Extending Cluster Tree Compilation with non-Boolean variables in Product Configuration: a Tractable Approach to Preference-based Configuration

Bernard Pargamin
Direction des Technologies et Systèmes d'Information
Renault S.A.S. Boulogne, France
bernard.pargamin@renault.com

Abstract

Product knowledge compilation is gaining momentum in configuration, because it provides complete deduction with a guaranteed bounded response time, which is the basis for truly interactive and accurate configurators. In order to be compiled, a Product Knowledge Base must be expressed in propositional logic or in a CSP formalism with multi-valued discrete variables. This is sufficient to express product diversity and to allow a basic configuration based on product features only. But it is not sufficient to efficiently implement sophisticated strategies using non-Boolean functions such as price or user preference, and the resulting usability of the configurator will be poor.

These optimization functions can be expressed in a Penalty Logic framework, and we show in this paper that within the Cluster Tree compilation scheme, weighted statements can be incorporated to the compiled structure, allowing tractable queries and adding desirable functionalities such as price filtering and user preference maximization, with exact backtrack-free algorithms and with the same bounded response time guarantee as a basic configuration.

1 Introduction

Product *knowledge compilation* is gaining momentum in product configuration, because it has the potential to provide complete deduction with a guaranteed bounded response time for structured product domains where the treewidth of the interaction graph of the constraints is low.
Various forms of compilation have been proposed, [Darwiche and Marquis, 2001; Sinz, 2002]

The minimal requirement is that clausal entailment must be allowed in linear time on the size of the compiled Product Knowledge Base (PKB). This directly ensures tractable completeness of inference in a configuration process.
This requirement is met by several compilation strategies: d-DNNF, OBDD, prime implicates, and Cluster Tree compilation [Dechter and Pearl, 1989].

In order to be compiled, a PKB must be expressed in propositional logic or in CSP formalism with multi-valued discrete variables. Although this is sufficient to express product diversity in many domains (e.g. vehicle diversity) and to allow a basic configuration based on product features only, it is definitely not sufficient to implement tractable sophisticated strategies using non Boolean functions such as price or user preference, and the resulting usability of the configurator will be poor.

In this paper, we focus on the cluster tree compilation approach developed and used at Renault as the basis of its second generation vehicle sales configurator [Pargamin, 2002] currently deployed on the company's B to C web sites. We will show that we can incorporate weighted logical statements in a Cluster Tree compiled structure. This will allow tractable queries and will add functionalities of interest such as price filtering and user preference maximization with exact backtrack-free algorithms with the same bounded response time guarantee.

We will discuss, in the context of a partial configuration, the problem of finding efficiently optimal configurations with respect to a variety of optimisation functions that can be expressed in a Penalty Logic framework [Darwiche and Marquis, 2002] and we will show specifically that Preference-based Product Configuration, a much needed feature for a web based sales configurator, can be handled within the Cluster Tree compilation scheme, as a special case of more general utility functions.

This paper is structured as follows: In section 2, we briefly summarize propositional knowledge compilation and the Cluster Tree compilation approach, section 3 outlines the shift from logical to weighted cluster trees and section 4 discusses the implementation of preference-based configuration as weighted cluster trees. Finally, section 5 focuses on dealing with user preference during the configuration process.

2 Knowledge compilation: the Cluster Tree approach

We are mainly interested in the sub-class of configuration problem where the product generic model can be expressed as a set of propositional constraints, or alternatively as a

Constraint Satisfaction Problem (that can be transformed into a set of propositional constraints by a polynomial algorithm). This is the case for important mass customized complex industrial products such as cars.

Propositional knowledge compilation has attracted much interest in recent years as an effective way to deal with the intractability of propositional logic [Darwiche and Marquis, 2001]. A propositional theory is turned into a compiled theory off-line, on which multiple queries can then be answered tractably.

For a configuration use, we require that clausal entailment should be performed in linear time on the size of the compiled theory, which is here the Product Knowledge Base. This directly ensures tractable completeness of inference in an interactive configuration process.

This requirement is met by several compilation strategies: d-DNNF [Darwiche, 1999], OBDD [Bryant, 1992], prime implicates [Marquis, 1995; Sinz, 2002], Cluster Tree compilation [Dechter and Pearl, 1989, Pargamin, 2002] and finite automata [Amilhastre *et al.*, 2002].

The size of the compiled theory is thus crucial for performance issues. For d-DNNF and cluster tree compilation, this compiled size is exponential in the treewidth of the interaction graph of the constraints, a parameter measuring the connectivity of the constraints. So, compilation is not adequate to treat *any* propositional or CSP problems, but only low treewidth problems. Under this assumption compilation schemes can be very efficient because they exploit the structure of the domain to achieve a compact compiled form. The compiled size is not related to the number of logical models of the theory and can be exponentially smaller. It is compact because conditional logical independence is captured and integrated into the compiled structure.

Compilation benefits include:

- High runtime performance since most of the work is done at compile time
- Predictable response time depending only on the cluster tree size
- No influence of syntactical variations in the formulation of the constraints

The main shortcoming is that we can only directly treat configuration variables with discrete domains.

2.1 Cluster tree compilation:

(for a more detailed discussion see [Pargamin, 2002])

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. Tractable 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.

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*).

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

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

We then use a graph-partitioning algorithm to split it into non-connected sub-graphs by removing a minimal set of nodes. We apply the algorithm recursively until the size of the sub-graphs are tractable. This defines a cluster tree of close to minimal treewidth where the leaves correspond to the sub-graphs and the intermediary nodes correspond to shared variables between descendant clusters.

Cluster implementation

Clusters must 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 Boolean variable x_i a Boolean vector (VBV) whose j^{th} position is set to TRUE if x_i is TRUE in the j^{th} model, FALSE otherwise.

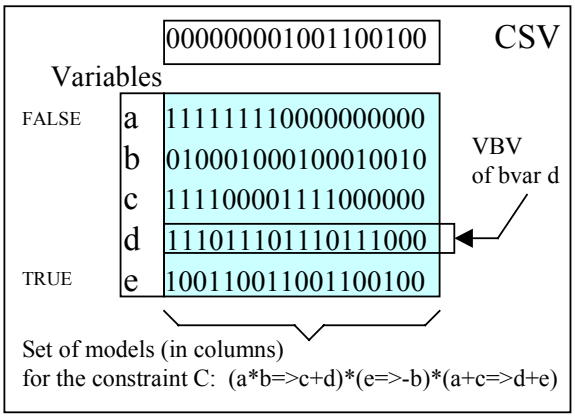

Figure 1. A cluster implementation

We then build a cluster state vector (CSV), a Boolean vector whose j^{th} position is set to TRUE if the j^{th} model is consistent with the current state of knowledge on the values of the Boolean variables of the cluster, FALSE otherwise.

During the configuration process, when a Boolean variable 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 Boolean variable of the cluster that are deduced to TRUE (resp. FALSE) if their VBV (resp. the negation of their VBV) is compatible with the CSV.

Local inference in the cluster is sound and complete and can be computed 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.

2.2 State 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.

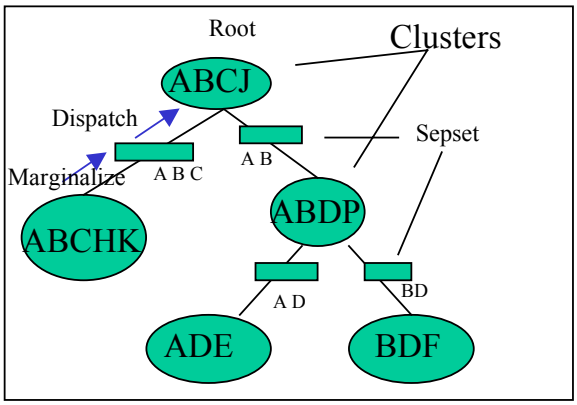

Figure 2. A cluster tree with its sepset

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

3 From logical to weighted cluster trees

Constraints compilation is limited to knowledge bases with discrete variables and it seems to be a major shortcoming of this approach, because it is clear that there are some important customer choice criteria that cannot be modeled by discrete variables: price is a good example in vehicle sales configuration. Any direct attempt to treat it in the same way as other configuration variables by discretization breaks the structure of the problem and causes the treewidth to increase excessively. A more practical approach is to consider the price as a utility function defined additively over the set of all car configurations. This can be done in the framework of *Penalty Logic* [Dupin de St Cyr *et al*.,1994].

In this framework we can represent a propositional *weighted base* W by a set of couples (F_i , w_i) where F_i is a propositional formula and w_i is the weight of the formula, the price to pay if the formula is violated.

$W=\{ (F_1 , w_1),\ldots(F_n , w_n) \}$

Weights are integers or $+\infty$, corresponding respectively to *soft* and *hard* constraints. A weighted base is a compact way to implicitly encode a total pre-ordering over the set of logical models of the hard constraints theory and we can compute the weight of each model additively, by summing the weights of the formulae that are not satisfied by the model.

We transform a propositional knowledge base K into a weighted base by setting to $+\infty$ the negation of all the constraints of K and by further adding weighted statements. The main inference problem in a weighted base is to find models of K with the minimal or maximal weight [Darwiche and Marquis, 2002]. For configuration purposes, we actually need this ability for any partial instantiation of variables of K. This problem is intractable, but it can become tractable on a cluster tree compilation of the weighted base.

3.1 Compilation of a weighted cluster tree

A compiled weighted cluster tree consists of a logical cluster tree extended with a weight vector for each cluster giving the contribution of each model of the cluster to the total weight.

We add to the hard (logical) constraints a set of tautologies built from the soft constraints (e.g. F_i or not F_i) in order to ensure that every soft constraint will fit inside a cluster (all its variables belong to the same cluster). We then build the cluster tree in the usual way, giving a logical cluster tree where we can assign every soft constraint to a cluster. We end the process by evaluating for every cluster its weight vector: for every logical model we sum the weights of the soft constraints it violates.

3.2 Weight Propagation algorithm

Now we can evaluate by propagation the minimal (maximal) weight of the set of models consistent with a partial instantiation of the variables.

Propagation of weight 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 weight vector of sepset ij to the minimum of its compatible positions in the weight 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 weight vector of the cluster j to the k^{th} value of the weight vector of the sepset.

We are now ready to describe our algorithm MWP (Minimal Weight by Propagation):

Algorithm MWP

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

Weight propagation is linear in the size of the clusters.

A price list is a good example of weighted statements: The total cost of a vehicle is the sum of pricing elements that apply only for certain vehicles. A Price List appears to be a list of weighted statements. Once compiled with the cluster tree, we obtain a priced cluster tree which allows us to precisely filter a configuration on a maximum price specified by the customer.

User preference is a further important example we will develop below.

4 Application to preference-based configuration

In the current trend toward smarter product configurators, integrating the user preferences in the process is a major challenge [Junker, 2001; Ardissono *et al.*, 2002; Brafman and Domshlak, 2002; Syrjänen, 2000].

More and more customers expect the same kind of personalized advice that can only be given by a salesperson today.

A configurator basically allows the user to select a valid product by interactively choosing the product's features while remaining consistent with the product constraints. But it has been emphasized [Brafman and Domshlak, 2002] that the mere satisfaction of the constraints is not enough: the configurator should lead the user toward the most desirable product for him. To be able to do that, it must have some information about the user's preferences.

A user's model must thus be associated with the product's model: user's preferences are part of this user's model [Ardissono *et al.*, 2002]. They can be either self-assessed at the beginning of the configuration, or estimated during the configuration by observing the user's behavior. In both cases, they us allow to define and compute at run time the specific desirability function of the products for this specific user.

A first way to deal with the customer's preferences is to present her with sets of alternate choices and to suggest her to express her choices in decreasing priority order: first, important choices that cannot be removed, then less important choices that could be changed in case of inconsistency with previous choices. In order to do this, you need a free order configurator. This user-driven choice ordering has two advantages:

1. It captures important aspects of user preference as the choice inside a set of alternate possibilities immediately shows which one are excluded,
2. It minimizes the occurrence of conflicts because the last conflicting choice will usually be changed to another one in the set of the alternate possibilities.

But for a complex product such as a car, this is definitely not enough. There is another level of user preferences that are not associated with configuration variables. These are high level preference axis, the *value dimensions* of the product [Ardissono *et al.*, 2002]: "reliability, safety, comfort, performance…" are not native configuration variables today in car sales configurators, but they should be integrated in the process.

These criteria of choice are fuzzy, not Boolean. Moreover, they should not behave as hard constraints, eliminating solutions, but as soft constraints, merely ranking remaining solutions.

We use *Multi-Attribute Utility Theory* (MAUT) for the construction of our *desirability function*.

MAUT is an evaluation scheme which is very popular and widely used by consumer organisations for product evaluation. It can be used to model user preferences for products that can be described by features: the overall evaluation v(x) of a product x is defined as a weighted addition of its evaluation with respect to its relevant *value dimensions* [Schäfer, 2001], which all share a common *utility* for the customer. For example, a car can be evaluated on the value dimensions *habitability, performance, safety, gas mileage reliability and comfort.*

The evaluation of a car feature is the weighted sum of its evaluations along the value dimensions d_i.

$$v(f)= \sum_{i=1}^{n} w_i v_i(f)$$

where $v_i(f)$ is the evaluation of the feature f on the i-th value dimension and w_i the relative importance of dimension i. The overall evaluation is the sum of its features evaluations.

Feature evaluations along the value dimensions, giving its contribution to each value dimension, are part of the input data for compilation, along with diversity specification, cost elements and other relevant marketing informations. They are not specified at run-time by the end-user but offline by the marketing department.

	Habitability	Performance	Safety	Gas mileage	Reliability	Comfort
Air conditioning	0	-2	1	-3	-1	5
ABS	0	0	5	0	0	0
ESP	0	0	5	0	0	0
Sun roof	1	0	0	-1	0	0
Leather seats	0	0	0	0	0	2

Figure 3: Features contribution to value dimensions.

The set of w_i reflecting the weight, the importance for the customer of each value dimension is the customer's preferences profile. It is assessed by a new customer at the beginning of the configuration session (in a few seconds) or retrieved from a CRM database. In case a new customer doesn't assess anything, a standard profile is used instead, reflecting current means values.

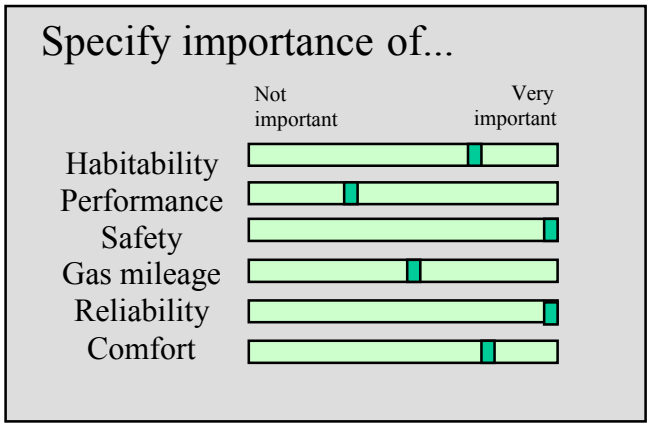

Figure 4: a car customer profile (on-line specification).

The user's preferences and feature's contribution are combined at run-time to compute the user's *desirability function* allowing us to compute a desirability value for each individual product.

MAUT is simple to implement, but it makes quite strong assumptions about the utility function: a common scale of evaluation of different criteria and aggregation of different criteria by an additive function. Automotive end-user preference is more complex but we assume MAUT is sufficient for our use even if we can't treat all context-dependent customer preferences.

5 Dealing with the user's preferences during configuration

So far, we have defined a utility function providing a quantitative measure of the desirability of the different final configurations for the customer, defining a strict partial order on the set of all configurations which we wish to use for optimisation purposes, e.g. maximizing desirability while completing a partial configuration.

We then notice that for a given customer, the set of attributes contributions constitutes a weighted base extension to the cluster tree and that, in the same way as price, the desirability function can be factorised on the cluster tree in the form of a desirability vector associated with each cluster. The same propagation algorithms can compute maximum desirability configuration in a partial configuration in linear time on the size of the clusters.

The use of a desirability function can significantly enhance the configuration experience for a customer.
The main uses are:

5.1 Ranking solutions and suggesting a "best choice" in a category:
Here is a sample screen for the choice of a gearbox:

Gearbox	Minimum price	Maximum desirability
1 Automatic gearbox	$14500	7
2 Manual gearbox 5 speeds	$13250	8.2
3 Manual gearbox 6 speeds	$13600	7.4

Here, maximum desirability means: desirability of the highest ranked car consistent with the current state of the configuration. It may be different from the feature's desirability because of the effect of the constraints. When presented with a choice of 3 gearboxes, the customer is informed that the second choice best suits his profile: if he chooses it, we can guarantee that at least one choice in further categories will maintain this desirability of 8.2.

5.2 Completing a partial configuration by maximizing the preference function: the configurator completes each remaining category by choosing the feature with maximum desirability.

5.3 Mixed strategies: completing a partial configuration by maximizing the preference function with a bounded price constraint.
Suppose the customer budget is $15000. Here, we have no exact solution to find the car under $15000 with maximum desirability without evaluating all solutions, but we can still

use a simple heuristic: the configurator will choose the feature giving the highest maximum desirability if its associated price is under $15000. Else, it will choose the variant giving the best guaranteed desirability increase for the money (i.e. the choice giving the best *efficiency*, defined as the ratio additional desirability / additional cost):
The initial situation (after the previous choice) was:

	Minimum price	(Min) desirability (Max)
	$13250	6 ($14500) 8.2 ($15500)

The current choice of gearbox is:

Gearbox	Minimum price	(Min) desirability (Max)
1 Automatic gearbox	$14500	7.2 ($14500) 8 ($14900)
2 Manual gearbox 5 speeds	$13250	6 ($13500) 8.2 ($15500)
3 Manual gearbox 6 speeds	$13600	7.1 ($13600) 7.4 ($14700)

efficiency of choice 1= (7.2-6)/(14500-13250)=0.00096
efficiency of choice 3= (7.1-6)/(13600-13250)=0.00314
Here, the configurator will make choice 3.

6 Conclusion

The main contribution of this paper is to show that user preference defined in a MAUT framework can be compiled in cluster tree-based configurators and then can be used tractably on low treewidth problems to compute the most desirable choice for each configuration variable, at every step of the configuration, so that a final product with this desirability rating is guaranteed to exist. This is a step toward the smarter web configuration needed to sell complex products such as cars or digital cameras without the assistance of a salesperson.

Such a preference-based configuration is currently being implemented in the Renault new configuration engine C2G, based on cluster tree compilation, so at this time no end-user feedback is available.

Price filtering and user preferences maximization are two interesting examples of the way we can extend cluster tree compilation to deal tractably with non-Boolean constraints in an exact way in product sales configuration.

References

[Amilhastre *et al.*, 2002] Jérôme Amilhastre, Hélène Fargier, Pierre Marquis. *Consistency restoration and explanations in dynamic CSPs Application to configuration.* Artificial Intelligence, 135,199-234 , 2002 ftp://ftp.irit.fr/pub/IRIT/RPDMP/AIJfinal1.pdf

[Ardissono *et al.*, 2002] L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, M.Meyer, G.Petrone, R.Schafer, W.Schutz and M. Zanker *Customizing the Interaction with the User in On-Line Configuration Systems* Configuration Workshop ECAI-2002 http://www.enstimac.fr/recherche/gind/ecai-2002-config-ws/Config-Wrksh-ecai02.pdf

[Brafman and Domshlak, 2002] Ronen I. Brafman and Carmel Domshlak. *TCP-Nets for Preference-based Product Configuration* Configuration Workshop ECAI-2002 http://www.enstimac.fr/recherche/gind/ecai-2002-config-ws/Config-Wrksh-ecai02.pdf

[Bryant, 1992] Randal Bryant *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

[Chin and Porage, 2001] Chin, D. and Porage, A. *Acquiring User Preferences for Product Customization*. In Proceedings of the 8th International Conference, UM 2001. http://mmlab.ceid.upatras.gr/hci/docs/UM2001/21090095.pdf

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

[Darwiche and Marquis, 2001] Darwiche, A and Marquis, P. *A Perspective in Knowledge Compilation* In IJCAI-01. http://www.cs.ucla.edu/~darwiche/ijcai-01.ps

[Darwiche and Marquis, 2002] Adnan Darwiche and Pierre Marquis. *Compilation of Weighted Propositional Bases.* In Workshop on Non Monotonic Reasoning –Toulouse 2002. http://www.cs.ucla.edu/~darwiche/nmr-02.pdf

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

[Dupin de St Cyr *et al*., 1994] F. Dupin de St Cyr, J. Lang, and Th. Schiex. *Penalty logic and its link with Dempster-Shafer theory.* In Proc. of the 10th Conference on Uncertainty in Artificial Intelligence (UAI'94), 1994.

[Junker, 2001] Ulrich Junker. *Preference programming for configuration.* Configuration Workshop IJCAI-2001

[Marquis, 1995] Pierre Marquis *Knowledge Compilation Using Theory Prime Implicates*. IJCAI 1995

[Pargamin, 2002] Bernard Pargamin. *Vehicle Sales Configuration: the Cluster Tree Approach* Configuration Workshop ECAI-2002 http://www.enstimac.fr/recherche/gind/ecai-2002-config-ws/Config-Wrksh-ecai02.pdf

[Schäfer, 2001] Ralph Schäfer. *Rules for Using Multi-Attribute Utility Theory for Estimating a User's Interests ABIS Workshop 2001* http://www.kbs.uni-hannover.de/~henze/ABIS_Workshop2001/final/Schaefer_final.pdf

[Sinz, 2002] Carsten Sinz Knowledge Compilation for Product Configuration in Configuration Workshop ECAI-2002 http://www.enstimac.fr/recherche/gind/ecai-2002-config-ws/Config-Wrksh-ecai02.pdf

[Syrjänen, 2000] Tommi Syrjänen *Optimizing Configurations* Configuration. Workshop ECAI-2000