Industrial 4.0 and the Industrial Internet', *Journal of Simulation*. Palgrave Macmillan UK, 10(4), pp. 310–320. doi: 10.1057/s41273-016-0037-6.

Yang, T., Kuo, Y. and Cho, C. (2007) 'A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem', *European Journal of Operational Research*, 176(3), pp. 1859–1873. doi: 10.1016/j.ejor.2005.10.048.

Zschieschang, E. *et al.* (2014) 'Resource efficiency-oriented optimization of material flow networks in chemical process engineering', *Procedia CIRP*. Elsevier B.V., 15, pp. 373–378. doi: 10.1016/j.procir.2014.06.066.

Zúñiga, E. R., Moris, M. U. and Syberfeldt, A. (2017) 'Integrating simulation-based optimization, lean, and the concepts of industry 4.0', in *2017 Winter Simulation Conference (WSC)*. IEEE, pp. 3828–3839. doi: 10.1109/WSC.2017.8248094.

Zúñiga, E. R., Syberfeldt, A. and Moris, M. U. (2017) 'The Internet of Things, Factory of Things and Industry 4.0 in manufacturing: Current and future implementations', *Advances in Transdisciplinary Engineering*, 6, pp. 221–226. doi: 10.3233/978-1-61499-792-4-221.