Report Research Project

Solving NP-hard graph spanning problems using kernelization, linear programming and machine learning

Ramón Daniel Regueiro Espiño, Avraham Rosenberg & Abdoulaye Sakho

2021/2022

École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Contents

1 Introduction

Nowadays, our society uses network structures for different reasons like the flow of information. For this case, to build a reliable flow reducing its costs can be made studying the involved terminals and its links structure. In this project, we'll try to build an algorithm to find an efficient structure without a high computational cost.

We're going to modelize the problem creating a connected graph which will represent the terminals as nodes and its links are edges. In this case, the solution can be represented obtaining a spanning tree from the connected graph. Although, in some cases, like for optical networks, a limitation of the number of nodes of more than two edges might be needed, which will be called branch nodes. So, we're going to focus on this special case.

1.1 Construction of the spanning tree using ILP

In this subsection we'll introduce how can we build our problem as an Integer Linear Programming (ILP) problem.

For this problem, we'll always consider a connected graph $G = (V, E)$. The first one is based in [Martin, 1991] ILP formulation for the minimum weighted tree problem. For our case, we introduce a new binary variable indicating if the node will be a branch node and the loss function will be the number of branch nodes of the spanning tree.

$$
\begin{aligned}
\min \quad & \sum_{i \in V} y_i \\
\text{subject to:} \quad & \sum_{e \in E} x_e = n - 1, \\
& z_{k,i,j} + z_{k,j,i} = x_e, \\
& \sum_{s>i} z_{k,i,s} + \sum_{h<i} z_{k,i,k} \leq 1, \\
& \sum_{s>k} z_{k,k,s} + \sum_{h<k} z_{k,k,h} \leq 0, \\
& \sum_{(k,s) \in E} z_{i,k,s} - 2 \leq (n-1) y_i \\
& x_e \in \{0,1\} \ \forall e \in E, \quad y_v \in \{0,1\} \forall v \in V, \quad z_{k,i,j} \in \{0,1\} \ \forall k \in V, \forall e = \{i,j\} \in E.
\end{aligned}
\tag{1}
$$

In the equation (1), y_i indicates whether the node i is a branch node. We hihglight that the original problem considered linear variables. However, we'll need to consider them as binary variables. Otherwise, we'll obtain a huge number of edges with non-integer values.

The second formulation is based on [Cerulli et al., 2009]. In this case, we consider the ILP as a flow problem where we try to send an unique flow for all the graph.

$$
\begin{aligned}
\min \ & \sum_{v \in V} y_v \\
\text{subject to: } & \sum_{e \in E} x_e = n-1, \\
& \sum_{(s,v) \in A^+(s)} f_{sv} - \sum_{(v,s) \in A^-(s)} f_{vs} = n-1, \\
& \sum_{(v,u) \in A^+(v)} f_{vu} - \sum_{(u,v) \in A^-(v)} f_{uv} = -1 \ \forall v \in V \setminus \{s\}, \\
& f_{uv} \leq (n-1) x_e \ \forall e \in \{u,v\} \in E, \\
& f_{vu} \leq (n-1) x_e \ \forall e \in \{u,v\} \in E, \\
& \sum_{e \in A(v)} x_e - 2 \leq (n-1) y_v \ v \in V, \\
& x_e \in \{0,1\} \ \forall e \in E, \quad y_v \in \{0,1\} \forall v \in V, \quad f_{u,v}, f_{vu} \geq 0 \ \forall e = \{u,v\} \in E.
\end{aligned} \tag{2}
$$

As it can be seen in the equation (2), we have a fixed node, s, called the source node. We consider $n-1$ unities of flow going out from this node and, in the total flow of each other node, it has to arrive one unity of flow. In this way, we'll obtain a connected graph. Also, we'll consider the used edges as the ones that have a strictement positive flow.

2 Preprocessing

In this section we're going to introduce some previous steps that might reduce the Linear Program size. In the first place, we're going to consider the case of V_1 and V_3 nodes. In the subsection 2 we'll introduce a graph reduction.

2.1 V_1 and V_3 nodes

In this subsection we're going to see a way of reducing the number of potential candidates for being a branch node detecting either they're non-branch nodes or branch nodes in polynomial time. The main ideas of this part are obtained from [Merabet et al., 2018].

In the first place, we'll consider a node v as V_1 if, and only if, there are only one or two edges of the graph with a side being v. In this case, as there won't be more than two edges with a side being v, it's impossible for v to be a branch node. From this point, we'll denote $\{V_1\}_G$ the set of V_1 nodes of our graph G.

In the second place, we'll consider a node v as V_3 if, and only if, erasing all its edges and itself from the graph we'll obtain at least three different connected components. This implies that the MBVST will have at least three edges with v as one of its sides. Consequently, v is a branch vertex of the MBVST. We highlight that a node cannot be V_1 and V_3 at the same time.

We'll consider as V_2 the nodes that aren't V_1 or V_3. We can reduce our problem to detect the minimum number of V_2 that are in a spanning tree of the given graph.

2.2 Reduction of the graph

In this subsection we are going to introduce a graph reduction using what we've introduced in 2.1.

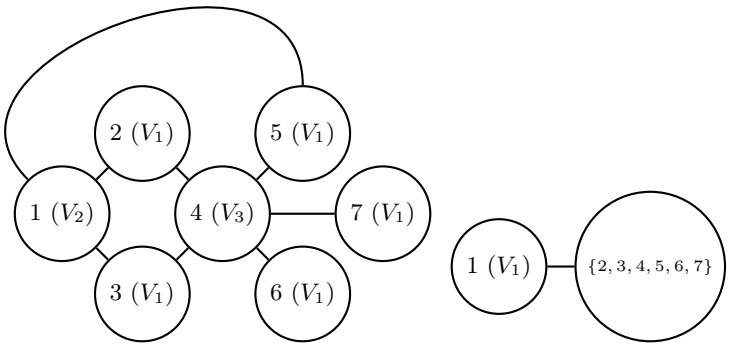
As we have already seen, V_3 and V_1 nodes might not be considered in the Linear Program, as we already know whether they are or they aren't branch nodes. So, we are going to look deeper in these nodes.

For the V_1 case, we know that all its edges we'll be present in the MBVST tree. Also, for a edge connecting two different V_3, we'll not increase the number of branch vertex. So, if we have and edge with both sides being nodes from $\{V_1\}_G \cup \{V_3\}_G$ we know that considering the edge as present in the spanning tree won't increase the number of branch vertex.

Then, we'll add all the edges of these type and we'll obtain different connected components. For each connected component, we build a tree.

Finally, we'll consider each connected component as only one node of a graph G' with an edge between two nodes if and only if there is an edge between a node of each connected component.

Then, each V_2 will be a connected component of one node. Although, it might only have all its edges connected to two different connected component. In this case, this V_2 node will become a V_1 node. This implies that some V_2 might become V_1, reducing the size of the graph and the number of potential candidates for being branch nodes.



(a) Before (b) After

Figure 1: Tranformation using the method for a graph G.

We can be the main idea of this preprocessing part in the Figure 1. In this case, without using the algorithm we already know that there is only one branch vertex for our MBVST without needing to even build the linear program as we don't have any V_2 node left after the preprocessing.

Proposition 2.1. *A V_2 node cannot be transformed in a V_3 node using this process.*

Demonstration 2.1. By reduction to absurdity.

We suppose that it exits a V_2 node that become a V_3 node after a number of applications of the process. So, in the obtained graph, if we delete the considered node and its edges, there are at least three different connected components. In the original graph, there can't be any path from one of this connected components to another different without going through the considered node. Then, in the original graph this node will be a V_3 node, which contradicts the fact that it was a V_2 node.

Remark. *For the code, we highlight that once that we'll consider a V_2 become V_1 there will be only two edges, one connected to the connected component which it'll be introduced and*

another one that we consider free (it doesn't matter). So, we can add it to any node of the connected component without problem. Afterwards, in any other iteration of the loop we can always consider its free edge without making it a branch node. This is usefull to justify why we don't have to do a precise selection joining two different connected components containing V_2 nodes that became V_1.

2.3 Considerations of the k-MBVST case

Until now, we have treated only branch vertex as nodes with more than two edges. Although, we could try the case of minimizing the number of k-branch vertex, *ie* the number of vertex with more than k edges. In this subsection we'll introduce the generalization to this case. Althoug, we'll not treat it in our experimental part.

In the first place, for the ILP formulation it's enough to change in the last condition of the formulation the two for the considered k. Indeed, we'll obtain that y_v will be 1 if the spanning tree uses at least k edges in this node.

In the second place, the same methodology could be applied to this case. To illustrate an example, as it is shown in [Merabet et al., 2018], for the preprocessing we can consider V_1 as the nodes with no more than k edges and V_3 the nodes that will generate at least $k+1$ connected components.

2.4 Experimental results

In this subsection we are going to introduce the experimental results for the Operational Research part. All of our computations were performed using networkx, [Hagberg et al., 2008], and Cplex, [Cplex, 2009], as the underlying ILP solver. Firstly, we'll compare both ILP formulations of our problem. Then,

2.4.1 Evaluation of both formulations

In the first place, we observed that the formulation based on the equation (1), which we'll call *Martin's formulation*, has a higher computational cost than the formulation of the equation (2). From now on, we'll call this formulation the *Flow formulation.*

An explication of this different computational computational costs might be the fact of using all variables as binary variables in *Martin's formulation.* This comes from the fact that LP binary variables requires a higher resolution use for solving the problem than linear variables.

So, as we want to reduce the computational cost, we decided to focus our work on the *Flow formulation* for the continuation of this project.

2.4.2 Analysis of the effect of the graph reduction

In the second place, we'll analyze the effect of the reduction of the graph. In this case, we'll consider only the small instances because the time and computational cost of considering all the instances is too high.

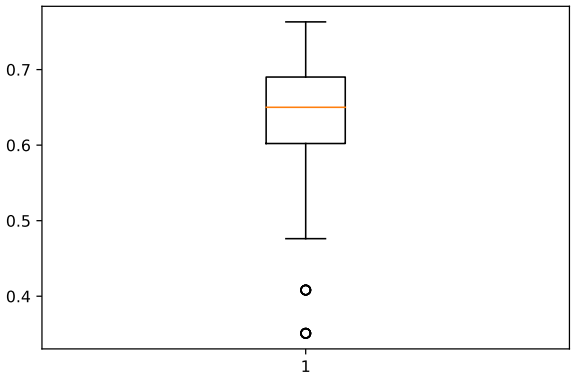


Figure 2: Relation between the number of nodes after the reduction of the graph and the initial number of node of the graph.

As it can seen from the Figure 2, we obtain a remarkable reduction of the number of nodes for our problem, with a mean of having only 63.8% of the original nodes only. This node reduction implies a considerably decrease of the number of binary variables making the ILP algorithm faster and with a smaller computational cost.

3 Data Science

The aim of this part is to use Machine Learning tools in order to improve the Integer Linear Program time resolution. In the first place, we'll estimate some characteristics of the global graph. In the second place, we'll focus on each node characteristics.

3.1 Predicting Time and Number of branch nodes

In this subsection we're going to focus on estimating the time resolution of the ILP and the number of branch nodes.

3.1.1 The first Data Frame

For this part we resolve 354 graphs with the ILP, creating a very exhaustive Data Frame which contains different variables that might be useful to build our models.

	nbre_sommets	nbre_arretes	Densite	Diametre	nbre_noeuds_branch
count	354.000000	354.000000	354.000000	354.000000	354.000000
mean	357.203390	433.923729	0.026584	9.494350	83.327684
std	288.589082	321.388379	0.056026	2.074046	86.149879
min	20.000000	27.000000	0.002096	3.000000	0.000000
25%	140.000000	193.000000	0.003751	9.000000	19.000000
50%	250.000000	297.000000	0.011156	9.000000	40.000000
75%	600.000000	740.000000	0.019836	11.000000	155.750000
max	1000.000000	1124.000000	0.300000	12.000000	313.000000

	nbre_noeuds_V1	nbre_noeuds_V3	nbre_noeuds_V2	nbre_cyclomatique
count	354.000000	354.000000	354.000000	354.000000
mean	207.522599	74.528249	75.152542	77.720339
std	174.860514	82.526504	50.168661	51.704250
min	0.000000	0.000000	6.000000	8.000000
25 %	76.000000	13.000000	37.250000	38.000000
50 %	130.000000	33.000000	61.000000	68.000000
75 %	363.000000	144.750000	101.250000	105.000000
max	643.000000	298.000000	223.000000	225.000000

Table 1: Information obtained from the created Data Frame.

From the Table 1, we can see the different 354 observations for considered variables. They are:

- `nbre_sommets`: It represents the number of nodes of the starting graph.

- `nbre_arretes`: It represents the number of edges of the starting graph.
- `Densite`: It represents the Density of the starting graph.
- `Diametre`: It represents the Diameter of the starting graph.
- `nbre_noeuds_V1`: It represents the V_1 nodes, defined in the Section 1.1.
- `nbre_noeuds_V3`: It represents the V_3 nodes, defined in the Section 1.1.
- `nbre_noeuds_V2`: It represents the V_2 nodes, defined in the Section 1.1.
- `nbre_cyclomatique`: It represents the cyclomatic number of the graph.
- `nbre_V2_devenu_branch`: It represents the V_2 nodes which became branch nodes.
- `temps1`: It represents the time resolution with the ILP.

From them, our considered target variables will be the time resolution of the LP and the number of branch nodes.

- The **time resolution** of the ILP: if the time resolution will be very low, we can use directly the ILP program. If it will be very high, we will use Machine Learning tools.
- The **number of branch nodes**: More specifically we will determine `nbre_V2_devenu_branch`. We will explain later.

3.1.2 Data Visualisation

Correlations:

A high correlation between explanatory variables might imply that our models might not estimate properly the effect of each explanatory variable in the target variable. Also, it allows us to have an intuitive vision on the relation between each explanatory variable and the target variable.

As it can be seen in the Figure 3, for **temps1** we see that `nbre_sommets`, `nbre_arretes`, `Diametre`, `nbre_noeuds_V1` and `nbre_noeuds_V3` are very correlated to our target. `nbre_noeuds_V2` and `nbre_cyclomatique` are also correlated to `temps1`. However, there is a strong correlation between `nbre_arretes` and `nbre_sommets` $(0, 99)$. Consequently, if we use a linear model, we can just choose `nbre_arretes`. `nbre_noeuds_V1` and `nbre_noeuds_V3` are also very correlated. So, for example, we can just choose `nbre_noeuds_V3`.

For `nbre_V2_devenu_branch`, as it is shown in the Figure 3, `nbre_sommets`, `nbre_arretes`, `Diametre`, `nbre_noeuds_V1` and `nbre_noeuds_V3` are very correlated to our target. `nbre_noeuds_V2` and `nbre_cyclomatique` are also correlated to `nbre_V2_devenu_branch`. Although, as for the case of `temps1`, we see that `nbre_arretes` and `nbre_sommets` are very correlated ($(0, 99)$. Consequently, we highlight the fact that a variable selection might be adequate to build our predictive models.

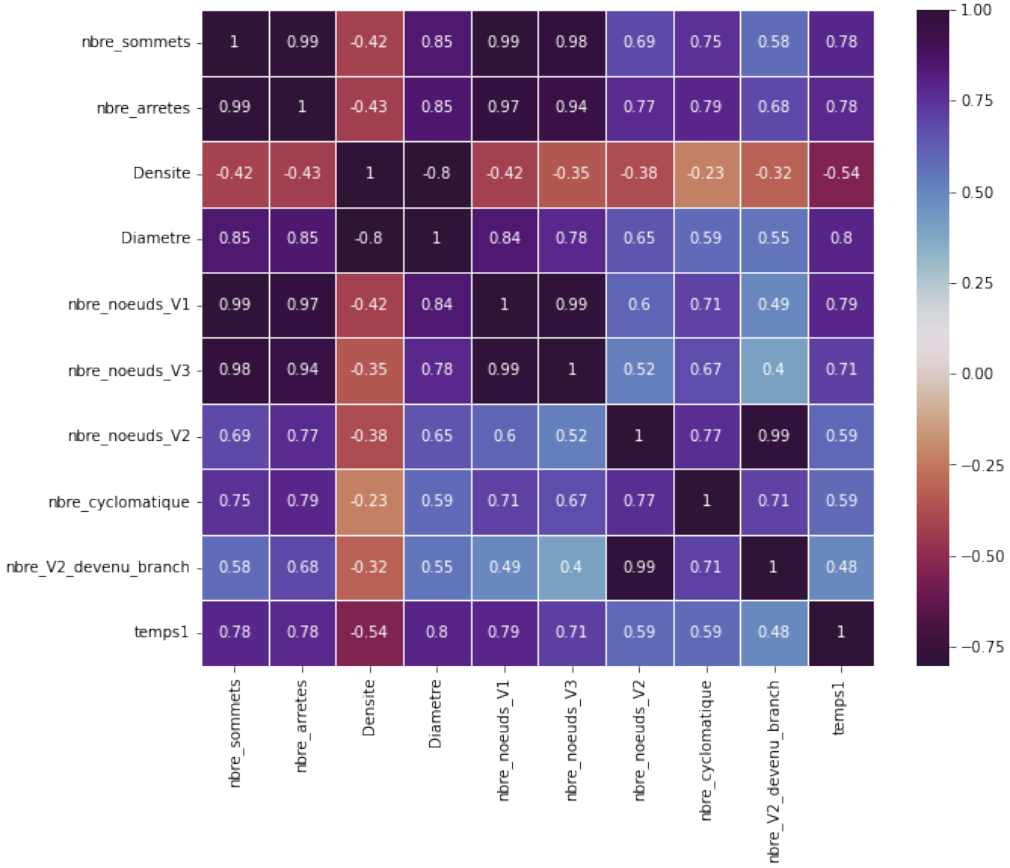


Figure 3: Correlations

3.1.3 Linear models

First, we use two different linear models. One to predict `temps1` and the other to predict `nbre_V2_devenu_branch`.

`temps1`: The simple linear regression gives correct results ($R^2 = 0.77$ and $RMSE = 2251s$).

`nbre_V2_devenu_branch`: The simple linear regression directly gives very good results ($R^2 = 0.94$ and $RMSE = 1,55$).

The simple linear model is perfectly suited to predict `nbre_V2_devenu_branch`. Although, we consider that we must perform our model for the estimation of `temps1`.

3.1.4 Feature selection

In the previous model, we use all features in the DataFrame. It is not the better way to work. Indeed, if features are correlated between them, there are some problems in the

error calculation (matrix will not be invertible).

Consequently, for `temps1` prediction, we selected features by two different ways:

- We analyzed the correlations and selected features "by hands". We obtained $R^2 = 0,74$ and $RMSE = 2350s$. It is better than the previous model.
- We use incremental methods like forward. We obtained $R^2 = 0,76$ and $RMSE = 2273s$. It is also better than the previous model.

3.1.5 Random Forest

We work with Random Forest. This algorithm is fast to run and gives results which can be generalized.

We use this algorithm and we obtained betters results for `temps1`:

- With the parameters set by default, we obtained $R^2 = 0,83$ and $RMSE = 2158s$
- Tuning the parameters to obtaine a better performance, combining the use of `GridSearchCV` and by hand, we decided to change the depth of the trees. In this case, we obtained an imporvement on our results, $R^2 = 0,87$ and $RMSE = 1870s$

3.1.6 Final Models

After a comparison of the different built models, that it can be seen in the Table 2, we decided to keep for our predictions the one with better R^2 and $RMSE$. This model is the Random Forest with `max_depth=3`.

Méthode	R2	MSE	RMSE
Linear_Regression_Train/Test	0.808255	3.703865e+06	1924.542747
Linear_Regression_KFold	0.770367	5.015873e+06	2236.846571
Linear_feature_selection_hands	0.736704	5.667870e+06	2376.235690
Linear_Regression_forward	0.780557	4.990593e+06	2229.907485
Random_Forest	0.740468	4.612496e+06	2147.672224
Random_Forest_KFold	0.854517	4.021853e+06	1945.278512
Random_Forest_changing_parameters	0.871350	3.424323e+06	1838.558711

Table 2: Measure performances for the different models.

For the estimation of the number of V_2 nodes that become branch nodes, we obtained lower results using the feature selection and the Random Forest. Consequently, we decided

to keep for our predictions the initial Linear model. This model, as it was described, has as performance measures $R^2 = 0.94$ and $RMSE = 1,55$.

For the model predicting the number of branch nodes, we realized a Quantiles Regressions. We oserved that 90% of our predicted données are in the in the interval $]5, 6[$.

3.2 Predicting probability of V2 nodes to be a branch nodes

In the previous part we predicted the time resolution of the Integer Linear Program and the number of V_2 nodes which became branch nodes. Now, in this subsection we go more deeper and we want to predict which V_2 node became a branch node.

For this part we use a local approach.

3.2.1 The Data Frame

To build this dataframe, we used the same graph instances as before and their optimal graph associated with the problem.

	nbre_voisins_V1	nbre_voisins_V2	nbre_voisins_V3	nbre_comp_connexes	nbre_base_cycle	Is_V2
count	26604.000000	26604.000000	26604.000000	26604.000000	26604.000000	26604.000000
mean	0.563599	2.322884	1.184596	1.266238	18.306533	0.113968
std	0.983940	1.417273	1.179083	0.441999	24.463379	0.317778
min	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000
25%	0.000000	1.000000	0.000000	1.000000	3.000000	0.000000
50%	0.000000	2.000000	1.000000	1.000000	5.000000	0.000000
75%	1.000000	3.000000	2.000000	2.000000	27.000000	0.000000
max	6.000000	12.000000	4.000000	2.000000	128.000000	1.000000

Table 3: Information obtained from the Data Frame.

In the Table 3, we can see that we have considered 26604 V_2 nodes. Also, the considered possible explanatory variables and the target variables of the Data Frame are:

- `nbre_voisins_V1`: The number of neighbours of type V_1 that the node in question has from the original graph.
- `nbre_voisins_V2`: The number of neighbours of type V_2 that the node in question has from the original graph.
- `nbre_voisins_V3`: The number of neighbours of type V_3 that the node in question has from the original graph.
- `nbre_comp_connexes`: If we delete this node from the original graph, the number of related components that we obtain for this new graph.
- `nbre_base_cycle`: By taking a random cycle base from the original graph. The number of times the node in question appears in the different components of the base.

- `Is_V2`: After solving the original graph. This variable is worth 1 if the node of V_2 in question is a branching node, 0 otherwise

We use this part in order to predict:

- Whether for a given node of V_2 it is a branch node or not in the associated solution graph. It is thus `Is_V2` our target variable.
- We thus carry out classification.
- We will obtain for the models in reality a probability that the node in question is a branching node or not. It is this probability that will be exploited

3.2.2 Data Visualisation

Correlations:

As before, we look at the correlations between our variables in order to perform an analysis:

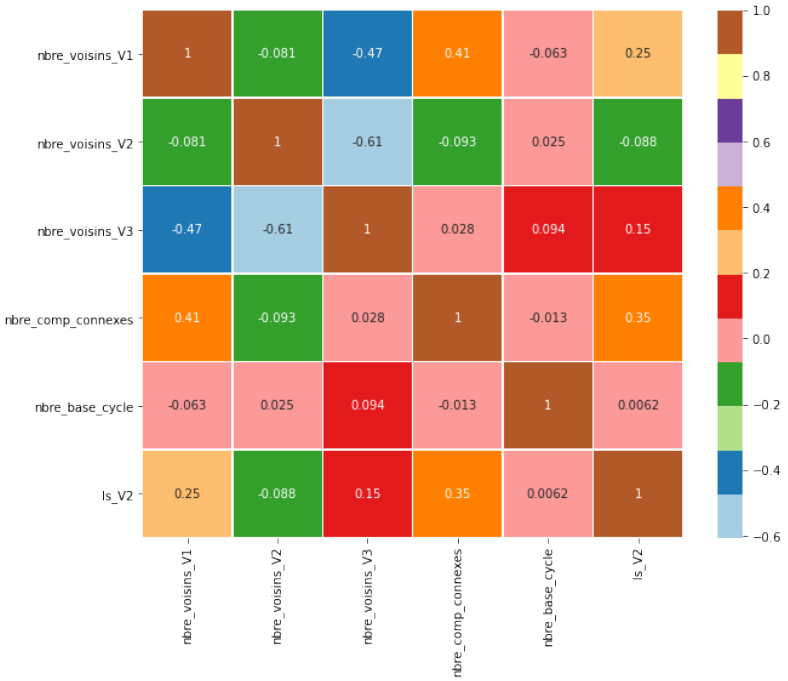


Figure 4: Correlations

In the Figure 4, our target variable is represented by the variable `Is_V2`. In this case, we observe the correlations between the differente variables. We highlight the fact that all the considered possible explanatory variables don't have a huge correlation with the

target variable. However, `nbre_comp_connexes`, `nbre_voisins_V1` and `nbre_voisins_V3` are correlated to it.

3.2.3 Logistic Regression

As our target variable is a binary variable, we're going to predict the probability of an observation. In this case, we consider logistic regression instead of the linear regression as it was considered initially to predict the time of resolution of the ILP or the number of branch nodes of the graph.

After building a logistic regression model using Python, we obtained the following confusion Matrix for a test set:

	True class	
	Positive	Negative
Positive	(TP) 5734	(FP) 149
Negative	(FN) 523	(TN) 245

Table 4: Confusion Matrix for the logistic regression model.

From the Table 4, we highlight that our observations don't have a balanced number between real positives and real negatives. Indeed, TP number >> TN number (5734 >> 245). This phenomenon might developp biased estimates.

We also obtained the following metrics:

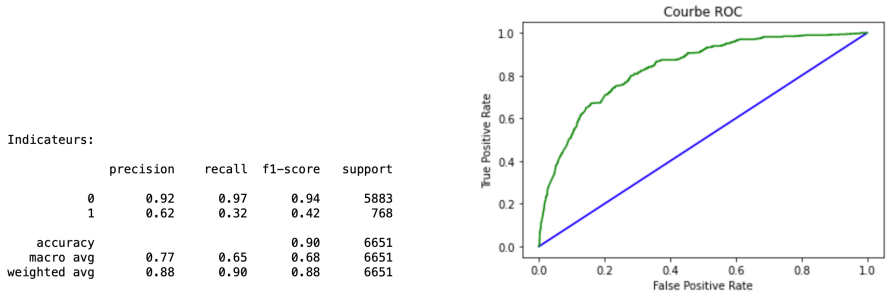


Figure 5: Evaluation of the logistic regression including the numerical values and the ROC curve.

As it can be seen in the Figure 5, the green curve is above the blue curve. We can determine the best threshold in order to categorize True Positive and False Postive.

3.2.4 Random Forest

After the creation of the logistic regression model, we tried to build a better model y using a Random Forest Classifier. We used the package Scikit-Learn agin and thanks to the function *GridSearchCV*. This function allowed us to find the best hyperparametres for: *max_ depth* and *n_ estimators*.

The 6 reprents the ROC cuvre for the best Random forest model. The AUC score associated to this curve is **0.8935**. This score is better than those of the logistic regression.

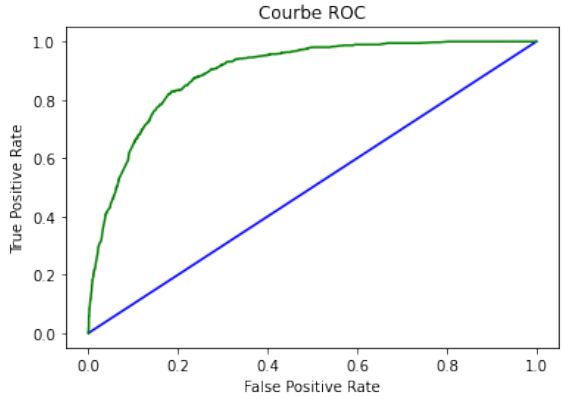


Figure 6: ROC curve for the Random Forest Classifier

3.2.5 Neural Network

Using Neural Network in another way to predict V2 nodes which became a branch node.

For this Neural network we use 3 group of components:

- `outpout shape`= 12 and `activation=relu`
- `outpout shape`= 8 and `activation=relu`
- `outpout shape`= 1 and `activation=sigmoid`

This model is very efficient: we obtain the accuracy **0.98**.

4 Conclusions

In this section, we'll introduce our final algorithm, combining Operational Research and Data Science. Also, we'll describe some possible further improvements.

4.1 The final algorithm

The final algorithm provided is described in the Figure 7. The main idea is to start predicting the time resolution of the algorithm. Then, if this time is not big, the algorithm will obtain the exact resolution using the preprocessing and the graph reduction. If the estimated time of resolution is big, the algorithm will do a prediction of the nodes and then solve the problem using predictions.

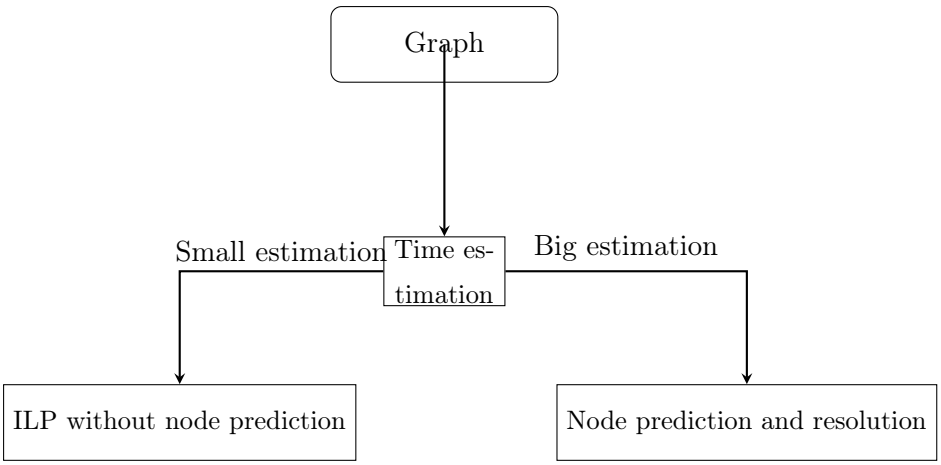


Figure 7: Flow chart of the final algorithm.

4.2 Further improvement

As further improvements, there are three main axes.

In the first place, we consider only the predictions before the reduction of the graph as we consider only this case for the data generation. Although, a possible improvement is to consider only nodes obtained after this reduction. This reduction of the different possibilities might imply better results as we'll know if some nodes aren't branch vertex in a polynomial time.

In the second place, a selection of different explanatory variables could be done to increase the performance of our estimations. For example, the selection of some nodes centrality measures.

In the third place, we can study the trade-off between the chosen prediction probability

for each node to consider it as branch or non-branch node and the time resolution of the ILP.

References

[Cerulli et al., 2009] Cerulli, R., Gentili, M., and Iossa, A. (2009). Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42(3):353–370.

[Cplex, 2009] Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

[Gargano et al., 2002] Gargano, L., Hell, P., Stacho, L., and Vaccaro, U. (2002). Spanning trees with bounded number of branch vertices. *Lect Notes Comput Sci*, 2380:355–365.

[Hagberg et al., 2008] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[Martin, 1991] Martin, R. (1991). Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10(3):119–128.

[Merabet et al., 2018] Merabet, M., Desai, J., and Molnar, M. (2018). A generalization of the minimum branch vertices spanning tree problem. In Lee, J., Rinaldi, G., and Mahjoub, A. R., editors, *Combinatorial Optimization*, pages 338–351, Cham. Springer International Publishing.