Vehicle Sales Configuration : the Cluster Tree Approach

Bernard Pargamin[1]

Abstract. Most current commercial configurators, mainly based on constraint propagation, are known, among other limitations, to be unable to guarantee complete deduction in free order interactive configuration. Yet, on specific ranges of products, completeness of inference can be achieved. New approaches, based on the compilation of the system of constraints defining the diversity of the product to configure, are making their way. We present such a complete configuration engine, C2G, developed internally by Renault, with a list of basic functional requirements that we couldn't find together in any commercial configurator. The compiled approach we describe made their implementation possible, especially the much needed 'filtering on the price' and 'user driven conflict resolution'.

Basically, compilation splits the configuration variables into clusters organised in a tree. Variables in a cluster are in high interaction, while variables of different clusters are in low interaction. Inference is done locally in each cluster and the updated state of the node is propagated between nodes to provide a global solution. Inference proves to be linear in the tree structure, allowing short and predictable run-times. This representation is especially efficient for low treewidth domains, which is the case for the automotive industry.

1 INTRODUCTION

Web enabled sales configurators are widely used in the automotive industry, but the level of functionality offered by these implementations is deceptively low. A basic customer's question such as: 'I have $15,000. What is the cheapest vehicle I can buy with a diesel engine and an automatic gearbox ?' won't generate any direct answer from the online configurators of the major vehicle builders world-wide. You will have to study the documentation first, choose a model outside the configuration process, configure your vehicle with possibly inadequate cost information, eventually iterate on this process and backtrack if you are unlucky until you get what you are looking for.

Customers don't like to iterate: they want responsiveness, accuracy, full information on the consequences of their choices, free order and full freedom of choice. They want effective filtering on their budget. They don't like backtracking. If your on-line configurator can't give that, they will zap.

Renault, a major vehicle manufacturer, was bothered by this situation and articulated its configuration vision in an internal white paper (2001, unpublished). It then arrived at the conclusion that the current mainstream commercial configurators, because of their general purpose, were unable to offer this kind of rich functionalities on Renault range of vehicles, and it decided to switch to an internal configuration project: C2G (2nd Generation Configurator) in order to provide them on the web by Q3 2002.

[1] Renault S.A., Direction des Technologies et Systèmes d'Information 92109 Boulogne, France.
e-mail: bernard.pargamin@renault.com

C2G was an outcome of a 'Product Diversity Modelling' Project initiated six years ago. In this framework, Product Diversity is represented by a compiled data structure on which operates a deductively complete configuration (inference) engine whose response time is linear on the size of the compiled structure. Basically, the set of boolean variables used in the constraints restricting diversity is split into clusters by converting a graphical representation of the problem into a tree structured representation where each node in the tree represents a tightly-connected sub-problem, and the arcs represent the loose coupling between sub-problems. Inference is done locally at each node and the updated state of the node is propagated between nodes to provide a global solution. Inference proves to be linear in the tree structure.

This paper is structured as follows: We will first outline Renault's basic requirements for configuration in section 2, then we will discuss Vehicle Diversity Specification and modelling in section 3. In section 4, we will present the principles of Diversity Compilation we used in C2G, and we will show how very fast complete deductive inference can be obtained by state propagation on a cluster tree. Section 5 will focus on the problem of price representation, the only non boolean part of our data, and we will present a propagation algorithm that finds the cheapest vehicle consistent with the current state of the configuration in linear time, thus enabling fast exact filtering by a customer's constraint on the maximum price. Section 6 will outline consistency restoration in C2G, a very basic feature for a commercial configurator, and we will present some concluding remarks in section 7.

2 RENAULT'S BASIC CONFIGURATION REQUIREMENTS

The functional configuration requirements we discuss here come from our experience with the shortcomings of the mainstream commercial configurator we used previously. They reflect the views of both Renault Marketing managers and configuration specialists.

2.1 Completeness of inference

should be guaranteed. This means that *all* that can be logically inferred at a given state of the process is actually inferred. No dead end, no backtracking for the user. This is absolutely the number one item on top of the requirements list. Configurator vendors say it is impossible to obtain, because of the NP-Completeness of the problem. Actually, we arrived at another conclusion, as far as automotive sales configuration is concerned.

The importance of this point is better seen if we realize that the diversity specification could be inconsistent, and the configurator wouldn't even detect it. The failure rate of the configuration would then be 100%.

2.2 Full information on the consequences of a choice should be provided (may be optionally in some cases) : excluded or implied features must be exhibited because they may cause the customer to change his choice when he sees the undesirable consequences of his projected choice. This feature is a dynamic way to inform the user of the logical constraints, at the moment they will trigger inference. Without it, the user would discover the effect of the constraints too late, and would have to force a conflicting choice, which would cause computational overhead to the configurator to restore consistency, and a real nuisance for the user. Full information includes also:

2.2.1 Pricing information: in the real life, no one makes choices without any cost consideration. You wouldn't fill your cart in a supermarket where you only discover the final price at the pay-desk. Moreover this pricing information must be accurate and guaranteed. At every step, we need the minimum price to be shown for each choice, i.e. *the price of the cheapest vehicle fulfilling the partial configuration*. This is very different to simply showing the price of an option, often irrelevant in order to show the real financial impact of a choice: if option A implies option B or option C (a very common type of constraint), in front of A you should put: Previous Min Price + price option A + min(price option B, price option C). Adding only the price of option A would be misleading.

2.2.2 Delivery timing information is also needed. This is not trivial, because you need a way to evaluate the best possible delivery date on a partially configured vehicle. This is the only way to show which specific choice makes the delay grow. An evaluation at the end of the configuration would be too late and would not say what to change in the vehicle features to improve its delivery date. In the most sophisticated implementation, this requirement is a real challenge: a centralized application must manage capacity constraints and manufacturing schedules to give accurate estimates. As a first step, an approximation based on manufacturing eligibility dates and marketing dates would probably be sufficient.

Existing stocks of readily available vehicles should be taken into account in the configuration process.

2.3 Filtering on a maximum budget and/or a maximum delivery lead time must be possible. In certain cases (fleet configuration) they are strict constraints that can't be bypassed.

2.4 Financing possibilities (loan) should be integrated in the process in the same way as the price. The user is not expected to learn at the end of a 10 minutes configuration that this vehicle exceeds its financing capability: we need a filtering capability similar to price filtering. This is much more difficult to achieve because there are several types of loan and the rate of the loan may depend on the final price of the vehicle, which is not known until the end of the configuration.

2.5 Adequate treatment of option packs must be provided. Packs generate implicit constraints that must be explicited (for instance, two packs with a common feature are exclusive, so as not to make the customer pay twice that feature). In the configuration process, they should be treated just like any other features.

2.6 Freedom of choice

∞ **Free configuration order**: *Configuration is a completely user driven process*: the user chooses the specification order that best fits his priorities. There is no compulsory "natural order", but of course, when the user has expressed his priorities, the configurator can choose the order that best optimizes its computational task. There is a suggested order that optimizes configuration, and if the user is pleased with it, we use it.

∞ **Negative choice**: A feature can be either chosen or *excluded* by the customer. Excluding a feature is a valid choice. Allowing only the selection of a feature can't prevent an undesired feature to appear inadvertently by having been implied by a constraint.

∞ **Permissiveness:** The user can change his mind without any restriction during the process: an inconsistent choice can be forced at any step by the customer, and it is the configuration engine's task to *restore consistency*, by proposing changes to some previous choices. This situation may arise routinely simply because the user doesn't know the logical constraints concerning features, so he may be surprised by their effect. It will happen less often if he enters his choices by decreasing priority order.

Consistency restoration must be completely *user driven*: there are usually several ways to do it, and it is the user's choice to determine which one is best for him, not the system's choice.

∞ **No obligation to choose**: Any choice at a given state can be delayed by the user: "I don't know yet" or "I don't bother" are perfectly acceptable answers to a proposed choice. So, the user is never stuck because he has no answer to give. The configurator will come back later on these choices, and sometimes it will not even be necessary because the answers will be automatically deduced from other user's choices. This applies of course to the model's choice, and it means that we need unrestricted transversality in the configuration process.

2.7 No prerequisite knowledge of the product should be expected from the customer. Any choice should have an associated help giving extensively the pro's and the con's of each alternative. This requirement introduces some form of *recursive configuration* inside the primary configuration. For instance the choice of a radio set is itself a second level configuration based on features such as: number of channels, power per channel, CD with charger, traffic information , …Just showing the list of the various possible radio types with their price is clearly not enough to let the user make an informed choice.

2.8 Tranversality of the configuration process is required: first generation configurators require that the process can only begin when the customer has already chosen a vehicle Model. This seems a reasonable assumption, but *it is not*. The customer is thus expected to have previously collected paper information about models and versions, studied it carefully by himself and to have made his choice alone. This means that a whole important part of the configuration process is left aside, without any assistance.

We think that Model and Version choices are obviously part of the configuration process, and that in this case, these choices might be the consequence of other features choices. Note that this reverses the traditional view that the natural order is to begin with Model/Version, and then to choose options whose availability is the consequence of the version's choice.

2.9 Automatic completion of a partial configuration must be available, with several alternate optimization criteria: cheapest, most quickly available or mixed, such as 'the cheapest vehicle under $15,000 available in less than 3 weeks'.

2.10 Assisted conflict resolution

The main configurator's task is to prevent the appearance of conflicts by automatically excluding all choices that wouldn't be consistent with the user's former choices. If the configuration engine guarantees completeness of inference, a conflict should never appear. But even in this case, the user may change his mind. He can force an inconsistent choice, and the configurator must analyze the inconsistency so as to find the minimal sets of former choices that have to be revised to restore consistency. Without this feature, the user would have to start a new configuration session and to repeat most of his previous choices.

2.11 Response time must be compatible with an interactive web usage: at most a few (< 2) seconds in the worst case.

3 RENAULT VEHICLE DIVERSITY SPECIFICATION AND MODELLING

Vehicle diversity (i.e. the number of distinct possible customer's choice, can be huge, reaching at Renault 10^{20} at the engineering stage and up to 10^{10} at the showroom. Moreover, this diversity is a source of major industrial complexity because of technical, commercial and legal constraints resulting in numerous implications and/or exclusions of features and options.

In the automotive industry, Product Diversity Specification (PDS) is addressed by an ISO norm: STEP-AP214 (ISO 10303-214:2001). In STEP terminology, vehicle descriptive features or attributes are known as *specifications* (abbrev. *specs*), grouped in *specifications_categories* (abbrev. *spec_cat*).

A spec_cat is a variable whose discrete possible values are the specs. For example we have a spec_cat 'gearbox type' with 2 specs: 'manual' and 'automatic'. A spec is an instantiation of a spec_cat, and we will attach a boolean variable (abbrev. *bvar*) to each spec, among which we will find our configuration variables.
The set of spec_cat, called *lexicon*, is globally common to all models of the constructor's range, with some exceptions. A specific vehicle is uniquely defined by a tuple of specs, one and only one for each spec_cat of the lexicon.

A subset of the vehicle range is intentionally defined by a boolean formula on specs, called a *specification_expression* (abbrev. *spec_expr*).

Vehicle diversity is usually not practically enumerable, and this prevents an extensional definition such as a simple enumeration. Diversity must be intentionally specified, by spec_expr.

The basic idea is that the entire combination of specs is allowed, except those that are explicitly excluded by boolean constraints expressed by spec_expr. The task of PDS is to build, control, store and retrieve these constraints.

Actually, for ergonomic and convenience reasons, engineering staff in charge of PDS do not usually directly manipulate spec_expr, but instead they complete compatibility tables of various shapes and size, they explicitly enumerate valid tuples of specs on selected subset of the lexicon, and sometimes they write spec_expr in a simplified syntax.

The main point is that PDS of any form can be translated into boolean constraints in linear time, giving a CDNF (Conjunction of Disjunctive Normal Form). Alternatively, a CSP (Constraint Satisfaction Problem) formalism can be used, but it is known to be equivalent.

Implicit constraints due to the spec_cat/spec structure must be explicitly added: inside a spec_cat, any 2 specs are exclusive and the disjunction of the specs is TRUE.

Our model of vehicle diversity is thus a pure boolean CDNF, equivalent to a propositional theory, and in this first step, we have transformed Renault PDS into a propositional knowledge base (PKB) able to feed a configuration engine. In this model, most interesting tasks become *reasoning* tasks, involving logical inference [21, 19, 20] , such as satisfiability tests and clausal entailment.

This first modelling step is necessary but is not sufficient, because as it turns out, most available complete algorithms for those tasks are exponential in the number of variables or in the number of constraints. Brute force, even with gigahertz computers, leads us instantly to exponential blow-up.

A typical Renault Model is described by a lexicon of 100 spec_cat containing about 400 specs. The CDNF has about 2 to 3000 conjunctive terms and commercial diversity (size of the search space for the configurator) ranges from 10^3 to 10^{10} . At the engineering stage, diversity is much higher, reaching 10^{20}.

An example of translated PDS for a Renault X64 model can be found for benchmarking purpose at :
http://www.irit.fr/ACTIVITES/RPDMP/CSPconfig.html
A cnf sample file is also available on demand (10813 clauses on 658 variables)

4 PRINCIPLES OF RENAULT CONFIGURATION ENGINE C2G

We have to deal with the well known NP-completeness of the propositional satisfiability problem, and consequently of boolean configuration. But NP-completeness doesn't prevent polynomial or even linear solutions in specific domains. While a general polynomial time configurator is impossible, a linear time vehicle configurator proves possible if you can exploit the structure of the domain.

C2G is fully based on the standard compiled representation of Renault vehicle diversity in the form of a cluster tree that has been used in various applications since 1995 . Compiled approaches for propositional theories, increasingly popular, consist in investing once in a heavy off-line compilation whose computational overhead can be amortised with many fast on-line queries. The size of the compiled structure is not directly related to the number of models of the theory and can be exponentially smaller. BDD (binary decision diagrams [5]), finite automata (a variant of BDD [9, 18]), DNNF (Deterministic negation normal form [7]), prime implicates [15] are mainly used to this effect. See also [13] for a classification of compiled approaches. We must stress that there is no *best* compilation, because it depends on the structure of your problem and on what you intend to do with your data. We, at Renault, decided to use cluster tree compilation, as the most effective and efficient representation of vehicle diversity, seen as a propositional theory, for deductive and probabilistic inference.

Cluster trees are well known in the field of probabilistic inference, especially in the Bayesian Network area. A main advance was given by Lauritzen and Spiegelhalter [6, 11] with the use of a secondary structure, the cluster tree (also called join tree, junction tree, clique tree) and an exact probabilistic inference algorithm based on Probability Propagation in the Cluster Tree (PPCT). See [10] for a procedural description of PPCT. We are not aware of any significant use of cluster trees in deterministic inference problems, except by Darwiche [7, 8] and Dechter [22].

4.1 Construction of the cluster tree

A cluster tree is a tree whose nodes are clusters of variables, satisfying the running-intersection property: if a variable is shared by 2 nodes, it must be present in every node on the path between

the 2 nodes. This property is needed for the completeness of inference algorithms operating on the cluster tree. The *treewidth* of the cluster tree is the number of variables in the largest cluster. The best cluster tree is the one with minimal treewidth w* (the size of the cluster tree is exponential in w*). In a sense, the treewidth is a measure of the connectivity of the constraints graph.

We are looking for the minimal treewidth cluster tree in which every constraint fits in at least one cluster (i.e. all the bvars of the constraint belong to this cluster).

We first build the *interaction graph* of the boolean constraints in the following way: we create a node for each boolean variable, and put an edge between node i and j if v_i and v_j appear simultaneously into a constraint. This graph shows graphically the structure of the problem.

We then built the cluster tree with the smallest possible treewidth. This is a NP-hard problem, but a number of algorithms are available in the literature for computing a close to optimal cluster tree from an acyclic undirected graph. [2, 3]. Moreover, a linear time algorithm for finding tree decompositions of small treewidth was presented by Bodlaender [4]

4.2 Cluster implementation

Clusters are supposed to be small enough so that brute force can solve any inference problem inside the cluster. Yet, a clever implementation is still of utmost importance to improve speed and allow for bigger clusters. First we build the set of all boolean variables instantiations consistent with the constraints that fit in the cluster (logical models). We associate to each 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.

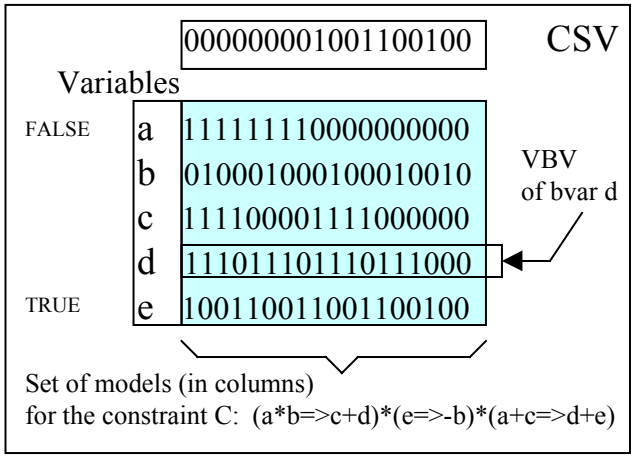

Figure 1. A cluster implementation

We then build 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 state of knowledge on the values of the bvar of the cluster, FALSE otherwise.

During the configuration process, when a bvar is instantiated by the user, the CSV is re-evaluated by a boolean AND operation between the prior CSV and the VBV or its negation. The effect of the change is then propagated to all other bvar of the cluster that are deduced to TRUE (resp. FALSE) if its VBV (resp. the negation of its VBV) is compatible with the CSV.

Local inference in the cluster is sound and complete in linear time in the number of models. In the worst case, this size is exponential in the number of variables, but as a cluster groups a small number of variables with high interaction, the real size is usually much lower. This cluster implementation has two nice properties:

- the more we add constraints to the cluster, the smaller its size is.
- All operations are vector boolean operations, taking advantage of internal register parallelism on 32 or 64 bit words.

4.3 Propagation on the cluster tree

We choose an arbitrary cluster as the root of the tree.

Two adjacent nodes of the cluster tree share variables. We can treat these variables as a cluster, named the *sepset*, that we introduce on the graph as a new node between the two clusters.

Propagation of state from cluster i to parent cluster j involves 2 phases:

- *Marginalisation*: propagation from cluster i to sepset ij: we set to FALSE all positions of the CSV of the sepset whose compatible positions in the CSV of cluster i are all FALSE.
- *Dispatching*: propagation from sepset ij to cluster j: For every position k of the sepset with a FALSE value, we set to FALSE all compatible positions of the CSV of the cluster j.

Whenever the state of a cluster changes, we make a global propagation on the whole structure, by propagating the change up to the root following the path of the cluster tree, and then propagating down to the leaves. During this process, whenever a cluster's state changes, it propagates the change to all the variables in the cluster.

In this way all bvar truth values that can be deduced from a state change are deduced, and we have a deductively complete truth maintenance system.

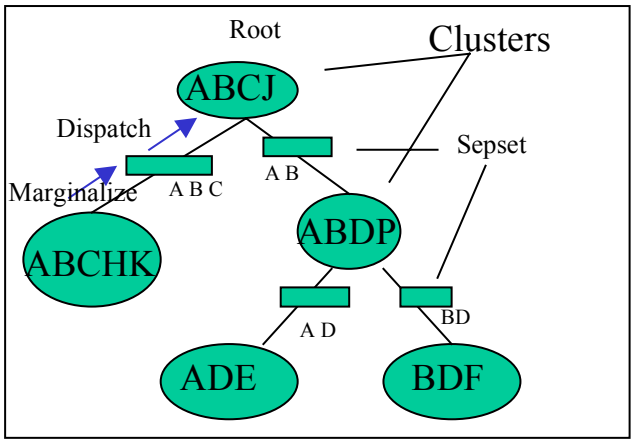

Figure 2. A cluster tree with its sepsets

4.4 Main benefits of the cluster tree

Cluster tree compilation is efficient and compact because logical independence is built in the structure: The compiled structure fully exploits the factorisation resulting from the logical conditional independence relations expressed graphically by the cluster tree: any two nodes and their descendants are logically independent given an instantiation of their shared variables.

4.5 Characteristics of Renault's cluster trees

The cluster tree structure is not related in any way to the physical structure of the car. The treewidth is around 40 to 50 for our model's cluster tree representation, depending on the model. But,

as noted above, the constraints application strongly reduces the size of the biggest cluster, usually around a few hundreds columns.

Constraints can change every week, so a full offline compilation has to be done 4 to 6 times a month for each country: 10 to 15 cluster trees are compiled in about 2 minutes, with a total storage requirement of 300 to 400 KB for the logical structures.

4.6 Other uses of the cluster tree

The cluster tree is a highly tractable compiled representation of diversity for Renault vehicles: a number of different NP-complete real life problems involving diversity can be solved with it in polynomial time, and often in linear time.

- Probabilistic applications: Probabilistic constraints used to express vehicle forecast fit in the cluster tree, and by solving a linear problem, we can compute a factorised joint probability distribution (JPD) consistent with the forecast. The global JPD is split into local JPD, expressed at the cluster level by a probability vector for each model of the cluster.
- Propagation algorithms: the same scheme of propagation up and down allows for rapid calculus of the
  - cardinality of the solutions space
  - maximum entropy distribution by iterating propagations
  - entropy of a factorised jpd, cross entropy between 2 JPD's
  - probability of a variable under evidence on other variables

Within Renault Information Systems, they are used routinely in various applications, with about 60,000 queries per day:

- Consistency of Product Diversity Specification
- Consistency of parts documentation with diversity specification
- Consistency of vehicle forecast with diversity specification
- Vehicle forecasting with Maximum Entropy completion of incomplete forecast
- Engineering and Commercial vehicle configuration

5 FILTERING ON A MAXIMUM PRICE

In theory, we could handle the price in a pure boolean way:
We add a new spec_cat 'Price' with specs ranging from $5,000 to, say, $50,000 (by increments of $1), and calculate the price of every vehicle configuration, then, for every value x we build the boolean constraints expressing the equivalence between spec $x and the DNF of all vehicle configurations costing $x.

Practically, it would be intractable, and we will of course proceed in another way: The idea is to represent the cost function as a utility function that can be factorised on the cluster tree, and to use a propagation algorithm that finds the cheapest vehicle compatible with the current state of the configuration in linear time, thus enabling fast exact filtering by a customer's constraint on a maximum price.

5.1 Price List external specification

The total cost of a vehicle is the sum of pricing elements that apply only for certain vehicles. There is a price for a version, and prices for the various options available on this version.

A Price List is a list of *pricing elements*, each consisting of a set of 2 elements: a spec_expr and a price.

5.2 Price List internal representation

We assume that every spec_expr fits in at least one cluster. If it were not the case, we would rebuild the cluster tree to satisfy this condition. We associate a numerical Price Vector to the (boolean) state vector of each node and sepset of the cluster tree.

The i^{th} position is set to the sum of all the prices associated with spec_expr satisfied with the partial instantiation i of specs in the cluster. In this way, the price vector stores the contribution of each partial instantiation of the bvars of the cluster to the global price.
Propagation of price vector from cluster i to parent cluster j involves 2 phases:

- *Marginalisation*: propagation from cluster i to sepset ij: set every position k of the price vector of sepset ij to the min of its compatible positions in the price vector of cluster i.
- *Dispatching*: propagation from sepset ij to cluster j: For every position k of the sepset, set all compatible positions of the price vector of the cluster j to the k^{th} value of the price vector of the sepset.

We are now ready to describe our algorithm MPP (Minimal price by Propagation):

5.3 Algorithm MPP

1 Initialise each cluster's Price Vector
2 For each node, from the leaves up to the root:
  - Combine the direct descendant messages by adding their propagated price vector
  - Propagate up:
    - Marginalize min price on the sepset
    - Dispatch up
3 Select the min price on the price vector of the root

6 RESTORING CONSISTENCY

This is a major feature for a commercial configurator, as it has been discussed in a earlier section.

Whenever a contradiction is deliberately entered by the user, C2G finds a minimal subset of inconsistent former choices, in linear time in the number of former choices. The user then has to change at least one of these choices. This change may in turn create a new contradiction, so the process is recursive. The maximum price given by the user is treated like any other vehicle feature's choice. Very commonly, there will be an inconsistency between the maximum price and the set of n options chosen, showing a n+1 choices inconsistency, that can be eliminated by changing the maximum price or dropping any subset of options, at the user's request. In any case, restoring consistency is a fully user-driven process.

Finding a minimal subset of inconsistent former choices is performed in this way:
We maintain an ordered list of user choices. The completeness of inference of G2G guarantees that the contradiction is detected when it happens, on the last choice. We then reorder the list by putting this last choice in first position, we make a full roll-back in the configuration session, and a roll-forward with the reordered list, reintroducing one by one each former choice. We will necessarily encounter a new contradiction, that will give us the 2^{nd} term of the contradiction, and so on until all terms of the contradiction are grouped in the beginning of the list.

7 SUMMARY AND CONCLUSION

Modelling vehicle diversity by a system of boolean constraints is a first necessary step, but it is insufficient to solve efficiently NP-Complete problems of interest such as clausal entailment, and especially to feed a complete configurator.

Renault C2G configurator does not make inferences by constraint propagation, but adds a second crucial step by compiling the constraints into a secondary structure, a cluster tree on which complete deductive inference by state propagation is linear on the size of the compiled structure (which in turn happens to be exponential in the treewidth w* of the interaction graph). Provided that w* is small, we can perform complete inference with a bounded response time guarantee.

As it happens, small treewidth is the general case for vehicles diversity, as well as many other industrial domains, and our approach has a certain generality.

The key step to transforming a NP-Complete general problem into a linear time algorithm is thus to exploit the *structure* of the problem, as expressed by the connectivity of the interaction graph. The small treewidth of this graph allows for fast dynamic construction of the cluster tree from the boolean constraints and for compact representation of the nodes of the cluster tree.

This off-line compilation allows for very fast predictable on-line queries, and the computational overhead of the compilation can be easily amortised with thousands of queries. Compilation pays for itself.

Propagation algorithms we developed are closely related to probability propagation (see [6, 10, 11, 15]).

These ideas were implemented at Renault in a new configuration engine, C2G, that will be used by Q3 2002 for commercial configuration applications, proving their robustness in real life industrial-strength problems.

REFERENCES

[1] Amir, E and McIlraith, S (2001) *Theorem proving with structured theories,*
17th Intl' Joint Conference on Artificial Intelligence (IJCAI'01), 2001
http://www-formal.stanford.edu/eyal/papers/oor-theory-1.0-ijcai-final.ps

[2] Amir, E (2001) *Efficient Approximation for Triangulation of Minimum Treewidth,* 17th Conference on Uncertainty in Artificial Intelligence (UAI '01), 2001.
http://www-formal.stanford.edu/eyal/papers/decomp-uai01-final-1.0.ps

[3] Becker, A and Geiger, D. *A Sufficiently Fast Algorithm for Finding Close to Optimal Junction Trees*
In Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence 1996

[4] Bodlaender, H. L. (1992*) A linear time algorithm for finding tree-decompositions of small treewidth.*
ftp://ftp.cs.uu.nl/pub/RUU/CS/techreps/CS-1992/1992-27.pdf

[5] Bryant, R. (1992) *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams.*
In ACM Computing Surveys Vol. 24 n° 3.
http://www.cs.cmu.edu/~bryant/pubdir/acmcs92.ps

[6] Cowell, Dawid, Lauritzen, Spiegelhalter. (1999) *Probabilistic Networks and Expert Systems*, Springer Verlag .

[7] Darwiche, A. (1999) *Compiling knowledge into decomposable negation normal form.*
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1999
http://singapore.cs.ucla.edu/darwiche/dnnf.ps

[8] Darwiche, A. (2000) *On the Tractable Counting of Theory Models and its Applications to Belief Revision and Truth Maintenance*. in International Workshop on Belief Change & Journal of Applied Non-Classical Logics, 2000
http://www.cs.ucla.edu/~darwiche/count.ps

[9] Fargier, H and Amilhastre, J (2000) *Handling interactivity in a constraint based approach of configuration* ECAI 2000

[10] Huang C. , Darwiche A. (1996) *Inference in belief networks: A procedural guide.*
International Journal of Approximate Reasoning, 15(3):225-263.
http://www.cs.ucla.edu/~darwiche/ijar95.pdf

[11] Lauritzen, S. L. and Spiegelhalter, D. J.(1988) *Local computations with probabilities on graphical structures and their application to experts systems. In journal of the Royal Statistics Society,* B, 50,157-224 1988

[12] Mathieu, P and Delahaye, J.P. (1994) *A kind of logical compilation for knowledge bases.*
Theoretical Computer Science 131 (1994) 197-218
ftp://ftp.lifl.fr/pub/projects/achievement/tcs94.ps.gz

[13] Marquis, P and Darwiche, A (2000), *A Perspective in Knowledge Compilation* In IJCAI-01.
http://www.cs.ucla.edu/~darwiche/ijcai-01.ps

[14] Marquis, P (1995) *Knowledge Compilation Using Theory Prime Implicates*. IJCAI 1995

[15] Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*
Morgan Kaufmann, San Mateo, California, 1988.

[16] Roussel, O (1997)
L'achèvement des bases de connaissance en calcul propositionnel et en calcul des prédicats.
Thèse de doctorat. Lille 1997
ftp://ftp.lifl.fr/pub/projects/achievement/these_roussel.ps.gz

[17] Schrag, R and Miranker, D. (1996) *Compilation for critically constrained Knowledge Bases.*
submitted to the Journal of Artificial Intelligence Research.
http://www.iet.com/users/schrag/my-papers.html

[18] Veron, M Fargier, H and Aldanondo, M (1999) *From CSP to configuration problems*
AAAI-99 Workshop on Configuration
http://wwwold.ifi.uni-klu.ac.at/~alf/aaai99/08Veron.doc

[19] C. Sinz, A. Kaiser, W. Küchlin (2000) *SAT-Based Consistency Checking of Automotive Electronic Product Data.* Presented at the ECAI 2000 Configuration Workshop, Berlin.
http://www-sr.informatik.uni-tuebingen.de/projects/pdm/ecai2000.ps

[20] C. Sinz, A. Kaiser, W. Küchlin (2001) *Detection of Inconsistencies in Complex Product Configuration Data Using Extended Propositional SAT-Checking.* Proceedings of the 14th International FLAIRS Conference, AAAI Press

[21] Kaiser, A. and Küchlin, W (2001) *Automotive Product Documentation.* Proceedings of the 14th International IEA/AIE Conference, Springer, 2001.

[22] Dechter, R and Pearl, J (1989) *Tree Clustering for Constraint Networks*, Artificial Intelligence pp. 353-356 1989
http://www.ics.uci.edu/~csp/r06.pdf