Automated Parameter Determination for Enhancing the Product Configuration System of Renault: An experience report

Hao Xu, Souheib Baarir, Tewfik Ziadi, Siham Essodaigui, Yves Bossu

▶ **To cite this version:**

Hao Xu, Souheib Baarir, Tewfik Ziadi, Siham Essodaigui, Yves Bossu. Automated Parameter Determination for Enhancing the Product Configuration System of Renault: An experience report. 28th International Conference on Engineering of Complex Computer Systems (ICECCS 2024), Jun 2024, Limassol, Cyprus. hal-04539975

HAL Id: hal-04539975

https://hal.sorbonne-universite.fr/hal-04539975v1

Submitted on 5 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Automated Parameter Determination for Enhancing the Product Configuration System of Renault: An experience report

Hao Xu
LIP6
France
hao.xu@lip6.fr

Souheib Baarir
EPITA
France
souheib.baarir@lip6.fr

Tewfik Ziadi
LIP6
France
tewfik.ziadi@lip6.fr

Siham Essodaigui
Renault
France
siham.essodaigui@renault.com

Yves Bossu
Renault
France
yves.bossu@renault.com

***Abstract*—The problem of configuring the variability models is pervasive in plenty of domains. Renault, a leading automobile manufacturer, has developed an internal product configuration system to model its vehicle diversity. This system is based on the well-known knowledge compilation approach and is associated with a set of parameters. Different input parameters have a strong influence on the system's performance. The parameters actually used are determined manually. Our work aims to study and determine these parameters automatically. This paper studies Renault's variability models and product configuration system and presents a parameter prediction model for this system. The results show the predicted parameters' competitiveness compared with the parameters by default.**

***Index Terms*—Variability model, knowledge compilation, machine learning, parameter tuning**

I. Introduction

Variability Modeling Problems [1] are very common in real life. In the car industry, such problems are important since they are related to different business activities, including engineering design, manufacturing, etc.

As an example of Variability Model (VM) in car industry, let's consider a VM M: it has a variable $model$, which represents the vehicle model with a value range in the domain $\{m_1, m_2\}$; a variable $fuel$ with the domain $\{petrol, diesel, lpg\}$. Then, variable dependencies describe activity-related constraints (business, technical, legal requirements, and many others). For instance, the constraint $model = m_1 \Rightarrow fuel = lpg$ leads to four possible combinations that form the different configurations for this vehicle. We refer to the set of possible configurations for a VM as the *configuration* (or the *solution*) space.

Renault, a world-leading automobile manufacturer [2], uses such VMs to model its vehicle range. Some ranges of its vehicles can reach 10^{32} possible configurations. With such a large configuration space, a common requirement is to be able to search for satisfying configurations based on users' queries. These queries can include consistency checks (to determine if a specified vehicle model exists) or requests for all the possible satisfied configurations, among others.

To deal with such requests, Renault has adopted a *knowledge compilation* [3] based approach. The idea of *knowledge compilation* is using symbolic structures (e.g., BDDs [4], SDDs [5], etc.) to represent the problem *configuration space*. An internal product configuration system has been developed to assist in building the configuration space for VMs using these symbolic structures.

Although this system provides an efficient method to handle large vehicle models, its performance can be limited by the memory size of the symbolic structure. The total size of such compiled structures can reach dozens of gigabytes, which puts pressure on memory usage. In Renault's configuration system, the compilation[1] of such symbolic structures is associated with several parameters that greatly influence the structure's size. Determining the best-performing parameters *manually* for each VM is difficult and tedious. This paper addresses this problem, proposes, and implements an automated parameters prediction model to obtain the best-performing parameters for each VM.

The paper is organized as follows: Section II presents related research; Section III describes the internal system with its associated parameters; Section IV presents the parameter tuning and prediction process; Section V presents the obtained results; and finally, Section VI concludes the paper and provides perspectives for future work.

II. Related Work

The performance of many algorithms relies heavily on carefully tuned parameter configurations based on user preferences or performance criteria [6]. Over the years, various

[1]Here, "compilation" refers to the process of building the configuration space in knowledge compilation terminology.

automatic parameter tuners have been proposed, which can be categorized into two types: *local* methods ¡¡¡¡¡¡and *model-based* methods.

In the category of local methods, notable examples include GGA [7], paramILS [8], SPOT [9], and irace [10]. These methods employ local search strategies in the configuration space, such as genetic algorithms, iterated local search, racing procedures, and more. They have demonstrated strong competitiveness when applied to solvers in diverse problem domains like mixed integer programming (MIP) [11], machine learning, and propositional satisfiability solving [12]. However, these tools suffer from a limitation of being problem feature-independent. In other words, the parameter tuning results are static and cannot be adjusted based on the characteristics of the input problem instance [13]. Theoretical and empirical studies on various algorithms and problems have shown that algorithm parameters are highly dependent on specific instance features of the target problem [14]. Indeed, the optimal parameter values can vary significantly with different input instance features, such as problem size [15].

On the other hand, model-based tuners consider problem features during the tuning process by leveraging machine learning techniques to build a model. Two notable examples in this category are SMAC and PIAC.

SMAC, sequential model-based algorithm configuration [16], is an algorithm that automates the process of finding the optimal parameter set for algorithms. It is often used for parameter tuning in machine learning models. SMAC iteratively runs the model with different parameter combinations and uses the results to learn which parameters are likely to yield good performance. It employs a Bayesian optimization strategy that takes into account both the model's performance and the uncertainty in estimated performances. This enables efficient search in the parameter space, achieving good solutions with fewer iterations compared to other methods.

PIAC (per instance algorithm configuration) relies on learning an empirical performance model (EPM) that can predict algorithm performance based on the instance and specified parameter settings [17]. The empirical performance model captures the relationship between instance features, parameters, and algorithm performance, enabling performance prediction with given features and parameters. However, PIAC faces challenges when it comes to predicting or searching for the best parameter setting in cases where the parameter space is large.

In our case, we aim to develop a feature-dependent, model-based automated algorithm tuner for each instance. In the following sections, we describe the problem and present our contributions.

III. Background & Problem Statement

The product configuration system used by Renault involves the generation of VMs and their associated configuration spaces. In this section, we provide a brief overview of Renault's variability model and its configuration system. We also introduce the system's parameters and their usage. Finally, we outline the objectives and analyze the challenges associated with the tuning process.

A. Variability model: Definitions

A variability model represents variables (also known as features) and their options, along with the relationships and dependencies between them. Formally, the VM can be defined as follows [18]:

A Variability Model is a triple (V, O, C), where V is a set of variables, O is a set of options for the variables, and C is a set of constraints. Each variable $v \in V$ is associated with a domain $Domain(v) \in O$. A *constraint* $c \in C$ can be intentional or extensional:

- *Extension constraint:* Also known as a table constraint, it is defined by enumerating a list of allowed or forbidden value tuples. It has the form $extension(V, S)$, where $V = \langle v_1, .., v_n \rangle$ and S is a set of supported/forbidden value tuples, $S = \langle \langle d_1, ..., d_n \rangle, ... \rangle$ (with $d_i \in Domain(v_i)$).
- *Intension constraint:* It is a constraint of the form $intension(V, P)$, where $V = \langle x_1, ..., x_n \rangle$ is a sequence of n variables (the scope of the constraint), and P is a predicate expression with n formal parameters on the variables of V.

A *literal* is a statement of the form v = d, where $d \in Domain(v)$. An *assignment* is a set of literals covering all the variables in V. A *partial assignment* is a set of literals covering a subset of V. A *solution* is an assignment consistent with all the constraints in C.

Here we give an example of a VM M:

- $Variables = \{model, fuel, airconditioning, dustfilter\}$
- $Options = \{\{m_1, m_2\}, \{petrol, diesel, lpg\}, \{manual, auto\}, \{with, none\}\}$
  with, $Domain(model) = \{m_1, m_2\}$,
  $Domain(fuel) = \{petrol, diesel, lpg\}$,
  $Domain(airconditioning) = \{manual, auto, none\}$
  and $Domain(dustfilter) = \{with, none\}$
- $Constraints$:

$$c_1 : extension(\langle model, fuel \rangle, \langle \langle m_1, lpg \rangle, \langle m_2, petrol \rangle, \langle m_2, diesel \rangle, \langle m_2, lpg \rangle \rangle)$$

$$c_2 : intension(\langle model, fuel, airconditioning \rangle, \langle (((model = m_1) \vee (model = m_2)) \wedge ((fuel = petrol) \vee (fuel = diesel))) \Rightarrow airconditioning = auto \rangle)$$

$$c_3 : intension(\langle airconditioning, dustfilter \rangle, \langle airconditioning = manual \Rightarrow dustfilter = with \rangle)$$

B. Variability model as undirected graphs

We introduce two undirected graphs that are used to encode a VM. These graphs are defined as follow:

- *variable-constraint graph*: each constraint is represented by a node; Each variable is represented by a node; An arc exists between a constraint node and a variable node if the variable is involved in the constraint.
- *variable graph*: each variable is represented by a node; An arc exists between two variable nodes if the two variables are involved in the same constraint.

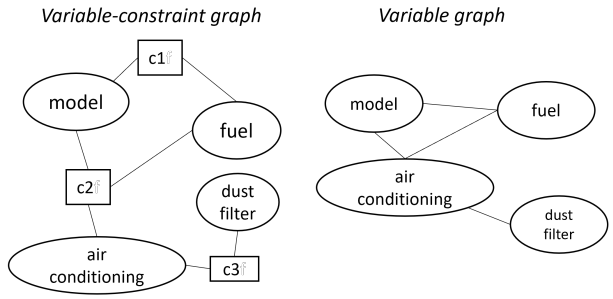

Fig. 1: *Variable-constraint graph* and *variable graph* of M.

C. The product configuration system of Renault: Overview & Challenges

Renault's product configuration system can be divided into two parts: the offline part generates and saves the configuration space, while the online part deals with different requests. The parameters are taken as inputs for the offline part and are then used to help control and make decisions during the compilation process. The online process searches for solutions in the compiled structure generated by the offline phase. Figure 2 presents the system architecture.

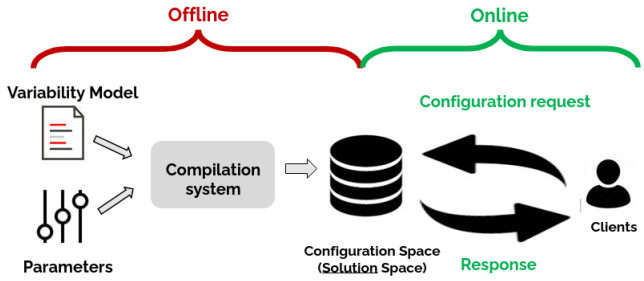

Fig. 2: Renault's product configuration system.

As mentioned before, the configuration space is represented in the form of a symbolic structure, which implicitly saves all the possible configurations. More specifically, it is based on a private compiled representation of vehicle diversity in the form of a cluster tree, which has been used in various applications at Renault since 1995 [19]. Here, we detail the offline process of the construction of the cluster tree (more details can be found in [19], [20]).

- **A cluster** is a group of variables associated with a set of partial solutions of a set of constraints. These constraints should only involve the variables in the cluster. The compilation process encodes literals as Boolean variables, where a literal represents a variable value assignment. Each partial solution is encoded as a vector of bits.
  For M, consider that $cluster_1$ refers to variables $model$ and $fuel$, and $cluster_2$ refers to variables $airconditioning$ and $dustfilter$. c_1 contains two variables, $model$ and $fuel$, so c_1 is associated with $cluster_1$. c_3 is associated with $cluster_2$. Boolean variables encode literals: a for $model = m_1$, b for $model = m_2$, c for $fuel = petrol$, d for $fuel = diesel$, and e for $fuel = lpg$. f, g, h encode the choices $manual$, $auto$, $none$ for $airconditioning$. i, j encode the choices $with$, $none$ for $dustfilter$. The partial solutions for each cluster, $cluster_1$ and $cluster_2$, are presented in Figure 3.

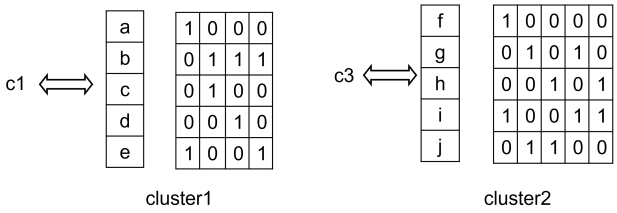

Fig. 3: $cluster_1$ and $cluster_2$.

- **A cluster tree** is a tree where each node represents a cluster. An arc between two clusters indicates a dependency between them in terms of constraints. These constraints involve variables from both clusters. The arc between the clusters contains a *Matrix* which evaluates whether the partial solutions within the linked clusters are consistent with the constraints. The *Matrix* enables the restoration of complete configurations from the partial solutions in each cluster. Figure 4 presents an example of a cluster tree and the *Matrix* is contained in the red arrow.

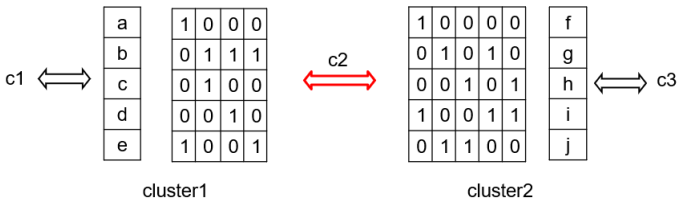

Fig. 4: Cluster tree example.

Multiple cluster trees can be derived for the same problem, depending on how the variables are arranged within clusters. To optimize the size of the cluster tree, the current system constructs it using heuristic analysis of the *variable-constraint graph* [21]. The system follows a process of *variable-constraint graph*-based partitioning to assign variables to clusters. Two important parameters are considered during this process to achieve an optimized cluster tree size:

- **PartitionVariables**: This parameter represents a list of variables in the variability model. The order of variables in the list is significant. The system uses these variables to partition the *variable-constraint graph* and construct the cluster tree. Each graph partition results in several subgraphs (or sub-cluster trees). In each sub-cluster tree, the partitioned variable is assigned a value from its domain to remove it. For example, in Figure 1, if we partition the graph using the node $airconditioning$, we obtain three sub-cluster trees as shown in Figure 5. Each sub-cluster tree consists of two clusters linked by an arc, with each

arc containing a matrix that represents the consistency information of partial solutions.

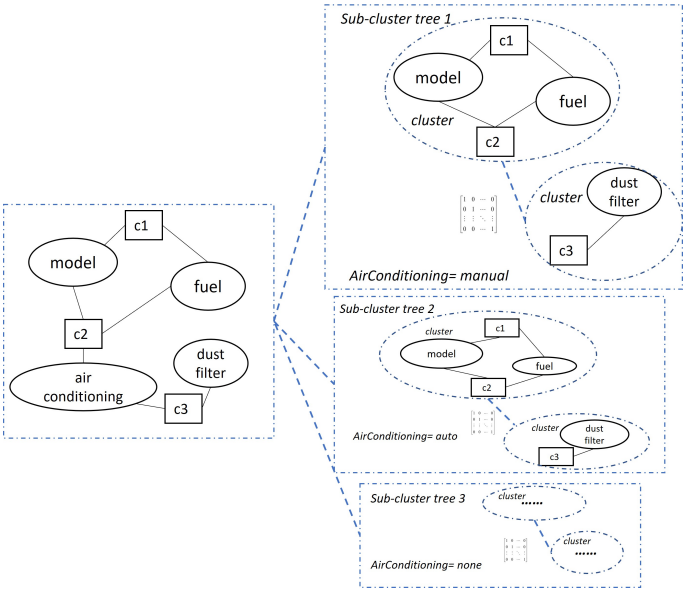

Fig. 5: Example of graph partition

- **MatrixSize**: This parameter represents the maximum size threshold of the matrices in the cluster tree. A matrix between clusters reflects the consistency of partial solutions within the linked clusters. The value of this parameter has a significant impact on the final size of the cluster tree. A large **MatrixSize** allows for variables to be dispatched with large matrices between clusters, resulting in a larger compiled structure. However, a small **MatrixSize** can lead to a smaller overall size of the compiled structure. Nevertheless, it also leads to more graph partitions. As a consequence, more sub-cluster trees are created in the compiled structure, which in turn increases the time required to iterate through all the sub-cluster trees when responding to requests.
  In summary, the choice of **MatrixSize** affects both the size of the compiled structure and the computational efficiency of processing requests. It is crucial to strike a balance between the size of the compiled structure and the runtime performance.

The primary objective of the tuning process is to propose a parameter prediction model that accurately predicts the appropriate values of **MatrixSize** and **PartitionVariables** for each input instance. This prediction model aims to reduce the size of the cluster tree compared to the default parameters. The next section presents the process of tuning and predicting these parameters.

IV. Parameters Tuning and Predicting for Renault's product configuration System

Our objective is to develop an automated prediction model that is capable of generating feature-dependent parameter predictions. In this section, we outline the process of training such a model and integrating it into the configuration system.

A. Parameter Prediction Model (PPM) Training

Training a parameter prediction model involves two main processes: parameter *tuning* and *learning*. The tuning process is used to search for the best-performing parameters for each instance, while the learning process focuses on training a machine learning model that maps the features of instances to the corresponding best-performing parameters.

1) Parameters Tuning Process: The tuning process is a prep-processing work to prepare data for training set. It aims to identify the best-performing parameters that optimize a given objective. It is important to note that the best-performing parameter setting we search for is not necessarily the theoretically best parameter setting, as that would be computationally expensive and challenging to determine. Instead, we aim to find the parameter setting that optimizes the target algorithm the most within a given computational resource and time limit, based on specific evaluation metrics. In our case, the evaluation metric is the compilation result size. Figure 6 illustrates the steps involved in this process.

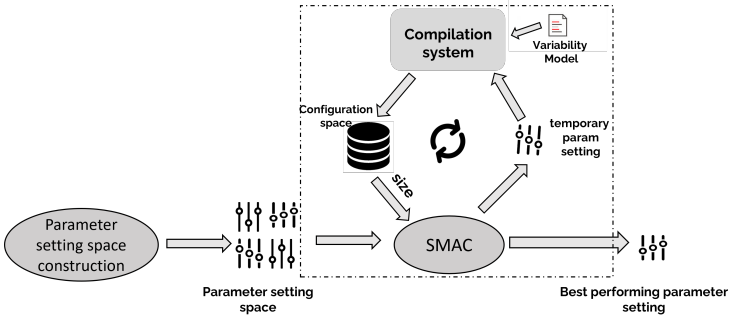

Fig. 6: Parameters tuning process

Parameter settings space construction. The initial step involves defining the parameter domains to create the parameter settings space. For the **MatrixSize** parameter, we have constrained the domain to integers ranging from 10 to 300. This choice is informed by practical experience, encompassing all suitable values for **MatrixSize**. Through observation, we have determined that values exceeding 300 result in a single, unpartitioned large sub-cluster tree, whereas values that are too small lead to excessive graph partitions. This, in turn, increases the response time, as previously mentioned.

To specify the domain for 'PartitionVariables,' we employ the concept of 'betweenness centrality' as introduced in [22]. Betweenness centrality is a measure of centrality in a graph based on shortest paths [23]. We create a 'variable graph' for each instance and compute the 'betweenness centrality' for each variable node. Subsequently, we rank the variables based on their centrality values and select the top five.

All subsets of this list are considered as potential domains for **PartitionVariables**, with different orders of the same variables treated as separate parameters. It's worth mentioning that the actual number of variables used in the daily parameters[2] ranges from 0 to 10. From extensive tests with multiple instances, we observed that the first five variables used in **PartitionVariables** often matter much more than the later variables over the compilation result. The alteration of the variables at the back position slightly change the compilation result size. So, to facilitate the experiments, we fix the number of used variables in **PartitionVariables** to five.

SMAC activity. Once the parameter domains are defined, we initiate the tuning phase to search for the best-performing parameter settings within the parameter settings space for each instance. To accomplish this, we employ SMAC, an automated algorithm configuration tool. As discussed in Section 2, SMAC optimizes the performance of an algorithm by executing it with different parameter settings. It employs various strategies, such as random forest [24] and Bayesian optimization [25], to guide the search process. Notably, SMAC excels in handling categorical parameters.

We configure SMAC to run a maximum of 60 iterations for each instance. The choice of 60 iterations is based on extensive testing of multiple instances, where we observed that the compilation size does not significantly decrease beyond approximately 50 iterations. The target for optimization is the compilation size. We record the best-performing parameter settings found by SMAC and also measure the time consumed during this process.

2) Parameters Learning Process: Before delving into the learning process, it is important to address the issue of choosing an appropriate machine learning model for the two parameters. Predicting **MatrixSize** is a classical regression task that involves mapping problem features to the optimal **MatrixSize** for each instance. On the other hand, predicting **PartitionVariables** is a classification task rather than regression. The model for predicting **PartitionVariables** should be capable of selecting variables from the instance and determining their position in the **PartitionVariables** list. To tackle this issue, we decided to train two separate models: one regression model for predicting **MatrixSize**, and one classification model for **PartitionVariables**. The choice of input instance features, models, and training processes for each model will be explained separately in this section. Figure 7 illustrates this process.

Graph encoding activity. The first step involves encoding the VM as a *variable-constraint graph* and a *variable graph* to facilitate feature extraction.

Features extraction activity.

- *Features for learning **MatrixSize**:* To train the model for predicting **MatrixSize**, we draw inspiration from a similar study called SATZILLA [26], which extracts 48 features to analyze SAT instances by encoding them as *variable graphs* and *variable-constraint graphs*. In our case, we also rely on graph-based features and select 14 specific features:
  - *Problem size features:*
    - Number of variables in the VM
    - Number of constraints in the VM
  - *Variable-constraint graph features:*
    - Minimum, average, and maximum *degree* values[3] of constraint nodes
    - Minimum, average, and maximum *degree* values of variable nodes
    - Minimum, average, and maximum *betweenness centrality* values of constraint nodes
    - Minimum, average, and maximum *betweenness centrality* values of variable nodes

  Since the construction of the cluster tree is based on the partitioning of the *variable-constraint graph*, it is reasonable to select features from such graphs as inputs for the learning process.

- *Features for learning **PartitionVariables**:* **PartitionVariables** is a parameter that is related to each variable of the instance. In order to build a model for the classification task, we need to extract features for each variable in the instance. For each variable, we calculate its node degree and betweenness centrality value separately in the *variable-constraint graph* and the *variable graph*. We use a four-bit vector to represent these pieces of information as features for each variable.

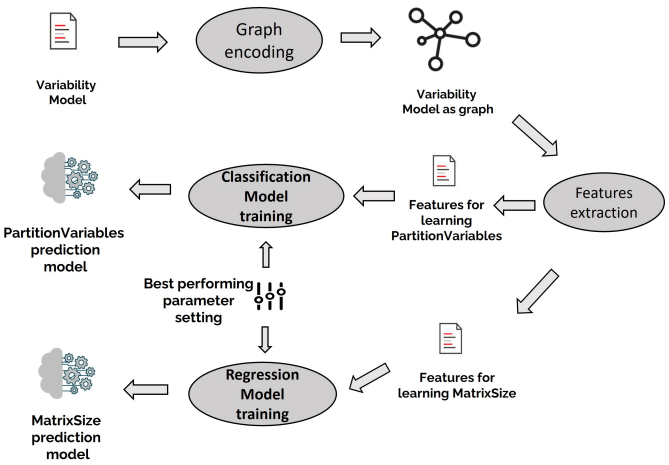

Fig. 7: *Parameters learning process*

Regression Model Training activity: With the target

[2]The daily parameter is also called the production parameter, which is manually determined and updated by the system developers

[3]The degree of a vertex in an undirected graph is the number of edges incident with (meeting at or ending at) itself. [27]

MatrixSize and available features for each instance, we proceed to the learning phase to train the regression model. Various standard machine learning models, such as Linear Regression [28], Support Vector Machines (SVM) [29], Gaussian Regression [30], etc., can be utilized. After careful evaluation, we select **Random Forest Regression** as our choice. This decision is based on its ability to provide reasonable predictions without requiring extensive hyper-parameter tuning. Furthermore, it effectively addresses the problem of overfitting that can occur with decision trees [24].

Classification Model Training activity: The process of selecting a variable partition for the compilation process involves evaluating the suitability of all variables in the instance. To accomplish this, we develop a scoring model based on the **SVM classification model [29]**. This model aims to assign scores to all variables in an instance, with a higher score indicating a more favorable position in the **PartitionVariables** list. To train this model, we begin by labeling the variables in the **PartitionVariables** list identified by SMAC. Each variable is assigned a score ranging from 0 to 1, with the leading variable receiving the highest score and the last obtaining the lowest score. Variables not included in the list are assigned a score of 0. Subsequently, we train a model to learn the relationship between the features of these variables and their corresponding scores. This trained model is capable of scoring variables for unseen instances. By ordering the variables based on their scores, we can dynamically define the number of required variables for each instance. The resulting ordered list serves as the predicted **PartitionVariables**.

B. Integrating the Prediction Models into Renault's Product Configuration System

Now that we have outlined all the necessary steps for the tuning and learning process, we proceed to integrate the prediction models into the configuration system. Figure 8 illustrates the enhanced system with the inclusion of the prediction models. When a VM is processed, it is initially passed through the "graph encoding and features extraction" activity to extract its features. This step is relatively quick and does not significantly impact the overall compilation time. Subsequently, the extracted features are fed into the prediction models, which generate the predicted values for **MatrixSize** and **PartitionVariables**. By applying these predicted parameters to the configuration system, we obtain a new data structure representing the configuration space. Figure 8 gives a global overview of the newly developed system.

In the next section, we will evaluate the performance of the parameters discovered by SMAC. Additionally, we will conduct separate and combined tests on the predicted **MatrixSize** and **PartitionVariables** to assess their effectiveness.

V. Experiments

In this section, we present the results of the tuning and prediction processes described earlier. Firstly, we obtain the best-performing parameters using SMAC. Then, with the parameter settings discovered by SMAC, we train two separate models to predict **MatrixSize** and **PartitionVariables**. The compilation results for different parameter configurations and the response time of the requests are analyzed. All experiments are conducted on the same execution environment: a CPU i7 with 3.00GHz and 32GB RAM.

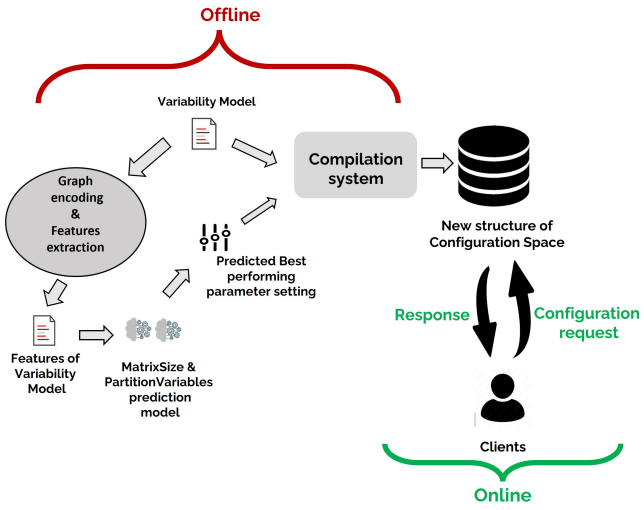

Fig. 8: *Integration of the parameters prediction model into the configuration system*

A. Datasets

We collected a dataset consisting of 600 instances, each representing a variable model of Renault. These instances are divided into a training set containing 200 instances and a validation set containing 400 instances.

To evaluate the influence of parameter tuning, we classify the instances into three classes based on a custom indicator provided by Renault. This indicator is associated with the cluster tree compilation process discussed in Section III-C and approximately measures the solving difficulty of each instance within the configuration system. It is calculated by analyzing the *variable-constraint graph* and determining the number of cycles present. Specifically, we compute the number of connected components (Ncc) in the graph and utilize Tarjan's cycle enumeration algorithm [31] to calculate the number of cycles (Nci) for each connected component. Additionally, we determine the number of nodes ($Nnode$) in each connected component. Finally, we use the sum of the product of Nci and $Nnode$ ($\sigma = \sum_{i=1}^{Ncc}(Nci_i \times Nnode_i)$) as an indicator for classifying the models. It's worth mentioning that we chose to use the number of cycles to calculate this indicator because the cycle is more difficult to deal with during graph partitioning. It takes more partitioning for the cycle parts in the graph to obtain the separate sub-cluster trees. Generally, we detect more cycles in the graph of large instances compared to the small instances, which proves the reliability of this indicator.

We order the 600 instances based on the σ value and assign them to three classes, each consisting of 200 instances. Within

each class, one-third of the instances are allocated to the training set, while the remaining two-thirds form the validation set. The details of the dataset subdivision are presented in Table I. The column $\#VM$ indicates the number of VMs in each class, while $Avg(\sigma)$ represents the average σ value for each class. Furthermore, $Min(\sigma)$ (resp. $Max(\sigma)$) denotes the minimum (resp. maximum) σ value within each class.

TABLE I: Subdivision of 600 instances in three classes using the calculated σ.

Class	#VM	$Avg(\sigma)$	$Min(\sigma)$	$Max(\sigma)$
C_1	200	210,517,750	1,593,181	2,462,269,412
C_2	200	138,079	235	1,593,181
C_3	200	61	0	235

B. Results

In this section, we present the results of the parameter tuning and prediction processes. We begin by introducing the **Production parameters**, which are currently used by Renault's configuration system as a reference. Then, we compare the performance of the parameters found by SMAC with the production parameters.

Production parameters. The production parameters are manually determined and updated by the system developers. They are obtained through an analysis of the input VM's *variable-constraints graph* and extensive simulation tests. All the experimental results are compared to the compilation results achieved using the production parameters.

Parameters found by SMAC. In this part, we evaluate the performance of the parameters discovered by SMAC on the *training set* instances, in comparison with the production parameters. The results are presented for each class. We calculate the sum of the instances' compilation sizes for each class and present the comparison in Table II. The column $size_{pp}$ represents the compilation size with the production parameters, while $size_{smac}$ represents the compilation size with the parameters found by SMAC. The column Δ indicates the percentage difference between the two sizes ($\Delta = (size_{pp} - size_{smac})/size_{pp}$). Additionally, the time taken by SMAC to find these parameters is presented in Table III.

TABLE II: Compilation result comparison between production parameters and parameters found by SMAC.(Unit:MB)

Class	#VM	$size_{pp}$	$size_{smac}$	Δ
C_1	67	44,948	22,030	51%
C_2	67	5,286	1,682	68%
C_3	67	17.31	17.19	0.12%

We observe that for all classes of instances, the parameters found by SMAC outperform the production parameters. In the case of the complex classes C_1 and C_2, the reduction in compilation size is up to 68%. However, it is worth noting that SMAC requires a significant amount of time to find these high-performance parameters.

TABLE III: Time usage of SMAC.(Unit:hour)

Class	#VM	time
C_1	67	68.73
C_2	67	2.61
C_3	67	0.17

MatrixSize Prediction Model. This experiment evaluates the performance of the model for predicting **MatrixSize**. For the instances in the *validation set*, we compile them using the **PartitionVariables** of the production parameters along with the predicted **MatrixSize**. The compilation results are presented in Table IV, where $size_{pp}$ represents the total compilation size of the class with production parameters, $size_{matrixmodel}$ represents the compilation size with the predicted **MatrixSize**, and Δ indicates the difference ($\Delta = (size_{pp} - size_{matrixmodel})/size_{pp}$).

TABLE IV: Compilation result comparison between production parameters and parameters predicted by MatrixSize prediction Model. (Unit:MB)

Class	#VM	$size_{pp}$	$size_{matrixmodel}$	Δ
C_1	133	155,845	101,030	35%
C_2	133	5,441	4,806	11.6%
C_3	133	54.13	64.91	-19.9%

From Table IV, we observe that the **MatrixSize** prediction model performs well for the complex classes C_1 and C_2. The parameters with the predicted **MatrixSize** can reduce the total compilation size by up to 35%. However, for the simple class C_3, it results in an increase in the compilation size. Upon further investigation, we found that for most instances in C_3, the compilation size remains the same, except for one instance where the compilation size increases.

PartitionVariables Prediction Model. This experiment evaluates the performance of the model for predicting **PartitionVariables**. For the instances in the *validation set*, we compile them using the **MatrixSize** of the production parameters and the predicted **PartitionVariables**. We trained a scoring model to assist in selecting the partition variables. Specifically, for each instance, we rank the variables based on the model's score and choose the top 5 variables as the **PartitionVariables**. The compilation results are presented in Table V, where $size_{varmodel}$ represents the total compilation size with the predicted parameters.

From Table V, we observe that for all the classes, the predicted parameters perform the same or worse compared to the production parameters. Specifically, for the complex class C_1, it results in a twofold increase in the compilation size. We suspect that this is due to the lack of controlled **MatrixSize**.

TABLE V: Compilation result comparison between production parameters and parameters predicted by PartitionVariables Prediction Model (Unit:MB)

Class	#VM	$size_{pp}$	$size_{varmodel}$	Δ
C_1	133	155,845	313,958	-101.45%
C_2	133	5,441	5,609	-2.9%
C_3	133	54.13	54.13	0.00%

Apparently, the **MatrixSize** of the production parameters is not suitable for the predicted **PartitionVariables**. As mentioned in Section III-C, improper variable dispatch without appropriate **MatrixSize** control can lead to a large cluster tree. The results of this experiment validate this observation, especially for the complex instances. Additionally, we observe that the compilation size remains unchanged for the 133 instances of the simple class C_3 with the predicted parameters.

Combination of MatrixSize and PartitionVariables Prediction Models. This experiment combines the parameters predicted by the MatrixSize and PartitionVariables models. For the instances in the *validation set*, we compile them using the predicted **MatrixSize** and **PartitionVariables** parameters. Table VI presents the compilation results, comparing them with the production parameters, where $size_{both}$ represents the total compilation size with the combined predicted parameters by the two models above.

TABLE VI: Compilation result comparison between production parameters and parameters predicted sequentially by Models for PartitionVariables and for MatrixSize.(Unit:MB)

Class	#VM	$size_{pp}$	$size_{both}$	Δ
C_1	133	155,845	216,827	-39.12%
C_2	133	5,441	5,022	7.7%
C_3	133	54.13	64.91	-19.9%

From Table VI, we observe that for the most complex class C_1, the compilation size still increases with the predicted parameters. However, the increment is smaller compared to the results in Table V. The performance of the predicted parameters improves when the predicted **MatrixSize** is considered.

For class C_2, the predicted parameters perform positively, reducing the compilation size by 7.7%. In the case of C_3, the increment is the same as in Table IV, indicating that it is caused by the same instance. The compilation size remains unchanged for all the other instances when using the predicted parameters compared to the production parameters.

Response Time for Requests. The online part of the compilation system is responsible for responding to requests with the compiled structure. In this section, we compare the response times of requests using different compilation results obtained with the parameters mentioned above.

We begin by specifying the type of request used for this test. For each instance, we generate 2000 requests for satisfiability checks [32]. These requests consist of partial configurations where certain variables are assigned specific values to check their satisfiability. Out of these 2000 requests, 1000 are satisfiable and 1000 are not. This choice is motivated by the actual number and type of requests used by Renault to verify the consistency of their product offer [18].

Next, for each instance in the validation set, we compile it separately using the production parameters, parameters predicted by the **MatrixSize** model, parameters predicted by the **PartitionVariables** model, and parameters predicted by both models. We measure the response times of the 2000 requests for each compiled structure. The results are presented in Table VII. RT_{pp} represents the total response time using the compiled structure with production parameters. $RT_{matrixmodel}$ represents the time using the predicted **MatrixSize** parameters. $RT_{varmodel}$ represents the time using the predicted **PartitionVariables** parameters. RT_{both} represents the time using parameters predicted by both models.

TABLE VII: Comparison of response times between compilation results with production parameters and predicted parameters (Unit: Seconds)

Class	RT_{pp}	$RT_{matrixmodel}$	$RT_{varmodel}$	RT_{both}
C_1	694	1133	2030	3349
C_2	22	22	28	27
C_3	3	3	3	3

From Table VII, we observe that for all compilations using predicted parameters, the response time is longer compared to using the production parameters. There are two reasons:

- For compilations using predicted **PartitionVariables** parameters, the size of the compiled structures increases (Table V and Table VI), which reasonably leads to longer response times.
- For compilations using predicted **MatrixSize** parameters and default **PartitionVariables** parameters, although the compiled size is smaller than the default parameters (Table IV), the response time still increases. This is because more sub-cluster trees are generated during compilation with predicted MatrixSize values. As mentioned in Section III-C, a smaller **MatrixSize** can result in more sub-cluster trees, reducing the size of the global structure but increasing the response time for requests. When we verify the predicted **MatrixSize** values, we confirm that they are generally smaller than the **MatrixSize** of the production parameters.

In fact, the remarkable reduction in size achieved by the predicted **MatrixSize** outweighs the slight increase in response time, making it a highly favorable trade-off. It adds only 3.3 seconds ((1133-694)/133) to the response time per instance when handling 2000 requests, resulting in a mere 1.6ms (3.3/2000) longer response time per request. Considering the substantial benefits gained from the reduced size, this minor

impact on response time can be deemed negligible.

C. Summary

Based on the five experiments conducted, we draw the following conclusions:

- SMAC demonstrates excellent performance in searching for the best parameters for each instance, resulting in a reduction of up to 68% in compilation size. However, the drawback is its time-consuming nature.
- The **MatrixSize** prediction model proves to be successful, achieving a significant reduction of up to 35% in compilation size for the complex class C_1. The trade-off is a mere 1.6ms increase in response time per request. The operational impact of this reduction is substantial.
- For simple instances, applying the predicted **MatrixSize** and **PartitionVariables** separately or together has minimal impact on the compilation result. In most cases, the compilation size remains unchanged, with only a few instances experiencing a slight increase.
- The automated tuning and prediction of **PartitionVariables** require further exploration. The related experiments yielded negative results thus far.

Overall, the parameters found by SMAC perform the best among the parameters above, but with the drawback of time-consuming. After discussing with the stakeholders, we discovered the compilation process can happen frequently. So, applying SMAC each time before the compilation process is impractical. On the other hand, with the **MatrixSize** model obtained, we gain a 35% reduction in the compilation size with the parameters predicted within seconds. So considering both the compilation size reduction and the rapid parameters prediction, integrating the **MatrixSize** model into the product configuration system of Renault seems promising.

To facilitate the replication of the experiment process, we have provided a file accessible at: https://github.com/chiyanfly/Automated-Parameters-Tuning. The file includes:

- Source code for building the "variable-constraint graph" and "variable graph" and calculating "betweenness centrality".
- Source code demonstrating the usage of SMAC.
- Examples of **MatrixSize** and **PartitionVariables** prediction.

It is important to note that the verification of the experiment results related to the configuration system is confidential, as it is an internal proprietary system of Renault.

VI. Conclusion

This paper presents an automated parameters tuning and predicting process to optimise Renault's product configuration system. The aim is to reduce the variability model's compilation size. We have shown the strong competitiveness of the **MatrixSize** parameter predicted by our model, and the reduction of compilation size is up to 35%.

Our first perspective is to continue exploiting the tuning process for **PartitionVariables**, since the current experiments show the negative results with its prediction model. We also intend to enlarge the data sets to improve the performance of models. In addition, since the **MatrixSize** prediction model has brought very promising results, this model is expected to be put into real usage for the operational system of Renault.

References

[1] T. B. et al., "A survey of variability modeling in industrial practice," in The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa , Italy, January 23 - 25, 2013, S. Gnesi, P. Collet, and K. Schmid, Eds. ACM, 2013, pp. 7:1–7:8.

[2] H. Xu, S. Baarir, T. Ziadi, S. Essodaigui, Y. Bossu, and L. Messan Hillah, "Optimization of the product configuration system of renault," in Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, 2023, pp. 1486–1489.

[3] A. Darwiche and P. Marquis, "A knowledge compilation map," Journal of Artificial Intelligence Research, vol. 17, pp. 229–264, 2002.

[4] S. B. Akers, "Binary decision diagrams," IEEE Transactions on computers, vol. 27, no. 06, pp. 509–516, 1978.

[5] A. Choi and A. Darwiche, "Dynamic minimization of sentential decision diagrams," in Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA, M. desJardins and M. L. Littman, Eds. AAAI Press, 2013.

[6] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Feature based algorithm configuration: A case study with differential evolution," in Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings 14. Springer, 2016, pp. 156–166.

[7] C. Ansótegui, M. Sellmann, and K. Tierney, "A gender-based genetic algorithm for the automatic configuration of algorithms," in Principles and Practice of Constraint Programming-CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings 15. Springer, 2009, pp. 142–157.

[8] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," Journal of Artificial Intelligence Research, vol. 36, pp. 267–306, 2009.

[9] T. Bartz-Beielstein, O. Flasch, P. Koch, W. Konen et al., "Spot: A toolbox for interactive and automatic tuning in the r environment," in Proceedings, vol. 20, 2010, pp. 264–273.

[10] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," Operations Research Perspectives, vol. 3, pp. 43–58, 2016.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Automated configuration of mixed integer programming solvers," in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings 7. Springer, 2010, pp. 186–202.

[12] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. Hoos, and K. Leyton-Brown, "The configurable sat solver challenge (cssc)," Artificial Intelligence, vol. 243, pp. 1–25, 2017.

[13] M. El Yafrani, M. Scoczynski, I. Sung, M. Wagner, C. Doerr, and P. Nielsen, "Mate: A model-based algorithm tuning engine: A proof of concept towards transparent feature-dependent parameter tuning using symbolic regression," in Evolutionary Computation in Combinatorial Optimization: 21st European Conference, EvoCOP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 21. Springer, 2021, pp. 51–67.

[14] B. Doerr, H. P. Le, R. Makhmara, and T. D. Nguyen, "Fast genetic algorithms," in Proceedings of the genetic and evolutionary computation conference, 2017, pp. 777–784.

[15] C. Witt, "Tight bounds on the optimization time of a randomized search heuristic on linear functions," Combinatorics, Probability and Computing, vol. 22, no. 2, pp. 294–318, 2013.

[16] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5. Springer, 2011, pp. 507–523.

[17] N. Belkhir, J. Dréo, P. Savéant, and M. Schoenauer, "Per instance algorithm configuration of cma-es with limited budget," in Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 681–688.

[18] H. Xu, S. Baarir, T. Ziadi, S. Essodaigui, Y. Bossu, and L. Messan Hillah, "An experience report on the optimization of the product configuration system of renault," in 2023 27th International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE Computer Society, 2023, p. 197–206.

[19] B. Pargamin, "Vehicle sales configuration: the cluster tree approach," in ECAI 2002 Configuration Workshop, 2002, pp. 35–40.

[20] P. Bernard, "Extending cluster tree compilation with non-boolean variables in product configuration: A tractable approach to preference-based configuration," in Proceedings of the IJCAI, vol. 3. Citeseer, 2003.

[21] A. Becker and D. Geiger, "A sufficiently fast algorithm for finding close to optimal clique trees," Artificial Intelligence, no. 1-2, pp. 3–17, 2001.

[22] M. Barthelemy, "Betweenness centrality in large complex networks," The European physical journal B, vol. 38, no. 2, pp. 163–168, 2004.

[23] U. Brandes, "A faster algorithm for betweenness centrality," Journal of mathematical sociology, vol. 25, no. 2, pp. 163–177, 2001.

[24] V. Rodriguez-Galiano, M. Sanchez-Castillo, M. Chica-Olmo, and M. Chica-Rivas, "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines," Ore Geology Reviews, vol. 71, pp. 804–818, 2015.

[25] P. I. Frazier, "Bayesian optimization," in Recent advances in optimization and modeling of contemporary problems. Informs, 2018, pp. 255–278.

[26] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: portfolio-based algorithm selection for sat," Journal of artificial intelligence research, vol. 32, pp. 565–606, 2008.

[27] A. Grami, "Chapter 18 - graphs," in Discrete Mathematics, A. Grami, Ed. Academic Press, 2023, pp. 327–350. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128206560000186

[28] S. Weisberg, Applied linear regression. John Wiley & Sons, 2005, vol. 528.

[29] W. S. Noble, "What is a support vector machine?" Nature biotechnology, vol. 24, no. 12, pp. 1565–1567, 2006.

[30] H. Zhu, C. K. Williams, R. Rohwer, and M. Morciniec, "Gaussian regression and optimal finite dimensional linear models," 1997.

[31] R. Tarjan, "Enumeration of the elementary circuits of a directed graph," SIAM Journal on Computing, vol. 2, no. 3, pp. 211–216, 1973.

[32] R. Pohl, K. Lauenroth, and K. Pohl, "A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models," in 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE, 2011, pp. 313–322.