International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 4, August 2019, pp. 2481~2490
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i4.pp2481-2490

Adapted branch-and-bound algorithm using SVM with model selection

Kabbaj Mohamed Mustapha[1], El Afia Abdellatif[2]
ENSIAS, Mohamed V University, Rabat, Morocco

Article Info

Article history:

Received May 1, 2018
Revised Jan 5, 2019
Accepted Mar 4, 2019

Keywords:

Node selection strategy
Variable branching strategy
Branch and bound
SVM
Cross validation
Model selection

ABSTRACT

Branch-and-Bound algorithm is the basis for the majority of solving methods in mixed integer linear programming. It has been proving its efficiency in different fields. In fact, it creates little by little a tree of nodes by adopting two strategies. These strategies are variable selection strategy and node selection strategy. In our previous work, we experienced a methodology of learning branch-and-bound strategies using regression-based support vector machine twice. That methodology allowed firstly to exploit information from previous executions of Branch-and-Bound algorithm on other instances. Secondly, it created information channel between node selection strategy and variable branching strategy. And thirdly, it gave good results in term of running time comparing to standard Branch-and-Bound algorithm. In this work, we will focus on increasing SVM performance by using cross validation coupled with model selection.

Corresponding Author:

Kabbaj Mohamed Mustapha,
ENSIAS, University Mohamed V,
Abdellah Regragui Street, Madinat AL Irfane, Rabat, Morocco.
Email: mustapha.kabbaj@um5s.net.ma

1. INTRODUCTION

In real life, MILP has countless applications in different fields like logistics, finance and transportation. A very common solution technique of MILP framework is Branch-and-Bound. It continues to prove its relevance nowadays. Branch-and-Bound algorithm is an iterative algorithm, and at each iteration, we eventually get a feasible or optimal solution of an initial problem. Concretely, the algorithm constructs little by little a tree of nodes, where each node represents a modified version of the original problem. The construction of child nodes is conducted by a variable branching strategy. Another fundamental element in Branch-and-Bound algorithm is Node Selection Strategy that aims to choose among a nodes queue, one that will speed up the search.

Recently, some works has been trying to identify an analytic approach that decide about strategies described above, given a set of problem features. Authors use likely machine learning techniques. The main remark is that few authors who deal with node selection strategy, and if so, they did not use machine learning framework.

Our contribution is oriented towards learning efficient branch-and-bound strategies. This is the result of a consistent methodology beginning with the collection of the data set, and ending with the test of the final hypothesis. More explicitly, we:
- Define features
- Collect data set
- Pick the optimal learning model
- Learn the final hypothesis with the chosen model
- Implement and test the final hypothesis

Next, we address research papers relative to our work. To do so, we divide contributions in five sub-sections relatively to the strategy and to the learning technique used. Firstly, relatively to variable branching strategy, authors in [1], learned a function to be with approximatively the same performance as strong branching in term of precision, and in the same time, results in a gain of the processing time.

In this same category, we cite [2], wherein authors infer consistent data from applying an algorithm for detecting clauses. A clause being a combination of binary values affected to a set of indexed variables, that if it happens, the whole problem would be infeasible. In addition, that algorithm generates minimal clauses and restarts with active clauses ("those that can be used in fathoming child nodes"). In parallel, this information is used to choose branching variable with the best effect. Also [3] and [4] use backtracking to improve branching decisions.

Besides, there are classic variable branching rules, like strong branching, pseudo costs branching, hybrid branching, reliable branching, inference-based branching [5]. Note that reliability branching is known to be the best branching rule with the reliability $\eta_{rel} = 4 \, or \, 8$ [6]. For our experiments, we used: $\eta_{rel} = 8$. The reliability parameter η_{rel} is fixed to stop the calculation of pseudocost values after attaining a certain level of the branch and bound tree. This is because pseudocost value remain approximatively constant after calculating it several times for a determined variable.

Secondly, we cite from node selection strategy literature, in addition to classic node selection strategies, such as depth-first rule, breadth-first rule, and node best-estimate [5], authors of [1], extracted information from MILP Benchmark libraries by using specific algorithm called oracle. Thirdly, concerning learning algorithms, they are used in different engineering fields. Algorithms purposes are classification, regression, clustering [7][8]. There are algorithms that tend to do well in practice more than others [9][10]. In the same context of applying learning in branch and bound, the ExtraTrees is applied in [11].

Fourthly, when looking at model selection and the performance of algorithms, there are techniques used to tune parameters such as Fuzzy Logic controller for Ant Colony System (ACS) epsilon parameter [12]. Also, [13] and [14] used Hidden Markov Model (HMM) algorithm to tune the Particle Swarm optimization population size and acceleration factors parameters. Besides, authors in [15] used HMM to tune the inertia weight parameter of the Particle Swarm Optimization algorithm. Moreover, [16] used Fuzzy controller to control Simulated Annealing cooling law, [17] and [18] used HMM to tune ACS evaporation parameter and local pheromone decay parameter respectively, [19] and [20] used HMM to adapt the simulated annealing cooling law. Furthermore, [14] used SVM algorithm to predict the performance of optimization problems. Finally, authors in [20] used the Expectation-Maximization algorithm to learn the HMM algorithm parameters. Finally, this paper is the continuity of our previous papers which deals with the learning of branch-and-bound algorithm strategies, namely variable branching strategy and node selection strategy [21], [22]. The learning algorithm used was Support Vector Machine (*SVM*).

The rest of this paper is organized as follows: Section 2 recalls some basics on branch-and-bound algorithm and SVM algorithm with parameter tuning. In section 3, we present our methodology of inferring efficient branch and bound strategies and experimentation configuration. Sections 4 is dedicated to results. Finally, we conclude and propose some future work.

2. BRANCH-AND-BOUND AND SVM

In this section, we are first going to see an overview of a formal description of branch-and-bound strategies and present the features used in the algorithm. Secondly, we will investigate SVM most important advantages with a remainder of learning theory.

2.1. Branch-and-bound algorithm

Branch-and-bound algorithm is outlined in this section. We first define useful notation and then proceed with the explanation of the algorithm steps. Let us define a general MILP problem P as follows:

$$z = \min\{c^T x \mid Ax = b, x \geq 0, x \text{ Non-negative vector of dimension } n \text{ containing at least one integer}\}$$

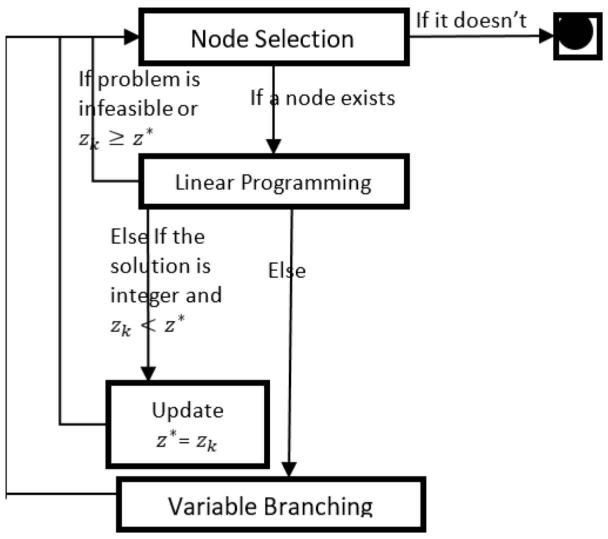
where $c \in R^n, b \in R^m and \, A$ is $m * n$ dimension matrix. We will use also define: P_{rel} is a relaxed version of P: which is

$$z_{rel} = min\{c^T x \mid Ax = b, x \geq 0\}$$

P_k is the problem in the k^{th} iteration which corresponds to a node in branch-and-bound tree.
$P_{k,rel}$ is a relaxed version of P_k.
z_k is the objective value of k^{th} node.
(x^*) is the incumbent point at iteration k, which means the vector that leads to the best z_k so far.
z^* is the objective function value on (x^*)

Briefly, the Branch-and-bound algorithm, in the case of minimization, is described as follows as shown in Algorithm 1. It is an iterative algorithm, and in each iteration k, we have at least three steps which are:

Firstly, the node selection step aims to retrieve a node from a node list that maximizes some criterion. This latter is specific to the node selection strategy. Secondly, and once we have picked a node P_k, we solve its relaxation $P_{k,rel}$ by an algorithm from the linear programming framework such as simplex or interior points. Depending on the results, we distinguish three cases. The first one is when the problem $P_{k,rel}$ is infeasible or the resulting objective function z_k value is greater than z^*. Consequently, the current iteration is termintated. The second case is when the solution is integer and $z_k < z^*$. In this moment, we update the incumbent point and its objective value z^*, then we move to the next iteration. In the third case, when none of the condifions mentioned before happens, we perform variable branching. In this final step, we must select a variable from a set of non-integer variables relatively to some defined criterion. And this criterion is defined by the variable branching strategy.



Algorithm 1. Branch-and-bound Algorithm

2.2. Support Vector Machine

SVM is in top ten machine learning algorithms [9], it is used for both classification and regression. It aims to find the hyperplane with the best margin. The best is demonstrated to be the large one differentiating between the hyperplane and nearest data points called support vectors.

2.2.1. Case of Linear Hypothesis set for SVM:

In the case of regression, and especially one variant of SVM called ε-SVM, we will present nextly, the case of linear hypothesis set. Let's have in have in the input, N training data, namely $(x_n, y_n), 0 \leq n \leq N$ The output of the algorithm is a linear function: $f(x) = w^T x + b$, with w a coefficient vector, x the unknown vector and b a constant.

The distance between a hyperplane of equation $w^T x + b = 0$ and the support vectors, is $\frac{1}{||w||}$. Consequently, maximizing the margin is equivalent to the next optimization problem:

$$P(w): \begin{cases} \min \frac{1}{2} w^T w \\ s.t.\ |y_n - (w^T x + b)| \leq \varepsilon, \forall n \end{cases}$$

With, ε being the error tolerance between y_n and $f(x)$.

The last problem might be infeasible. And to add more chance to be feasible, we add slack variables to the problem, in the following way:

$$P(w): \begin{cases} \min P(w) = \frac{1}{2} w^T w \ + C \sum_{i=1}^{N} (\xi_n + \xi_n^*) \\ s.t. \ y_n - (w^T x + b) \leq \ \varepsilon + \xi_n, \forall n \\ (w^T x + b) - y_n \leq \ \varepsilon + \xi_n^*, \forall \mathrm{n} \\ \xi_n, \xi_n^* \geq 0, \forall n \end{cases}$$

with, $\xi_n and \ \xi_n^*$ are the slack variables, and C is the cost parameter used to penalize data points outside the margin ε.

By using lagragian function, and quadratic optimization or other resolution methods, one can prove that the solution is with the form of:

$$f(x) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) x_i^T x + b$$

with $0 \leq \alpha_i, \alpha_i^* \leq C, \forall 1 \leq i \leq N.$

2.2.2. Case of non linear hypothesis set

In the situation, where we cannot find a hyperplane containing all training instances, one might transform the space of the training data to another, in such way can be comprised in one hyperplane on the new space. To do this transformation, one can use the well-known kernel methods. In fact, there are in literature different kernels used for SVM, such as RBF and polynomial. For the rest, we will present the distance calculation method for the RBF kernel.

Instead of using the standard $\mathrm{L}_2 - norm \ ||.||$, we used the norm associated with RBF kernel that is described as follows:

$$RBF(x_i, x_j) = \exp(-\gamma \left|\left|\mathrm{x}_j - \mathrm{x}_i\right|\right|^2)$$

with γ, is the gamma parameter. Its geometrical interpretation is, when the gamma parameter has larger values, the hyperplane associated with the solution will have more inclinations to contains, as far as possible, all training data. The form of the resulting target function, will be as follows:

$$f(x) = \sum_{i=1}^{N} (\alpha_i^* - \alpha_i) RBF(x_i, x) + b$$

In this paper, we will use ε-SVM regression algorithm with the RBF kernel twice for learning node selection strategy and variable branching strategy respectively.

2.3. Learning of variable branching strategy and node selection strategy

Concerning the variable branching strategy, we aim in this paper to imitate the behavior of the reliability branching rule. This rule is based on strong branching, which is time consuming. By and large, reliability branching uses an unreliability quality for variable pseudo-costs values. For this reason, reliability depends on numerous problem features. These features are to be classified in node-based features and variable-based features.

2.3.1. Node-based features

We use in this category features below**:**

- Reduced Objective values gain:

$\Delta_{k,red} = \frac{|z_k - z_{k-1}|}{|z_{k-1}|}$ (Note that the features should be independent of the problem scale)

- Depth in branch and bound tree d_k starting from zero.
- Node estimate.
- LP Objective Value

2.3.2. Variable-based features

In the same thinking line, we use:

- Pseudo-cost value
- The positive reduced cost and the negative reduced costs i.e.

$$\max\left(\frac{|c_{j,k}|}{\sum_{j:\, c_{j,k} \leq 0} |c_{j,k}|}, 0\right) \text{ and } \max\left(\frac{|c_{j,k}|}{\sum_{j:\, c_{j,k} \geq 0} |c_{j,k}|}, 0\right)$$

with $c_{j,k}$ is the j^{th} component value of cost vector of iteration k. These features aim to present either in minimization or maximization problems how we approach to the optimal solution.

The other specificity in our work beyond changes in features based on those presented in [11], is we add the value of learned function representing node selection in the set of features. This last point is justified in the following sub-section. For learning node selection strategy, we will imitate node estimate strategy. This strategy is the default one used in SCIP solver.

2.4. Interaction of node selection strategy and variable selection strategy

Intuitively, the choice of a node, by a node selection strategy, influences the choice the next branching variable. For this reason, we describe formally the variable branching strategy function *VB* in function of a combination of *NS* (Node selection strategy function) and other features described below:

$$VR(features\ set) = a * NS(\, d_k, negative\ reduced\ cost, positive\ reduced\ cost) + \sum_i a_i * feature\ i$$

where a and a_i are real numbers. Note that we double use *NS* features add more precision.

2.5. Overfitting and parameter tuning

In this sub-section, we will define overfitting, which is a very common problem in learning techniques that affects the final performance.

2.5.1. Overfitting

A learning model is, by definition, a couple of a learning algorithm and a hypothesis set. A learning algorithm is an iterative algorithm that searches the best hypothesis fitting the training data. This hypothesis is included in the hypothesis set chosen initially**.** A very common problem encountered in learning is overfitting. This phenomenon occurs when the learned hypothesis does not generalize well to all possible values beyond the training data. Causes are number of data points, noise and target complexity [7].

The choice of learning algorithms could affect the noise by affecting either bias or variance. In the case of SVM, the thorough choice of SVM parameters is required to prevent from overfitting. The RBF Kernel SVR algorithm used in this work has two parameters, cost and gamma. Cost defines how much is penalized misclassified examples and gamma defines how far the influence of a single training example reaches. As known small cost and large gamma, give higher bias and lower variance. In addition, large cost and small gamma give lower bias and larger variance. Consequently, we should tune cost and gamma parameters until we find tradeoff values to minimize the generalization error. One way to tune gamma and cost parameters is to use cross validation.

2.5.2. Cross validation with model selection

Before defining cross validation, let us find out what is validation. To do so, we define some useful notation:

D the data set
D_{train} the training set
D_{val} the validation set

The goal of validation is to give an estimation of the generalization error. First, it divides D of N data points, to D_{train} of size $N - K$ and D_{val} of size K, then learns the target function based on D_{train}. Finally, we calculate error of the target function in D_{val}. This latter error is proven an estimation of the generalization error. The Figure 1 represents what is described above.

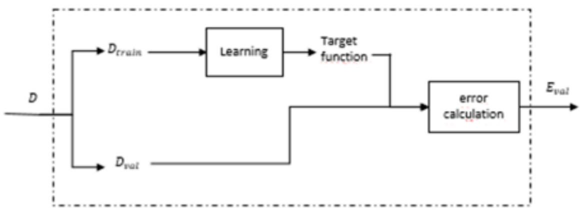


Figure 1. Validation method

The E_{val} error is a good estimate of generalization error but it is not too precise. To improve the precision, other techniques repose on validation like cross validation. Without the loss of generality, we present next 10-fold cross validation process.

Let's partition D to D_1, D_2 to D_{10}. We use validation process ten times for $D_{val,i} = D_i$ where $1 \leq i \leq 10$ and $D_{train,i} = D \backslash D_i$. In the output, we have 10 errors E_{val1} to E_{val10}. Then we calculate cross validation error denoted by E_{CV} which is the mean validation errors. The cross validation error is more precise that validation error. We resume this process in the Figure 2.

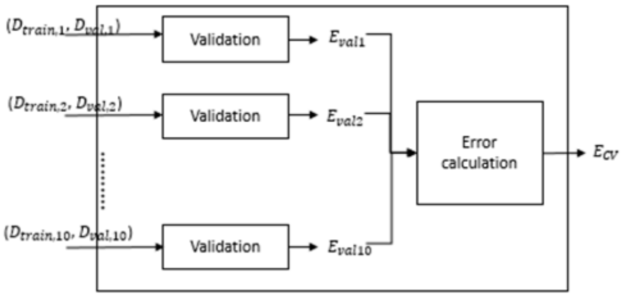
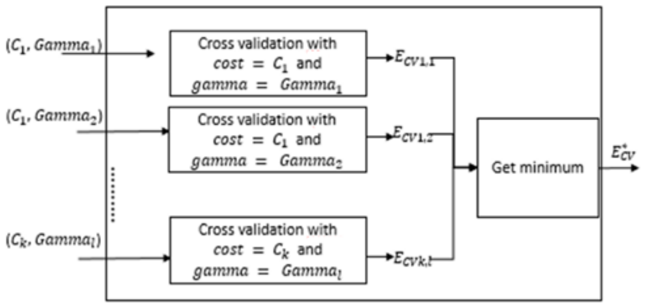


Figure 2. Cross validation for a specific learning algorithm

Now that we have presented cross validation, let us look forward model selection, that used in this paper to tune parameters of gamma and cost. For $k * l$ different combinations of cost and gamma, let's note a couple $(C_i, Gamma_j)$ with $1 \leq i \leq k$ and $1 \leq j \leq l$. As mentioned in the Figure 3, cross validation is executed multiple times with different parameter configuration. As result, we get errors $E_{CV1,1}$ to $E_{CVk,l}$. In the end, we have the configuration that have the lower error.



$(C_1, Gamma_1)$
$(C_1, Gamma_2)$
$(C_k, Gamma_l)$
Cross validation with $cost = C_1$ and $gamma = Gamma_1$
Cross validation with $cost = C_1$ and $gamma = Gamma_2$
Cross validation with $cost = C_k$ and $gamma = Gamma_l$
$E_{CV1,1}$
$E_{CV1,2}$
$E_{CVk,l}$
Get minimum
E^*_{CV}

Figure 1. Cross validation for model selection

3. RESEARCH METHOD

In this section, we outline the methodology, step by step, of learning the node selection strategy *NS* and variable branching strategy *VB* using parameter tuning**.** Then, we present the experiment configuration.

3.1. Collecting Datasets

We use the MIPLIB2010 library as instances to which we apply the Branch-and-Bound algorithm featured by reliability branching rule and best estimate selection rule. Then we extract information of features described before. Note that the best estimate selection rule is the default one is various optimization tools like SCIP. Here is the pseudo-code of the data collection step as shown in Algorithm 2.

```
For I instance in MIPLIB2010
    Solve I by branch-and-bound ruled by reliability branching and node estimate selection rule and
    collect feature values.
Return the list of features values
```

Algorithm 2. Data Collection

3.2. Learning NS and VB

First, we divide collected data into two sets, one used for training and validation and the other for test. By applying twice cross validation based model selection descripted above, we firstly learn the score function of every node in the nodes queue. So the node having the best score $NS(node)$ will be chosen in branch-and-bound. Secondly, we will learn the score function *VB* of variable branching selection.

3.3. Experiments

In this sub-section, we present the lab-test used to experiment our methodology and we give the pseudo-codes used for tests.

3.3.1. Experimentation configuration

We used SCIP 3.2.1 for the raison that is the best in open-source and free tools [23]. Moreover, for SVM algorithm and model selection, we used the package e1071 [23] of the language R 3.2.5 known to be among the most performant languages in implementation of SVM algorithms [13].

The cost range used is $\{10^{-4}, 10^{-3}, \dots, 10^{5}\}$ and the gamma range is $\{2^{-8}, 2^{-7}, \dots, 2^{1}\}$. These ranges cover too small and too high values of cost and gamma. The OS used is Debian 7 32 bits, 8 Go RAM, Intel 2.40 GHz Processor. We use for MILP instances the benchmark set of MIPLIB2010 [28] to collect data, valid it, and to test resulting models. For training and validation set, we took the following instances as shown in Figure 4.

30n20b8.mps acc-tight5.mps aflow40b.mps air04.mps app1-2.mps ash608gpia-3col.mps bab5.mps beasleyC3.mps biella1.mps bienst2.mps binkar10_1.mps bley_xl1.mps bnatt350.mps core2536-691.mps cov1075.mps csched010.mps danoint.mps dfn-gwin-UUM.mps eil33-2.mps eilB101.mps enlight13.mps enlight14.mps ex9.mps glass4.mps gmu-35-40.mps iis-bupa-cov.mps iis-pima-cov.mps lectsched-4-obj.mps m100n500k4r1.mps macrophage.mps mcsched.mps mik-250-1-100-1.mpsmine-166-5.mps mine-90-10.mps n3div36.mps n4-3.mps neos-1109824.mps neos-1337307.mps neos-1396125.mps neos13.mps neos-1601936.mps neos18.mps neos-686190.mps neos-849702.mps neos-916792.mps neos-934278.mps net12.mps

Figure 4. Training and validation sets

These instances about tens of thousands of rows and columns. The total description is available in [28]. Concerning the validation set, it contains approximatively fifth of number of mentioned instances above [7]. Finally, the node limit is fixed to five hundred nodes and running time limit is fixed to six-hundred seconds.

3.3.2. Pseudo-codes

In the solving process, the algorithm as it is implemented in SCIP is executing numerous event codes related to some events. Next, we will describe these events, and give the pseudo-code relative to each one. Besides, main events modified are respectively:

3.3.3. Node selection event

This event occurs when the algorithm is in the phase of selecting the next node to solve. The criteria of selection is determined by the strategy implemented. Note that this event code is also executed even in the root node selection. In this event, we implement the node selection rule score function *NS* that is already established. Moreover, it calculates the score for each node in the node list. Finally, it returns the node with the maximum score. The pseudo-code is the Algorithm 3.

```
For each leap n
        Calculate $NS$(n)
Return the leap with maximum $NS$(n)
```

Algorithm 3. Node selection event pseudo-code

3.3.4. Variable branching event

This event occurs when the algorithm is in the phase of selecting a branching variable of a node already solved and had gave a non-integral value. In this event, we implement the variable selection rule score function *VB*. The input of this function is the values of the node-based features and the variable-based features. The node-based features are related to a fixed node and they are calculated for the current node as a first step. Besides, we calculate, the value of *NS* on the current node. Then we calculate the values of the variable-based features and consequently the value of *VB* for each variable. In the output, we will return the variable with the maximum score *VB.* Finally, we branch on the selected variable and created two related children nodes. The pseudo-code is as Algorithm 4.

```
Calculate the value of the node-based features
  Calculate the value of NS relative to the present node.
  For each branching variable candidate
          Calculate values of variable-based features
          Calculate VB in terms of calculated features.
  Return the max of VB and relative variable
  Create two children nodes relying upon the chosen variable
  Calculate possible node-based features values of the two children
```

Algorithm 4. Variable branching event pseudo-code

3.3.5. Node Solved event

This event occurs when the algorithm is the state of leaving the node already solved. We use this event to calculate the values of LP Objective Value of the current node, and the reduced objective values gain. The pseudo-code is the Algorithm 5.

```
Get the LP Objective Value of the present node.
Calculate the reduced objective values gain
```

Algorithm 5. Node solved event pseudo-code

4. RESULTS

We present in this section, a comparison between algorithms resulted from our approaches and standard branch-and-bound algorithm. The comparison is done in term of *Running Time, Dual Bound* and *Number of Solved nodes.* The dual bound being a quantity converging to the optimal solution if it exists. The greater value of dual bound is the best one.

To get to this comparison, we did three different solving configurations on test set. The first is done by standard branch-and-bound (SBB) algorithm ruled by reliability pseudo-cost branching rule and best estimate node selection rule. Then for the second and third, we used our algorithms with SVR without model selection (ABB) and with model selection (ABB+MS). The results are detailed in the Table 1.

Table 1. Results of experimentation

Instance Name				SBB, ABB and ABB+MS respectively								
	Type	# of Rows	# of Columns	Dual Bound			Running Time			# of Solved nodes		
noswot.mps	MIP	182	128	-43	-43	-43	0.41	0.53	0.59	500	500	500
ns1208400.mps	MBP	4289	2883	0	0	0	355.58	153.92	152.40	500	500	500
ns1688347.mps	BP	4191	2685	2	14	14	251.51	126.59	126.02	500	500	500
ns1830653.mps	MBP	2932	1629	7507.43	8333.00	8333.00	79.36	600	600	500	303	303
opm2-z7-s2.mps	BP	31798	2023	-12412.11	-12602.02	-12455.10	271.64	283.73	283.73	500	500	500
pigeon-10.mps	MBP	931	490	-589.54	-577.46	-589.54	3.40	1.62	1.73	500	500	500
qiu.mps	MBP	1192	840	-10000	-10000	-10000	14.26	32.14	32.11	500	500	500
rail507.mps	MBP	509	63019	172.20	172.259	172.20	600	600	600	74	211	211
ran16x16.mps	MBP	288	512	3233.32	3324.73	3324.73	1.17	1.46	1.44	500	500	500
reblock67.mps	BP	2523	670	-36498090.97	-36159196.29	-36159196.29	23.22	29.00	28.93	500	500	500

In this test, we had ten instances from MIPLIB2010. These instances have three different types. The first one is mixed integer program (MIP) that regroups integer and continuous variables. The second is mixed binary program (MBP), which includes both continuous and binary variables. And the final one is Binary program (BP) that contains exclusively binary variables.

These results show up that our approaches give equivalent if not better dual bound comparing to standard branch-and-bound in term of dual bound in 80% of cases except from opm2-z7-s2 and ran16x16 instances. Another important result, is that our last approach gives equivalent or better running time comparing our last approach in 80% of cases. Also, when comparing it to the standard branch and bound algorithm ruled by reliability branching and node best esimate rule, our approach gives better or equivalent result in about half of total instances.

We noticed that there is an empirical relation between the performance of dual bound and the number of constraints of the problem from the one hand, and a relation between the performance of running time and the number of variables from the other hand. To concretize these last points, we plot these in Figure 5.

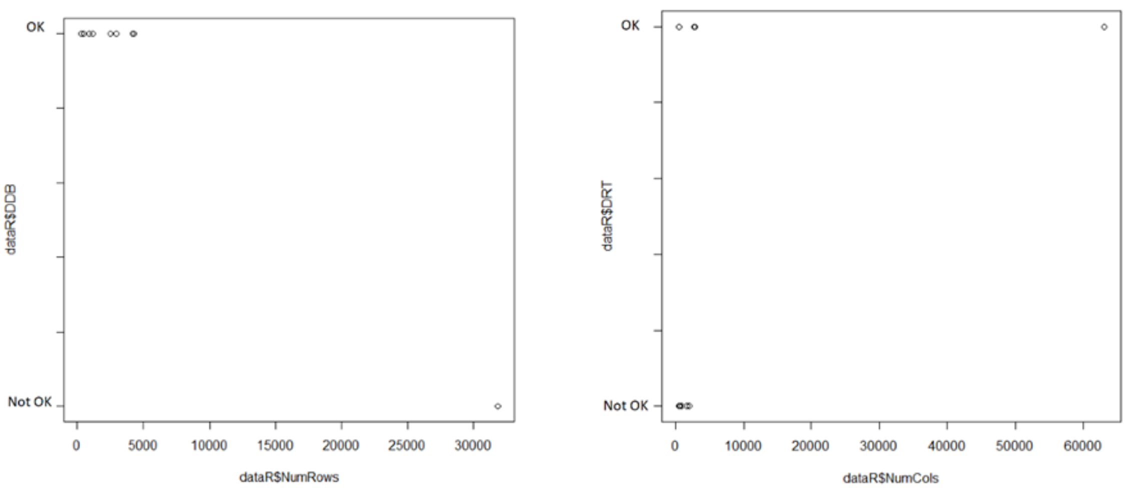


Figure 5. Increase or decrease of dual bound and running time respectively

The left-hand figure shows that instances with less that 5000 constraints gave better dual boud for our approaches comparing to standard branch and bound. As a matter of fact, the opm2-z7-s2 instance, which is represented by the isolated point in the down-rignt side has approximatively 31000 variables. Concerning the right-hand figure, it shows that instances with more than 2500 variables, increased the performance of running time, when resolved by our approaches, especially for ns1208400, ns1688347 and rail507 instances.

5. CONCLUSION

In this paper, we add parameter tuning to infer better configuration of SVM. Saying this, we used $\varepsilon -$SVM regression learning algorithm known for his high accuracy to learn branch-and-bound algorithm node selection and variables branching strategies. These choices lead to better results comparing to reliability pseudo cost rule and best estimate selection rule, which are known to be from the best in literature. In perspectives, we will work on eliminating noise in data, compare with different learning algorithms available in literature.

REFERENCES

[1] He H., et al., “Learning to search in branch and bound algorithms,” *Advances in neural information processing systems*, pp. 3293-3301, 2014.
[2] Moll R., et al., “Learning instance-independent value functions to enhance local search,” *Advances in Neural Information Processing Systems*, pp. 1017-1023, 1999.
[3] Karzan F. K., et al., “Information-based branching schemes for binary linear mixed integer problems,” *Mathematical Programming Computation*, vol/issue: 1(4), pp. 249-93, 2009.
[4] Davey B., et al., “Efficient intelligent backtracking using linear programming,” *INFORMS Journal on Computing*, vol/issue: 14(4), pp. 373-86, 2002.
[5] Chen D. S., et al., “Applied integer programming: modeling and solution,” John Wiley & Sons, 2011.

[6] Achterberg T., et al., “Branching rules revisited,” *Operations Research Letters,* vol/issue: 33(1), pp. 42-54, 2005.

[7] Abu-Mostafa Y. S., et al., “Learning from data,” New York, NY, USA, AMLBook, 2012.

[8] C. Rudin, “Prediction: Machine Learning and Statistics,” 2012.

[9] X. Wu, et al., “Top 10 algorithms in data mining,” *Knowledge and information systems,* vol/issue: 14(1), pp. 1-37, 2008.

[10] B. Lantz, “Machine learning with R,” Packt Publishing Ltd, 2015.

[11] Alvarez A. M., et al., “A machine learning-based approximation of strong branching,” *INFORMS Journal on Computing*, vol/issue: 29(1), pp. 185-95, 2017.

[12] Aoun O., et al., “Investigation of hidden markov model for the tuning of metaheuristics in airline scheduling problems,” *IFAC-PapersOnLine,* vol/issue: 49(3), pp. 347-52, 2016.

[13] E. Afia A., et al., “Hidden markov model control of inertia weight adaptation for Particle swarm optimization,” *IFAC-PapersOnLine,* vol/issue: 50(1), pp. 9997-10002, 2017.

[14] E. Afia A. and Sarhani M., “Performance prediction using support vector machine for the configuration of optimization algorithms,” *Cloud Computing Technologies and Applications (CloudTech), 2017 3rd International Conference,* pp. 1-7, 2017.

[15] E. Afia A., et al., “Fuzzy logic controller for an adaptive Huang cooling of simulated annealing,” *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, pp. 64, 2017.

[16] Bouzbita S., et al., “A novel based Hidden Markov Model approach for controlling the ACS-TSP evaporation parameter,” *Multimedia Computing and Systems (ICMCS), 2016 5th International Conference*, pp. 633-638, 2016.

[17] Lalaoui M., et al., “Hidden Markov Model for a self-learning of Simulated Annealing cooling law,” *Multimedia Computing and Systems (ICMCS), 2016 5th International Conference,* pp. 558-563, 2016.

[18] Lalaoui M., et al., “A self-adaptive very fast simulated annealing based on Hidden Markov model,” *Cloud Computing Technologies and Applications (CloudTech), 2017 3rd International Conference,* pp. 1-8, 2017.

[19] E. Afia A., et al., “The Effect of Updating the Local Pheromone on ACS Performance using Fuzzy Logic,” *International Journal of Electrical and Computer Engineering (IJECE),* vol/issue: 7(4), pp. 2161-8, 2017.

[20] Bouzbita S., et al., “Dynamic adaptation of the ACS-TSP local pheromone decay parameter based on the Hidden Markov Model,” *Cloud Computing Technologies and Applications (CloudTech), 2016 2nd International Conference,* pp. 344-349, 2016.

[21] Kabbaj M. M. and E. Afia A., “Towards learning integral strategy of branch and bound,” *Multimedia Computing and Systems (ICMCS), 2016 5th International Conference*, pp. 621-626, 2016.

[22] E. Afia A. and Kabbaj M. M., “Supervised learning in Branch-and-cut strategies,” *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications,* pp. 114, 2017.

[23] http://Scip.zib.de.

[24] http://Miplib.zib.de.

[25] David M., “Support Vector Machines: The Interface to libsvm in Package e1071,” David. Meyer@ *R-Project. org*. 2017.

BIOGRAPHIES OF AUTHORS

M. Kabbaj Mohamed Mustapha is a PHD student in at National School of Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He obtained M.Eng. in 2013 in Computer Science from National School of Computer Science and Systems Analysis. Research areas of interest are Machine Learning, Combinatorial Optimization, Stochastic Programming and Feature Selection.

Abdellatif El Afia is an Associate Professor at National School of Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He received his M.Sc. degrees in Applied Mathematics from University of Sherbrook. He obtained his Ph.D. in 1999 in Operation Research from University of Sherbrook, Canada. Research areas of interest are Mathematical Programming (Stochastic and deterministic), Metaheuristics, Recommendation Systems and Machine Learning.