Optimisation for the Product Configuration System of Renault: Towards an Integration of Symmetries

Hao Xu
LIP6
Renault
France
hao.xu@lip6.fr

Souheib Baarir
LIP6
Université Paris Nanterre
France
souheib.baarir@lip6.fr

Tewfik Ziadi
LIP6
Sorbonne Université
France
tewfik.ziadi@lip6.fr

Lom Messan Hillah
LIP6
Université Paris Nanterre
France
lom-messan.hillah@lip6.fr

Siham Essodaigui
Renault
France
siham.essodaigui@renault.com

Yves Bossu
Renault
France
yves.bossu@renault.com

ABSTRACT

The problem of configuring model variability is widespread in many different domains. Renault, a leading french automobile manufacturer, has developed its technology internally to model vehicle diversity. This technology relies on the approach known as knowledge compilation. Since its inception, continuous progress has been made in the tool while monitoring the latest developments from the software field and academia. However, the growing number of vehicle models brings potential risks and higher requirements for the tool. This paper presents a short reminder of Renault's technology principles and the improvements we intend to achieve by analyzing and leveraging notable data features of Renault problem instances. In particular, the aim is to exploit symmetry properties.

CCS CONCEPTS

• **Software and its engineering** → *Software product lines*; • **Theory of computation** → **Automated reasoning**.

KEYWORDS

Knowledge compilation, Product line, SAT, Symmetries

ACM Reference Format:
Hao Xu, Souheib Baarir, Tewfik Ziadi, Lom Messan Hillah, Siham Essodaigui, and Yves Bossu. 2021. Optimisation for the Product Configuration System of Renault: Towards an Integration of Symmetries . In *25th ACM International Systems and Software Product Line Conference - Volume B (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3461002.3473948

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8470-4/21/09…$15.00
https://doi.org/10.1145/3461002.3473948

1 INTRODUCTION

Variability modeling problems (VMP)[4] are pervasive in real life. In the car industry, this problem is even crucial, as it touches all business activities. These range from the design (by engineers) to the manufacturing (by technical staff), passing by the vehicles' documentation and marketing forecasts.

As a simple example of such a VMP, consider the fuel type and the number of place for a car. Let us take as a definition domain for the fuel type the set {*petrol, diesel, LPG*} and the set {*4,5,7*} for the number of place. We end up by having nine possible combinations that form the different configurations for the chosen car. Then, we must impose between these features dependency relationships that describe activity-related constraints (business, technical, legal requirements, and many others) to complete the picture. Therefore, we can easily perceive here the complexity of real-life VMPs, and the hardness of their solving: some ranges of vehicles of Renault count around 10^{21} possible configurations [2]. In the following, we call such a VMP the *basic problem*.

Besides the problem mentioned above, an essential business requirement is to rapidly respond to customers' online vehicle configuration. For example, during online shopping on Renault's website, customers can choose the car model, the motor type or manipulate the other possible options provided online. Renault needs to respond to those customers' requests within less than a second with the satisfying cars. The choices the customer has made include a chain of specific values for each option. For example, *{model = Clio, color = Red, motor = Diesel}* is a set of choices the customer could have made during the online shopping. We call such a set a *configuration*.

From the theoretical point of view, the *basic problem*, as well as the *configuration* problem are typical Constraint Satisfaction Problems (CSP). Since the domains of all variables are discrete and finite, one can straightforwardly encode these CSPs as (Boolean) Satisfiability problems (SAT)[8].

If we focus on solving the online configuration problem, one can think of two possible solutions. The first, the natural one, is to solve a CSP/SAT problem at every request. As the user inputs a *configuration* online, it is taken as a new set of constraints and combined with the *basic problem* to form a new CSP/SAT problem. We then feed

the resulting problem to a CSP/SAT solver. This solution is simple but is costly in general. Indeed, one has to solve an NP-complete problem at each request [6]. If, however, the underlying problem falls into a polynomial category (e.g., 2-SAT [11], Horn-SAT [7]), or the *basic problem* is too easy to solve (less than a second), it can then be solved many times costless. When evaluating the benchmarks of Renault, it turns out that none of the above statements is true. In particular, testing state-of-the-art SAT solvers (e.g., kissat [5], glucose [3]) on the *basic problem*, without adding any *configuration* request, reveals that around 30 % of the instances are solved in more than 5 seconds to get only one solution. So this is not an option for our configuration problem.

The second possible solution is to build, once and for all, the *configurations space*, a solution known as *knowledge compilation* [13]. Hence, the customer's online *configuration* requests can be easily checked in polynomial time. Of course, building the *configurations space* is more difficult than finding just one solution. However, since this procedure can run offline and the configuration engine can reuse such a compiled knowledge for all requests, the approach seems to be a good compromise.

Actually, for its VMP system, the second solution is the one Renault has retained, implemented, and is being used for a long time [12]. Though, it is subject to a critical limit, the memory size. Indeed, the computed configurations reside in memory, and all requests are operated from there. Otherwise, the whole approach becomes meaningless. Of course, Renault can easily increase the machine memory facing the memory shortage, but the configuration space size could be exponential due to the problem complexity [13], so we can't always guarantee the memory is enough. So far, several heuristics and optimizations control the pressure on the memory. Here we want to go further and continue the improvement process of this solution by exploring new tracks.

Hence, we discuss in this paper the ins and outs of a possible direction to lighten the memory pressure when constructing the *configuration space*, namely by exploiting *symmetry properties*. When present, symmetries can be very helpful for compressing the underlying structure at hand and thus optimize memory usage.

The remainder of the paper is organized as follows. Section 2 briefly presents the configuration system of Renault and its underlying data structure. Section 3 introduces symmetries, shows how we have figured them out from a sample of Renault's problem instances and explains how we plan to apply them. Finally, section 4 hints at other possible directions to explore for solving the memory pressure issue, while section 5 ends this paper.

2 BACKGROUND ON THE CONFIGURATION SYSTEM OF RENAULT

We consider a vehicle (the product) as a set of variables following a system of constraints. The left-hand side of Figure 1 shows an example with a set of seven variables and their four constraints. Different Renault divisions create these constraints for manufacturing requirements, marketing requests, ecological aims, and many other purposes. The constraints reduce the product diversity range but may increase the complexity of its configuration.

2.1 The product modeling system

As explained by B. Pargamin [12], the internal configurator of Renault is entirely built on the standard compiled representation of Renault vehicle diversity in the form of a cluster tree. Three processes formalize the construction and the manipulation of the cluster tree:

- **Construction:** A cluster tree is a tree whose nodes are clusters of the variables (e.g., engine, color). To ensure the completeness of inference algorithms operating on the cluster tree, a variable shared by two nodes must be present in every node on the path between the two nodes.
- **Implementation:** This process first builds the set of all Boolean variables instantiations consistent with the constraints that fit into the cluster. Consistent instantiations are called logical models. For example, in the matrix on the right-hand side of Figure 1 each column encodes a logical model of the vehicle represented with the seven variables x1 to x7.

  The process associates to each Boolean variable x_i (hereafter *bvar* x_i) a Boolean vector (VBV) whose i^{th} position is set to TRUE if x_i is TRUE in the i^{th} model, FALSE otherwise. Each line in the matrix Figure 1 encodes a Boolean vector for each *bvar*. Each logical model thus represents a valid configuration for the vehicle. In Figure 1, the column *CSP solution model* encodes such a valid configuration.

  The process finally builds a cluster state vector (CSV), a Boolean vector whose i^{th} position is set to TRUE if the i^{th} model is consistent with the current assignment of Boolean values to the corresponding *bvars* of the cluster, FALSE otherwise. Figure 2 shows such as a CSV for the cluster presented in Figure 1, with the current assignment of values to variables $x1$, $x2$, and $x3$.
- **Inference:** During this process, whenever a cluster's state changes, it propagates the change to all the variables in the cluster, then to all other clusters of the tree, ensuring a deductively complete truth maintenance system.

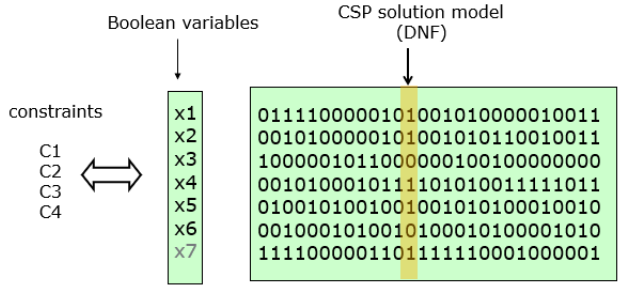

Figure 1: A cluster implementation

2.2 The vehicle sales configuration

During the configuration process, when the customer (indirectly) instantiates a Boolean variable x_i, the cluster state vector is re-evaluated by a Boolean AND operation between the prior CSV and the VBV or its negation.

The change then propagates its effect to all other *bvars* of the cluster that thus turn to TRUE (resp. FALSE) if their VBV (resp. the negation of their VBV) is compatible with the CSV. Figure 2 illustrates such an ongoing configuration process for the cluster of Figure 1, with the current assignment of values to variables $x1$, $x2$, and $x3$.

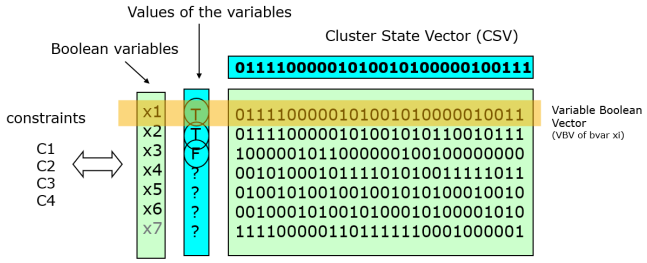

Figure 2: Configuration on a cluster of variables

Now that we have seen how a constraint propagates within a cluster, we can see the propagation between two clusters within the cluster tree, in Figure 3. This figure shows a compatibility matrix, called arc matrix, between the cluster AB and the cluster CD.

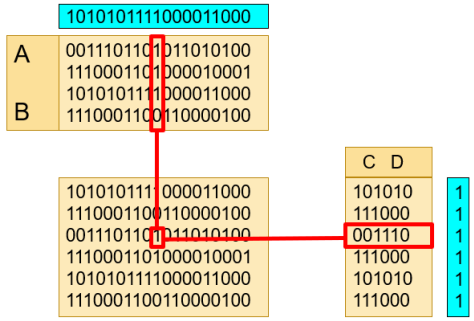

Figure 3: Arc between two clusters : propagation

The CSV of each cluster is updated according to this equation:

$$CSV\_cluster(C,D)(j) = \sum_i CSV\_cluster(A,B)[i] \times matrix\_arc[i,j]$$

Figure 4: formula of updating cluster

Figure 5 illustrates the result:

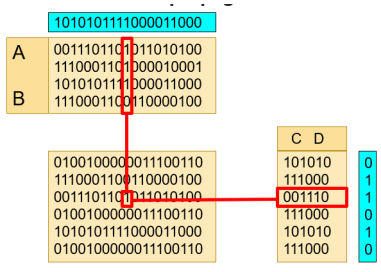

Figure 5: Propagation of choices between two clusters

2.3 Benefits and Challenges

The knowledge compilation, which consists of investing once in a heavy off-line compilation, provides the benefit of a very high-performing configurator in requests response time. Therefore, the challenge is to improve memory usage by the knowledge representation without degrading response time, neither for the compilation process nor for the configuration. For this challenge, optimising the representation structure by profiting the data structure feature could be a promising direction. The following section will present how the data feature *symmetries* could help achieve this challenge.

3 SYMMETRIES TO IMPROVE THE CONFIGURATION SYSTEM OF RENAULT

An idea to tackle the memory pressure problem that we face when using *knowledge compilation* approach could be to exploit compression techniques. These allow the reduction of the size of the underlying structure while preserving the targeted properties. An example of such techniques is compressing using *symmetries*. In this case, the *configurations space* is subdivided in parts, called *equivalence classes*. Each class groups those configurations obtained from each other just by permuting the identity of the objects. Hence, to represent the whole configuration space, one has to keep a representative from each class, thus drastically reducing memory usage.

The relevance and effectiveness of a symmetry-based approach depend on several requirements. The most critical ones are the following:

- **Requirement 1:** the original problem must exhibit a certain degree of symmetries;
- **Requirement 2:** the structure used for representing the *configuration space* must be formally well defined and suitable for exploiting symmetries;
- **Requirement 3:** the operations that one has to perform on the reduced structure must remain at the same complexity level as the one on the original structure.

3.1 Symmetries in Renault's variability modeling problems

The notion of symmetry relies on the concept of permutation. A permutation is a bijection from a set X to itself. For example, given a set $X = \{x1, x2, x3, x4\}$, $g = (x1, x2)(x3, x4)$ is a permutation that maps $x1$ to $x2$, $x3$ to $x4$ (and vice-versa).

In practice, computing the symmetries of a CSP/SAT problem boils down to the problem of graph isomorphisms [1]. Indeed, the problem at hand is encoded as a colored graph from which we extract the set of *automorphisms* using classical tools (e.g., saucy3 [10], bliss [9]). The computed set of *automorphisms* represents the set of permutations of our problem, called *generators*. We call symmetries of the problem [14] the group constructed on top of these *generators*.

Hence, to assess the first requirements mentioned above, we first created a benchmark with around 2500 instances of Renault's VMPs. We then computed the symmetries of these instances using `bliss` [9]. Table 1 summarizes the obtained results.

number of generators	percentage of instances
1-30	57.33%
31-60	23.71%
61-90	13.06%
91-120	4.60%
121-150	0.69%
151-180	0.61%

Table 1: Symmetry measurement results

For instance, we observe that 57.33% of the instances exhibit between 1 and 30 generators. Overall, 100% of the instances exhibit symmetries, which is the significant observation here. Therefore, we can safely conclude that this outcome completely fulfills our first requirement. Such a first positive result motivates us to go further and investigate the two remaining requirements.

3.2 Towards an integration of symmetries

As mentioned above, to achieve a complete integration of symmetries in Renault's variability modeling problems, the second requirement concerns the need to formally define the structure used for representing the *configuration space*.

As this requirement has never been considered before within Renault, and the whole of Renault's configuration system is already operational for several years, the difficulty resides in accurately reproducing this system, i.e., preserving completeness and correctness while defining the formal viewpoint.

As soon as the second requirement is fulfilled, it will be possible to address the third requirement and derive the theoretical complexities of different operations (e.g., the configuration operation). At first sight, all needed operations remain in polynomial time, but this has to be confirmed by a theoretical study.

Once all the requirements are satisfied, it will be time to integrate our new symmetries-based approach in Renault's product configuration eco-system. We plan to do that by injecting three independent modules, as illustrated in Figure 7.

- **Symmetry detection:** it takes the CSP/SAT instance as input, detects the symmetries, and outputs the symmetries along with the original problem.
- **Symmetry application:** it takes the output data of the Symmetry detection module, combines the symmetries with the data structure used during the compilation, and outputs a *configuration space* constructed with clusters of *equivalent classes*. Each class contains only one configuration that represents all the equivalent configurations of the class.
- **Solutions restoration:** this module restores the original configurations compressed by the symmetry application module. We need this module to restore the original solution for responding to customers' online configuration requests.

Figure 6 shows the current workflow of Renault's configuration system, while Figure 7 shows the resulting workflow after the integration of the planned symmetry application modules. Unlike the detection and application modules that are part of the compilation process, the restoration module is part of the configuration phase when the customer interacts with Renault's configurator.

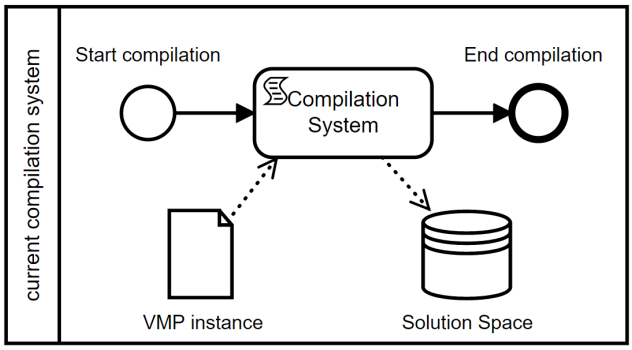

Figure 6: Current compilation system of Renault

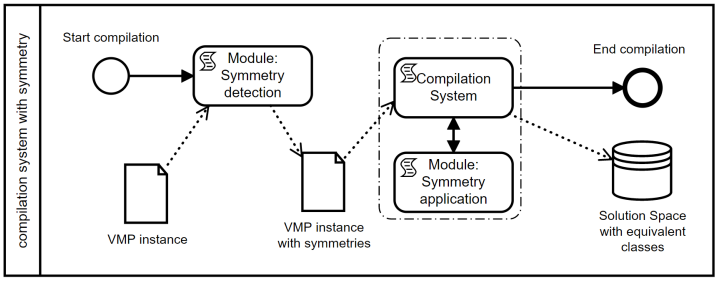

Figure 7: Integration of symmetry detection and application modules in Renault's compilation system

Another challenge to acknowledge is that the symmetry application module may increase the complexity of the configuration process. Indeed, such an approach may influence the time and memory requirements of the compilation. However, we cannot rule out these possible factors before carrying out a theoretical study with the formalization of the data structure. After all, very possibly, applying symmetry over the current data structure without increasing the algorithm complexity will become an important issue to handle.

4 OTHER DIRECTIONS

The current Renault compilation system always needs to take multiple parameters to control, adjust, and monitor the overall process. These parameters are directly related to the operations of the data structure. Therefore, they can directly influence memory usage during the compilation process. In addition to applying symmetries, another possible improvement is an automatic determination of computation parameters during the compilation of the original CSP problem to the *configuration space*. The action of decision for parameters can be static or dynamic:

- **Static determination:** Set the execution parameters at the beginning of the compilation process according to the input instance features (e.g., number of variables, number of constraints.);
- **Dynamic determination:** Set the parameters at the beginning and continuously adjusting them dynamically during the compilation process according to memory usage or some

other technical parameters, which can help further adjust the execution parameters in real-time.

This work should be suitable for machine learning techniques considering multiple parameters to configure and internal feedbacks within Renault.

5 CONCLUSION

This paper presents the background of Renault's model variability problem over the vehicle product line and its continuous improvement issues. We mainly focus on the challenge of the memory pressure for the configuration system. After a short reminder of Renault's technology, we introduce our method for this challenge: improvements based on the existing data structures using symmetries. We also discuss some other promising directions, such as compilation parameters tuning. These directions should have promising results based on our current progress.

REFERENCES

[1] Vikraman Arvind and Johannes Köbler. 2000. Graph Isomorphism Is Low for ZPP(NP) and Other Lowness Results. In Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS '00). Springer-Verlag, Berlin, Heidelberg, 431–442.

[2] Jean-Marc Astesana, Laurent Cosserat, and Hélène Fargier. 2010. Constraint-based Vehicle Configuration: A Case Study. In ICTAI (1). IEEE Computer Society, 68–75.

[3] Gilles Audemard and Laurent Simon. 2009. Glucose: a solver that predicts learnt clauses quality. SAT Competition (2009), 7–8.

[4] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. 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, Stefania Gnesi, Philippe Collet, and Klaus Schmid (Eds.). ACM, 7:1–7:8. https://doi.org/10.1145/2430502.2430513

[5] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions (Department of Computer Science Report Series B, Vol. B-2020-1), Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda (Eds.). University of Helsinki, 51–53.

[6] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing (Shaker Heights, Ohio, USA) (STOC '71). Association for Computing Machinery, New York, NY, USA, 151–158. https://doi.org/10.1145/800157.805047

[7] William F. Dowling and Jean H. Gallier. 1984. Linear-time algorithms for testing the satisfiability of propositional horn formulae. The Journal of Logic Programming 1, 3 (1984), 267–284. https://doi.org/10.1016/0743-1066(84)90014-1

[8] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. 1996. Algorithms for the satisfiability (SAT) problem: A survey. In Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11-13, 1996 (DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35), Ding-Zhu Du, Jun Gu, and Panos M. Pardalos (Eds.). DIMACS/AMS, 19–151. https://doi.org/10.1090/dimacs/035/02

[9] Tommi Junttila and Petteri Kaski. 2007. Engineering an efficient canonical labeling tool for large and sparse graphs. In 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX). SIAM, 135–149.

[10] Hadi Katebi, Karem A Sakallah, and Igor L Markov. 2010. Symmetry and satisfiability: An update. In International Conference on Theory and Applications of Satisfiability Testing. Springer, 113–127.

[11] M. R. Krom. 1970. The Decision Problem for Formulas in Prenex Conjunctive Normal Form with Binary Disjunctions. Journal of Symbolic Logic 35, 2 (1970), 210–216. https://doi.org/10.2307/2270511

[12] Bernard Pargamin. 2002. Vehicle sales configuration: the cluster tree approach. In ECAI 2002 Configuration Workshop. 35–40.

[13] Bernard Pargamin. 2003. 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.

[14] Karem A Sakallah. 2009. Symmetry and Satisfiability. Handbook of Satisfiability 185 (2009), 289–338.