Journal of Universal Computer Science, vol. 26, no. 3 (2020), 318-342
submitted: 15/1/19, accepted: 15/3/20, appeared: 28/3/20

Experimental Evaluation of Three Value Recommendation Methods in Interactive Configuration

Hélène Fargier
(IRIT, Université de Toulouse, CNRS, UT3, France
fargier@irit.fr)

Pierre-François Gimenez
(LAAS–CNRS, Université de Toulouse, CNRS, France
pierre-francois.gimenez@laas.fr)

Jérôme Mengin
(IRIT, Université de Toulouse, CNRS, UT3, France
mengin@irit.fr)

Abstract: The present work deals with the recommendation of values in interactive configuration, with no prior knowledge about the user, but given a list of products previously configured and bought by other users ("sales histories"). The basic idea is to recommend, for a given variable at a given step of the configuration process, a value that has been chosen by other users in a similar context, where the context is defined by the variables that have already been decided, and the values that the current user has chosen for these variables. From this point, two directions have been explored. The first one is to select a set of similar configurations in the sales history (typically, the k closest ones, using a distance measure) and to compute the best recommendation from this set – this is the line proposed by [Coster et al., 2002]. The second one, that we propose here, is to learn a model from the entire sample as representation of the users' preferences, and to use it to recommend a pertinent value; three families of models are experimented: the Bayesian networks, the naive Bayesian networks and the lexicographic preferences trees.

Key Words: product configuration, recommendation, machine learning, knowledge-based configuration
Category: Topic D.2.1 – Requirements/Specifications

1 Introduction

In on-line sale contexts, one of the main limiting factors is the difficulty for the user to find products that satisfy her preferences, and in an orthogonal way, the difficulty for the supplier to guide potential customers. This difficulty increases with the size of the e-catalog, which is typically large when the considered products are configurable. Such products are indeed defined by a finite set of components, options, or more generally by a set of variables (or "features"), whose values have to be chosen by the user. The search space is thus highly combinatorial. It is generally explored following a step-by-step interactive configuration session: at each step, the user freely selects a variable that has not been assigned yet, and chooses a value. Our issue is to provide such problems with a recommendation facility, by recommending, among the allowed values for the current variable, one which is most likely to suit the user.

The problem of providing the user with an item that fulfills her preferences has been widely studied, leading to the content-based and the collaborative filtering approaches, and every variation in between [Adomavicius and Tuzhilin, 2005, Ricci et al., 2011, Jannach et al., 2010]. However, these solutions cannot deal with configurable products, e.g. cars, computers, kitchens, etc. The first reason is that the number of possible products is huge – exponential in the number of configuration variables. For instance, in the car configuration problem described in [Astesana et al., 2010] the definition of "Traffic" delivery vans involves about 150 variables, and an e-catalog of 10^{21} feasible versions. The second reason is that the recommendation task considered in interactive configuration problem is quite different from the one addressed in classical product recommendation: the system is not asked to recommend a product (a car) but a value for the variable selected by the user. Note that we are not concerned here with the choice of the *variable* – this choice is under the control of the user, not under the one of the recommender system. It is worthwhile noticing that the fact that the variables are considered and assigned in a free order forbids the use of techniques based on decision trees. Finally, the third reason is that we cannot assume any prior knowledge about the user, nor about its buying habits – complex configurable products, like cars or kitchen, are not bought so often by one individual. So we have little information about similarity between users (upon which collaborative filtering approaches are based) or on the preferences of the current user (upon which content-based filtering approaches are based).

The present work[1] deals with the recommendation of values in interactive configuration, with no prior knowledge about the user, but given a list of products previously configured and bought by other users ("sales histories"). The basic idea is to recommend, for a given variable at a given step of the configuration process, a value that has been chosen by other users in a similar context, where the context is defined by the variables that have already been decided, and the values that the current user has chosen for these variables.

The recommendation of values, when any, is often limited to the proposition of a default value, generally the one advised by the seller in a static way or through a set of rules [Falkner et al., 2011]. Other approaches are based on similarity measures and propose to determine the k-nearest neighbouring configurations that are similar to the current set of user requirements [Coster et al., 2002]. The reader shall consult [Falkner et al., 2011] for a survey about recommenda-

[1] This article is an extended version of the preliminary work presented at [Fargier et al., 2016]; the experimental study has been completed by the evaluation of new models (k-LP-trees) and the investigation of more questions – clustering and influence of the constraints, for instance.

tion technologies for configurable product.

Several preferences models could be used for this task, e.g. ordinal models such as CP-nets (Conditional Preference nets [Boutilier et al., 2004]) and its variations, such as TCP-net [Brafman and Domshlak, 2002] and UCP-net [Boutilier et al., 2001], or numerical models like VCSP [Schiex et al., 1995] or GAI nets [Gonzales and Perny, 2004, Braziunas and Boutilier, 2005]. However, there are no learning algorithms to learn such models from a list of chosen items. One exception is the lexicographic preferences trees [Booth et al., 2010, Bräuning and Hüllermeyer, 2012], which can be learnt from a list of chosen items [Fargier et al., 2018].

The purpose of this article is to explore experimentally two directions and to point out the advantages and drawbacks of each solution. The first direction is to select a set of similar configurations in the sales history (typically, the k closest ones, using a distance measure) and to compute the best recommendation from this set – this is the line proposed by [Coster et al., 2002]. The second one is to learn, from the entire sample, a model of the users' preferences, e.g. a Bayesian net, a naive Bayesian network or a lexicographic preference tree, and to use it to propose a pertinent value.

The paper is structured as follows: the basic notations are presented in Section 2. The next three sections present the three families of approaches that we explore: Bayesian nets and naive Bayesian networks in Section 3, lexicographic preferences tree in Section 4 and k-closest neighbors in Section 5. They are experimentally compared and discussed in Section 6.

2 Background and notations

A configuration problem is defined by a set $\mathcal{X}$ of n discrete variables, each variable X taking its value in a finite domain $\underline{X}$. A complete configuration is thus a tuple $\mathbf{o} \in \prod_{X \in \mathcal{X}} \underline{X}$; we denote by $\underline{\mathcal{X}}$ the set of all of them.

If $\mathbf{W}$ is a tuple of variables, $\underline{\mathbf{W}}$ denotes the set of partial configurations $\prod_{X \in \mathbf{W}} \underline{X}$; we will often denote such a partial configuration by the corresponding lower case letter $\mathbf{w}$. Also, if $\mathbf{W}$ and $\mathbf{V}$ are two sets of variables, and if $\mathbf{w} \in \underline{\mathbf{W}}$, then $\mathbf{w}[\mathbf{V}]$ is the projection of $\mathbf{w}$ onto $\mathbf{V} \cap \mathbf{W}$. Furthermore, if $\mathbf{w} \in \underline{\mathbf{W}}$, $\mathbf{w}$ is said to be compatible with $\mathbf{v}$ if $\mathbf{w}[\mathbf{V} \cap \mathbf{W}] = \mathbf{v}[\mathbf{V} \cap \mathbf{W}]$; in this case we write $\mathbf{w} \sim \mathbf{v}$. Finally, in the case where $\mathbf{w}$ and $\mathbf{v}$ are compatible, we denote by $\mathbf{w}.\mathbf{v}$ the tuple that extends $\mathbf{w}$ with values of $\mathbf{v}$ for variables in $\mathbf{V} \setminus \mathbf{W}$ (equivalently, $\mathbf{w}.\mathbf{v}$ extends $\mathbf{v}$ with values of $\mathbf{w}$ for variables in $\mathbf{W} \setminus \mathbf{V}$).

Not all combinations represent feasible products, because of some possible feasibility or marketing constraints; for example, a sunroof cannot be installed on a cabriolet. In practice, the set of the feasible products is still a huge set.

In interactive configuration problems, the user builds the product she is interested in through a variable by variable interaction. At each step, let **Assigned** be the set of variables for which she has already chosen values and $\mathbf{u}$ be the tuple of values assigned to these variables; then the user freely selects the next variable to be assigned. We denote Next this variable. The system then has to:

- Compute the set of admissible values for Next: it is the set of values $v \in \underline{\text{Next}}$ such that there is at least one feasible product (i.e. satisfying the constraints) $\mathbf{o}$ compatible with this combination of values, that is $\mathbf{o}[\text{Next}] = v$ and $\mathbf{o}[\mathbf{Assigned}] = \mathbf{u}$.
- Propose a recommended value for Next, chosen among the admissible values.

The first problem is not the focus of this article: we assume that we are able to compute, for each variable, the values in its domain that are coherent with the current configuration of the user. Various techniques such as constraints propagation, global inverse consistency [Bessiere et al., 2013] or compilation [Pargamin, 2002, Amilhastre et al., 2002, Hadzic et al., 2007] can be used. We focus here on the second problem, which is that of producing good recommendations and to propose the most suited value for Next that is coherent with the current configuration.

In the context considered in this paper, *sales histories* are available, on which the system can rely to base its recommendation. Formally, a sales history is a multiset $\mathcal{H} \subseteq \underline{\mathcal{X}}$ of complete configurations that correspond to products that have been bought by past users. In the sequel, for a partial configuration $\mathbf{u}$, $\#(\mathbf{u})$ will denote the number of configurations in $\mathcal{H}$ compatible with $\mathbf{u}$.

Remark that we do not rely on any prior about the user's preferences, but such priors could be added in our framework by adding a preliminary phase just before the user chooses the first variable to assign. During this phase, the on-line configurator would automatically assign some variables such as the country of the user (obtained from her IP address) or some demographics data (obtained from specialized tools such as Google Analytics). However, these variables would be useful only if they appear in the constraints or the sales history.

3 Recommendation with Bayesian networks

Users have different preferences, depending on their tastes and their environments, which make them prefer different products – hence a large variety of products in the histories. We do not have any information about their taste nor about their environment. Instead, it can be assumed that there is a ground probability distribution p over the set of complete configurations (i.e. the space of all feasible products), indicating how likely it is that each item is the one that the current user prefers. This probability may depend on her personality and on her current environment, but it can be assumed that the sales history gives a good approximation of that probability distribution: the configured products eventually bought by the past users are the one they prefer.

Therefore, if Next is the next variable to be assigned a value, and if $\mathbf{u}$ is the vector of values that have been chosen for the variables already decided, we propose to estimate, for each possible value v for Next, the marginal conditional probability $p(\text{Next} = v \mid \textbf{Assigned} = \mathbf{u})$: it is the marginal probability that Next has value v in the most preferred product of the current user, given the choices that she made so far; hence we can recommend the most probable value:

$$\underset{v \in \underline{\text{Next}}}{\text{argmax}}\ p(\text{Next} = v \mid \textbf{Assigned} = \mathbf{u}).$$

The idea here is that the sales history is a sample of $\underline{\mathcal{X}}$ according to the unknown distribution p, that can be used to estimate probabilities. A first, naive method to compute $p(v \mid \mathbf{u})$ would be to count the proportion of v within the sold products that verify $\mathbf{u}$. Even if this idea works for small $\mathbf{u}$'s, after a few steps the number of products that verify $\mathbf{u}$ would be too low and the computations would not be reliable enough (and even impossible when no product in the history verifies $\mathbf{u}$). Hence the idea of learning, off-line, a Bayesian network from the dataset and to use it, on-line, during the step-by-step configuration session: the user defines a partial configuration $\mathbf{u}$ by assigning some variables and chooses a variable Next; the recommendation task consists in computing the marginal $p(\text{Next} \mid \textbf{Assigned} = \mathbf{u})$ and recommending the user with the value of Next that maximizes this probability.

3.1 Bayesian networks

A Bayesian network, introduced by [Pearl, 1989], over a set of variables $\mathcal{X}$ is a pair $(\mathcal{G}, \Theta)$ where:

- $\mathcal{G}$ is a directed acyclic graph (DAG) over a set of variables $\mathcal{X}$. In the following we will denote by $\mathbf{Pa}_{\mathcal{N}}(X)$ the parents of X in $\mathcal{G}$.
- Θ a set of conditional probability distributions. For each variable $X \in \mathcal{X}$ is the conditional probability distribution of X given a set of values of its parents, i.e. for any $x \in \underline{X}$, $\mathbf{u} \in \underline{\mathbf{Pa}_{\mathcal{N}}(X)}$, $\Theta(x, \mathbf{u})$ is the conditional probability of x given $\mathbf{u}$.

A Bayesian network $\mathcal{N}$ uniquely defines a probability distribution $p_{\mathcal{N}}$ over $\mathcal{X}$: the probability of a complete configuration $\mathbf{o} \in \underline{\mathcal{X}}$ is

$$p_{\mathcal{N}}(\mathbf{o}) = \prod_{X \in \mathcal{X}} \Theta(\mathbf{o}[X] \mid \mathbf{o}[\mathbf{Pa}_{\mathcal{N}}(X)])$$

In the sequel, we will often omit the subscript $\mathcal{N}$ when there is no ambiguity.

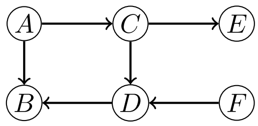


Figure 1: A Bayesian network

For example, consider the Bayesian network depicted in Figure 1 The probability of a configuration $abcdef$ can be computed as:

$$p_{\mathcal{N}}(abcdef) = \Theta(a)\Theta(c \mid a)\Theta(e \mid c)\Theta(f)\Theta(d \mid cf)\Theta(b \mid ad)$$

3.2 Learning a Bayesian network

The learning of Bayesian networks from data proceeds in two steps: finding the structure of the network, i.e. of the DAG underlying the Bayesian network and then its parameters, i.e. the conditional probabilities tables. Both aim at maximizing likelihood estimates, i.e. the probability of observing the given set.

Learning the most probable a posteriori Bayesian network from data is an NP-hard problem [Chickering, 1995]. There are two main families of approaches in structure learning: the score-based ones and the constraint-based ones. The former search for a network that maximizes a score pointing out to what extend the network fits the data [Cooper and Herskovits, 1992]. The second family of approaches looks for conditional independencies, through independence tests, assuming the faithfulness of the network to learn. An example is the algorithm PC [Spirtes et al., 2000]. Finally, hybrid method exist, such as MMHC [Tsamardinos et al., 2006], that learns the undirected structure of the network with a constraint-based approach (named MMPC) and then orients the edge of the DAG with a score-based method.

3.3 Computing marginals

The computation of the *posterior* marginal probability $p(\text{Next} \mid \mathbf{Assigned})$ is a classical task of Bayesian inference. In general, it is broken down into computations of two separate *prior* marginals, since, by definition $p(\text{Next} \mid \mathbf{Assigned}) = p(\text{Next} \wedge \mathbf{Assigned})/p(\mathbf{Assigned})$.

Recall that, for a given configuration $\mathbf{o}$, $p(\mathbf{o})$ is the product of local, conditional probabilities of the network that correspond to $\mathbf{o}$. Then, given a variable $\mathbf{X} \subseteq \mathcal{X}$ and a partial configuration $\mathbf{x} \in \underline{X}$, the marginal probability $p(\mathbf{x})$ is the sum of the probabilities of the complete configurations that extend $\mathbf{x}$:

$$p(\mathbf{x}) = \sum_{\substack{\mathbf{w} \in \underline{\mathcal{X}} \\ \mathbf{w}[\mathbf{X}] = \mathbf{x}}} \prod_{Y \in \mathcal{X}} \Theta(\mathbf{w}[Y] \mid \mathbf{w}[\mathbf{Pa}_{\mathcal{N}}(Y)]).$$

Computing such prior marginals is known to be an NP-hard problem when p is represented by a Bayesian network [Dagum and Luby, 1993]. Indeed, the size of the formula can grow exponentially fast with the number of variables. There are exact inference algorithms, such as the jointree algorithm proposed by [Lauritzen and Spiegelhalter, 1988], that work by breaking down this sum-product formula, into sub-sums and sub-products. These algorithms have a worst-case time complexity exponential with respect to the tree width of the network. Even if they target a NP-hard task, they are efficient enough on real world datasets to allow an on-line use. There also exists approximate inference algorithms, based on belief propagation (for example [Kim and Pearl, 1983]) or stochastic sampling (such as [Lemmer and Kanal, 2014]).

3.4 Recommendation using Naive Bayesian Networks

The naive Bayesian networks are a subset of the Bayesian network, where one central variable (the one on which inference is to be made) is targeted and the others are assumed independent from each other conditionally to this variable of interest. A naive Bayesian network is therefore a Bayesian network whose structure is a tree, and where the variable of interest (in our case, Next) is the parent of every other variables (in our case, the variables in **Assigned**). For any value v of Next and any assignment $\mathbf{u}$ of **Assigned**, we know that $P(v \mid \mathbf{u})$ is proportional to $P(v\mathbf{u})$. We can recommend the value v that maximizes $P(v\mathbf{u})$:

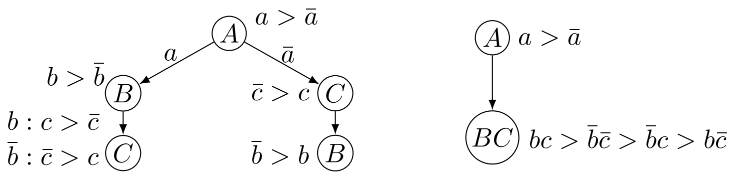
$$\operatorname*{argmax}_{v \in \underline{\mathrm{Next}}} P(v \mid \mathbf{u}) = \operatorname*{argmax}_{v \in \underline{\mathrm{Next}}} P(v\mathbf{u}) = \operatorname*{argmax}_{v \in \underline{\mathrm{Next}}} \left(P(v) \times \prod_{X \in \mathbf{Assigned}} P(\mathbf{u}[X] \mid v) \right)$$

Since the variable we are recommending a value for, Next depends on the configuration process, we would need a naive Bayesian network for every variable: to recommend a value for Next, we use the naive Bayesian network for which Next is the variable of interest. The computation of the networks is preprocessed: all the prior distributions $P(X)$ and all the conditional tables $P(Y \mid X)$ (i.e., potentially all the naive Bayesian networks) are computed off line, before the configuration process, from the sample, using Laplace smoothing:

$$\begin{cases} P(X = x) = \frac{\#(x)}{|\mathcal{H}|} \text{ for each } X \in \mathcal{X} \\ P(Y = y \mid X = x) = \frac{\#(x.y)+1}{\#(x)+|\underline{Y}|} \text{ for each pair } X, Y \in \mathcal{X} \end{cases}$$

The (pre)computation of n prior tables and n^2 conditional probability tables are thus sufficient to make a prediction for any variable at any moment.

The strong assumptions of the multiple naive Bayesian networks are mutually contradictory: when we want to recommend a value for Next, we assume that all the variables in **Assigned** are conditionally independent given Next. In spite of



$$abc \succ ab\bar{c} \succ a\bar{b}\bar{c} \succ a\bar{b}c \succ \bar{a}\bar{b}\bar{c} \succ \bar{a}b\bar{c} \succ \bar{a}\bar{b}c \succ \bar{a}bc$$

(a) A LP-tree.

$$abc \succ a\bar{b}\bar{c} \succ a\bar{b}c \succ ab\bar{c} \succ \bar{a}bc \succ \bar{a}\bar{b}\bar{c} \succ \bar{a}\bar{b}c \succ \bar{a}b\bar{c}$$

(b) A 2-LP-tree

Figure 2: Different LP-trees and the preference relations they induce

this naive and strong assumption, they are precise enough for some applications. Among their qualities, they are easy to learn and easily scalable, requiring a number of parameters quadratic in the number of variables.

4 LP-trees and k-LP-trees

Lexicographic preference trees [Booth et al., 2010], or LP-trees for short, are ordinal models based on the lexicographic preferences that rely of variables importance: when comparing two items $\mathbf{o}$ and $\mathbf{o}'$ of $\mathcal{X}$, the most important variable is considered; if $\mathbf{o}$ and $\mathbf{o}'$ have different values for that variable, then the one with the preferred value is deemed preferable to the other; otherwise one looks at the next most important variable, and so on.

A LP-tree is composed of two parts: a rooted tree indicating the relative importance of the variables, and tables indicating how to compare items that agree on some variables. Each node of the importance tree is labelled with a variable $X \in \mathcal{X}$, and is either a leaf of the tree, or has one single, unlabelled outgoing edge, or has $|\underline{X}|$ outgoing edges, each one being labelled with one of these values. No variable can appear twice in a branch. Moreover, one conditional preference table is associated to each node N of the tree. This table contains total orders over the domain of the variable labelling N and may depend on the values of variables that are at a node above N with a labelled outgoing edge. An example of a LP-tree is depicted in Figure 2a.

[Bräuning and Hüllermeyer, 2012, Bräuning et al., 2017] extend the expressiveness of LP-trees by allowing a node to be labelled with a set of variables, considered as a single high-dimensional variable: the rules in the conditional preference table of the node define orders on the Cartesian product of the domains of its variables. Generally, we restrict this expressivity by fixing the maximum

Algorithm 1: Recommendation of a value using a k-LP-trees

Input: An k-LP-tree $\mathcal{L}$, $\mathbf{u}$ a partial configuration, Next a variable
Output: Recommend a value for Next given $\mathbf{u}$
Algorithm `RecommendFromBestExtension(`$\mathcal{L}$, $\mathbf{u}$`)`

- $\mathbf{o} \leftarrow \mathbf{u}$
- $N \leftarrow$ the root of $\mathcal{L}$
- **while** $\mathbf{o}$[Next] *is not assigned* **do**
  - $\mathbf{X} \leftarrow$ the label of N
  - $\mathbf{o}[\mathbf{X}] \leftarrow$ the most preferred value of $\mathbf{X}$ compatible with $\mathbf{u}$ according to the conditional preference table of N
  - $N \leftarrow$ the child of N compatible with $\mathbf{o}[\mathbf{X}]$
- **return** $\mathbf{o}$[Next]

number of variables labelling a node. The trees whose nodes are labelled by at most k variables are denoted k-LP-tree. Figure 2b shows a 2-LP-tree whose preference relation cannot be expressed with a regular LP-tree.

[Fargier et al., 2018] proposes an algorithm to learn a k-LP-tree from a sales history. The learning algorithm has a temporal complexity in $O(n^{k+1}k^2|\mathcal{H}|^2)$; for this reason, we will limit the value of the parameter k to 2 or 3.

So, in order to recommend a value for Next given $\mathbf{u}$, we propose to recommend the value of Next of the most preferred item $\mathbf{o}$ that extends $\mathbf{u}$. This recommendation can be done with a top-down traversal of the k-LP-tree as described in Algorithm 1. The research of the most preferred extension $\mathbf{o}$ of $\mathbf{u}$ is done by choosing, at each depth of the k-LP-tree, the best values according to the conditional preference table; since we are only interested in the value $\mathbf{o}$[Next] (the recommended value), we stop the search as soon as this value is obtained.

5 Methods based on *k*-nearest neighbors

A third family of recommending algorithms is proposed in [Coster et al., 2002], based on the selection of a neighborhood. Rather than computing the preference from the entire sample, the system focuses on sold configurations that are similar to the present one – i.e. the k nearest neighbors. All the methods proposed in [Coster et al., 2002] are based on the Hamming distance; namely, given an assignment $\mathbf{u}$ of **Assigned**, and a complete configuration $\mathbf{w}$, $d_{\mathbf{Assigned}}(\mathbf{u}, \mathbf{w})$ counts the number of assigned variables on which the two configurations disagree:

$$d_{\mathbf{Assigned}}(\mathbf{u}, \mathbf{w}) = |\,\{X \in \mathbf{Assigned} \mid \mathbf{u}[X] \neq \mathbf{w}[X]\}\,| \tag{1}$$

At each step, these methods select the set $N(k, \mathbf{u})$ of the k-nearest neighbors of the current $\mathbf{u}$, and compute the recommendation on this basis.

5.1 Weighted Majority Voter

The simplest algorithm is the Weighted Majority Voter, which predicts the value of Next on the basis of a weighted majority vote of the k nearest neighbors. The weight of a configuration $\mathbf{w}$ in $N(k, \mathbf{u})$ is set equal to the degree of similarity between this configuration and the current one, $\mathbf{u}$, i.e. the number of variables that are given the same value by both:

$$weight(\mathbf{u}, \mathbf{w}) = |\,\{X \in \mathbf{Assigned} \mid \mathbf{u}[X] = \mathbf{w}[X]\}\,|$$

The recommended value for Next is chosen among the ones that are authorized by the constraints by maximizing:

$$vote(v) = \sum_{\mathbf{w} \in N(k,\mathbf{u}) | \mathbf{w}[\text{Next}]=v} weight(\mathbf{u}, \mathbf{w})$$

5.2 Most Popular Choice

Most Popular Choice predicts the most popular (actually, the most probable) extension of the current configuration, $\mathbf{u}$, from the knowledge of the neighbors and recommends the value supported by this configuration. It holds that, for any full configuration $\mathbf{uw}$ that extends $\mathbf{u}$, $P(\mathbf{uw}) = P(\mathbf{u} \mid \mathbf{w}) \cdot P(\mathbf{w})$. [Coster et al., 2002] makes the assumption that the variables that have not been assigned are mutually independent, and that the ones that are assigned are independent from one another given $\mathbf{w}$. Hence we have:

$$P(\mathbf{uw}) = \prod_{X \in \mathbf{Assigned}} P(\mathbf{u}[X] \mid \mathbf{w}) \cdot \prod_{X \in \mathcal{X} \setminus \mathbf{Assigned}} P(\mathbf{w}[X])$$

The probabilities are estimated from the k nearest neighbors of $\mathbf{u}$:

- for $X \in \mathcal{X} \setminus \mathbf{Assigned}$ and $x \in \underline{X}$:

$$P(x) = \frac{1}{k} |\{\mathbf{w}' \in N(k, \mathbf{u}), \mathbf{w}'[X] = x\}|$$

- for $X \in \mathbf{Assigned}$ and $x \in \underline{X}$, let $N(k, \mathbf{u}, \mathbf{w})$ be the set of neighbors of $\mathbf{u}$ that agree with $\mathbf{w}$ on $\mathbf{Assigned}$:

$$N(k, \mathbf{u}, \mathbf{w}) = \{\mathbf{w}' \in N(k, \mathbf{u}), \mathbf{w}'[\mathbf{Assigned}] = \mathbf{w}[\mathbf{Assigned}]\}$$

  Then $P(x \mid \mathbf{w})$ is the fraction of $N(k, \mathbf{u}, \mathbf{w})$ that has value x, with Laplace smoothing since $N(k, \mathbf{u}, \mathbf{w})$ may be empty:

$$P(x \mid \mathbf{w}) = \frac{|\{\mathbf{w}' \in N(k, \mathbf{u}, \mathbf{w}) | \mathbf{w}'[X] = x\}| + 1}{|N(k, \mathbf{u}, \mathbf{w})| + |\underline{X}|}$$

The value recommended for Next is the one prescribed by the $\mathbf{w} \in N(k, \mathbf{u})$ that maximizes $P(\mathbf{uw}[\mathcal{X} \setminus \mathbf{Assigned}])$. The drawback is that this method does not guarantee that the value computed is compatible with $\mathbf{u}$ according to the constraint.

5.3 Naive Bayes Voter

The Naive Bayes Voter is similar to the Naive Bayes method proposed in Section 3.4, with the difference that it uses the k nearest neighbors of $\mathbf{u}$ to build a naive Bayes network. Since these neighbors depend on $\mathbf{u}$, it is not possible to preprocess the computation of the probability table – this approach is much slower than the classical naive Bayes as the experiments point out.

The recommended value for Next is chosen among the ones that are authorized by the constraints by maximizing $P(v \mid \mathbf{u}) \propto p(v) \prod_{X \in Assigned} p(\mathbf{u}[X] \mid v)$, where:

- $p(v) = \frac{1}{k}|\{\mathbf{w} \in N(k, \mathbf{u})|\mathbf{w}[\text{Next}] = v\}|$
- for every $X \in \mathbf{Assigned}$ and every $v \in \underline{\text{Next}}$, let $N(k, \mathbf{u}, v)$ be the set of neighbors of $\mathbf{u}$ that have value v for Next, then[2]:

$$p(\mathbf{u}[X] \mid v) = \frac{|\{\mathbf{w} \in N(k, \mathbf{u}, v)|\mathbf{w}[X] = \mathbf{u}[X]\}| + 1}{|N(k, \mathbf{u}, v)| + |\underline{X}|}$$

6 Experimental evaluation

The approaches proposed in this paper have been tested on a case study of three sales histories provided by Renault, a French automobile manufacturer[3]. These datasets, named *Renault-44*, *Renault-48* and *Renault-87*, are genuine sales histories – each of them corresponds to a configurable car, and each example in the set corresponds to a configuration of this car which has been sold.

Most of the variables are binary, but not all of them. To each dataset is associated a Constraint Satisfaction Problem (CSP) containing the technical, legal and business constraints. However, since these constraints change over time (a new ecological law can forbid some configurations; an option can be included as standard), a non-negligible part of these dataset does not satisfy the constraints provided. More precisely:

- dataset *Renault-44* has 44 variables and 14786 examples including 8252 examples consistent with the constraints.

[2] We slightly modified the formula of [Coster et al., 2002], replacing the term k at the denominator by the term $|\underline{X}|$; otherwise, the conditional probabilities are not correctly normalized and do not sum to 1.

[3] available at `http://www.irit.fr/~Helene.Fargier/BR4CP/benches.html`

- dataset *Renault-48* has 48 variables and 27088 examples including 710 examples consistent with the constraints.
- dataset *Renault-87* has 87 variables and 17715 examples including 8335 examples consistent with the constraints.

As explained in section 2, we use an external tool to process the constraints and compute, at each step, the admissible values for each variable. Thus, the recommenders don't need to be modified to incorporate the constraints: they just need to be able to recommend a value among a set of admissible values.

6.1 Experimental protocol

We use a classical ten-fold cross-validation: each dataset is cut by ten, an algorithm learns with nine tenth (which constitute the sales history) and is tested with the last tenth (which can be viewed as a set of on-line configuration sessions). The protocol is described in Algorithm 2. Each test is a simulation of a configuration session, i.e. a sequence of variable-value assignments. In real life, a genuine variable ordering was used by the user for her configuration session and the different sessions generally obey different variable orderings. Unfortunately, the histories provided by Renault describe sales histories only, i.e. sold products, and not the ordered sequence of variable-value assignment in each session. That is why we generate a session *session* for each product $\mathbf{o}$ in the test set by randomly ordering the variables of $\mathcal{X}$.

At the beginning of a simulation, the partial ongoing configuration $\mathbf{u}$ is empty. Then, the next variable Next in *session* is considered and the set of admissible values (i.e. the set of values that can lead to a feasible complete configuration) of Next is computed. If there is only one admissible value, the recommendation is trivial and will not be taken into account for the success rate. Otherwise, the recommender is asked for a recommendation for Next, say r: r may be equal to $\mathbf{o}[\text{Next}]$ or may be different if the recommender considers the value r more suited than $\mathbf{o}[\text{Next}]$. We consider a recommendation as correct if the recommended value is the same as the value really chosen in the product. Any other value is considered as incorrect. Finally $\mathbf{u}[\text{Next}]$, the partial configuration, is set to $\mathbf{o}[\text{Next}]$ and the process is continued for the next variable Next.

The recommendation algorithms are evaluated by (i) the time needed for computing the recommendations and (ii) their success rate, obtained by counting the number of correct and incorrect recommendations, discarding the trivial recommendations.

6.1.1 Oracle

In order to easily interpret the results, we propose to compute an upper limit on the success rate. If there were an algorithm that already knows the testing

Algorithm 2: Protocol for evaluating value recommendation in interactive configuration

Input: A recommendation algorithm $\mathcal{A}$ and the test set *Htest*
Output: The success rate
main:
$\quad$ *success* $\leftarrow 0$
$\quad$ *error* $\leftarrow 0$
$\quad$ **for each** $\mathbf{o} \in$ *Htest* **do**
$\quad\quad$ *session* $\leftarrow$ a randomly drawn sequence of the variables $\mathcal{X}$
$\quad\quad$ $\mathbf{u} \leftarrow$ empty tuple
$\quad\quad$ **for each** Next $\in$ *session* **do**
$\quad\quad\quad$ *admissible* $\leftarrow$ the set of admissible values of Next given $\mathbf{u}$
$\quad\quad\quad$ **if** $|admissible| \neq 1$ **then**
$\quad\quad\quad\quad$ $r \leftarrow$ recommended value by $\mathcal{A}$ for Next given $\mathbf{u}$ among *admissible*
$\quad\quad\quad\quad$ **if** $r = \mathbf{o}[\text{Next}]$ **then** increment *success* by 1
$\quad\quad\quad\quad$ **else** increment *error* by 1
$\quad\quad\quad$ $\mathbf{u}[\text{Next}] \leftarrow \mathbf{o}[\text{Next}]$
$\quad$ **return** $success/(success + error)$

set, it could use the probability distribution estimated from this testing set. This hypothetical algorithm could attain a success rate that is not reachable by algorithms that have only access to the training set. In particular, this algorithm could recommend for the variable Next, given the assigned values $\mathbf{u}$, the most probable value of Next in the subset of products, in the test set, that respect $\mathbf{u}$. More precisely, for any x in the domain of Next, it would estimate $p(x \mid \mathbf{u})$ as $\#(\mathbf{u}x)/\#(\mathbf{u})$. Notice that $\#(\mathbf{u})$ is never equal to zero, since the test set contains at least one product consistent with $\mathbf{u}$: the one corresponding to the current session. It can be interpreted as an algorithm overfitted to the test set.

We call this algorithm "Oracle". This algorithm maximizes the probability of success measured in our protocol. This follows from a classical result of statistics: with a 0-1 loss function ("success" or "failure" of the recommendation in our case), the estimator that minimizes the loss is the maximum a posteriori estimator, that recommends the most probable value given the partial configuration. This is exactly what the Oracle does.

6.1.2 Experiments

The R package *bnlearn* was used to learn the Bayesian networks – more precisely, we used Hill Climbing (HC) to learn the two datasets of about 50 variables

(*Renault-44* and *Renault-48*) and MMHC to learn the bigger dataset *Renault-87*. The average number of parents of a node in the obtained Bayesian network is about 1.17, 1.02 and 0.98 – for *Renault-44*, *Renault-48* and *Renault-87*, respectively. As to Bayesian inference, we used the jointree algorithm provided by the library Jayes. We implemented the Naive Bayes approach, the k-LP-trees approach and the algorithms based on the k-nearest neighbors. We used the CSP compiler and solver SALADD [Schmidt, 2015] to compute the feasible configurations. The experiments have been made on a computer with a quad-core processor i5-3570 at 3.4Ghz, using a single core. Recommendation algorithms are implemented in Java[4]. We were interested in the following questions:

– Which method is the most efficient in terms of error rate and recommendation time?

– How does the size of the learning dataset influence the success rate?

– What is the influence of the feasibility constraints on the success rate?

Before entering these questions, we need to study two preliminary parameters, that may influence the efficiency of the methods:

– Concerning the methods based on k-nearest neighbors, which values of k minimize the error rate?

– Does the clustering of the learning set enhance the success rate?

6.2 Results

6.2.1 Neighborhood size

The algorithms based on the selection of k nearest neighbors [Coster et al., 2002] depends on the parameter k. The issue of choosing a good value for k is not tackled in the original paper but can have a direct consequence to the recommenders' performance. Indeed, the recommendation time of a neighbor-based recommender is the sum of the time used to find the k nearest neighbors and the time of the vote itself (Weighted Majority Voter, Most Popular Choice or Naive Bayes Voter). The algorithm implemented for this study finds the k nearest neighbors in a set $\mathcal{H}$ in $O(k|\mathcal{H}|)$. Remark that we cannot use traditional way of preprocessing the neighbors search since the neighborhood depends on the distance $d_{\mathbf{Assigned}}$, which depends on the set of assigned variables **Assigned** that contains 2^n possibilities.

Our empirical analysis (see Figure 3) reports the average error rate and time of the Naive Bayes Voter recommender on *Renault-44* and *Renault-48* w.r.t. k,

[4] The source code is available at `https://github.com/PFGimenez/PhD`

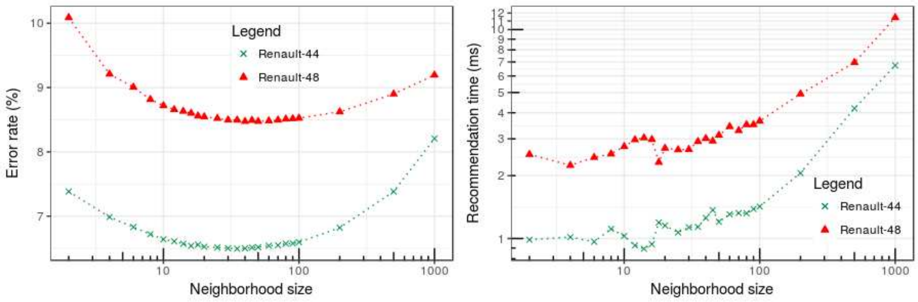


Figure 3: Average error rate (left) and recommendation time (right) of Naive Bayes Voter on *Renault-44* and *Renault-48* w.r.t. k, the number of neighbors

the number of neighbors. The results are similar for Weighted Majority Voter and Most Popular Choice, and on the other datasets.

We can clearly see a bell curve for the error rate: too few neighbors and the Naive Bayes does not have enough data to perform an accurate prediction. Too much neighbors and the neighborhood starts containing examples that may be too different from current configuration. The lowest error rate is achieved between $k = 20$ and $k = 50$. For the rest of the experiments, we set $k = 20$, since it is suited to the different datasets.

6.2.2 Is it worth clustering the sales history?

A well-known technique for increasing accuracy in machine learning is clustering: segregate data into homogeneous groups, called clusters, and learn the preference for each cluster. Clustering has been previously used successfully in collaborative filtering recommendations [Ungar and Foster, 1998]. That's why we chose to empirically verify whether, in the context of interactive configuration, clustering may enhance recommendations or not.

We based our clustering on the famous k-means method. Once the c clusters have been learned, one recommender is set up for each cluster. This means that there will be c recommenders, each one being learned with configurations in the training set belonging to its cluster. When the configurator must provide a recommendation for a partial configuration $\mathbf{u}$, the most appropriate recommender is used. During a configuration session, recommenders from different clusters can thus be used at different steps of the configuration. For neighborhood-based algorithms and the LP-tree algorithm, the most appropriate recommender is the recommender whose cluster center is the closest to $\mathbf{u}$ (according to the Hamming distance). For the Bayesian networks, since model represents a probability distribution, the most appropriate recommender is the recommender that maximizes the probability of $\mathbf{u}$.

We experimented various number of clusters, from one to three. The recommendation error rate on *Renault-44* w.r.t. the number of clusters is shown in Table 1. The tendencies are the same for *Renault-48* and *Renault-87*.

Cluster number	1 cluster	2 clusters	3 clusters
Naive Bayes Voter	6.52%	8.71%	9.74%
Bayesian Network	6.60%	8.55%	9.59%
Naive Bayes Network	8.59%	11.05%	12.65%
3-LP-tree	16.15%	16.19%	15.08%

Table 1: The error rate of three recommenders w.r.t. the number of clusters on *Renault-44*

We can see that clustering is not worthwhile for the neighborhood-based methods, the Bayesian networks and the naive Bayes network, but decreases the error rate for the 3-LP-tree.

This success rate loss of the neighborhood-based algorithms due to the clustering can be explained as follows. These algorithms already perform an on-line clustering in the form of a selection of the relevant sold items. It is important to remember that these recommenders look for the closest neighbors of the current, partial configuration: the distance $d_{\mathbf{Assigned}}$ relies solely on the variables assigned in the partial configuration, and ignores the values of the other variables (see definition (1)). However, the clustering distance, that we could denote $d_{\mathcal{X}}$, takes all the variables into account: the set of similar configurations for a partial configuration $\mathbf{u}$ are generally different from the set of similar configurations for an extension $\mathbf{uv}$. Since the problem comes from the clustering itself, the same reasoning can be applied to Bayesian networks. That's why, as we can see on the experimental result, the greater the number of assigned variables, the smaller the success rate loss for both algorithm: the loss is close to zero for the last recommendation on nearly complete configuration.

The success rate increase of the 3-LP-tree results certainly from the limited expressivity of the k-LP-tree. Using several k-LP-trees allows to represent more closely the true user preferences. Remark however that the success rate attains a maximum for a certain number of clusters; with more clusters, there won't be enough data in each clusters to learn reliably a k-LP-tree.

Clustering has nonetheless an advantage for neighborhood-based methods and Bayesian networks: Figure 4 shows that, for both the Naive Bayer voter and the Bayesian network, the usage of clusters reduces the recommendation time. Concerning Bayesian network, we believe that the reduced training set decreased the models complexity and, therefore, the inference time.

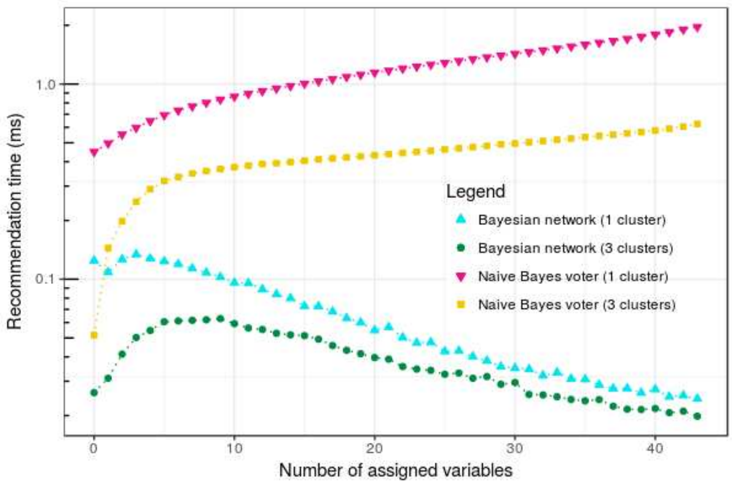


Figure 4: Average recommendation time on *Renault-44* for Bayesian Network and Naive Bayes voter with one or three clusters

In the next experiments, we won't use any clustering for Naive Bayer voter and Bayesian network and three clusters for k-LP-trees.

6.2.3 Success rate and temporal efficiency

Figure 5 shows the success rate of the pure Bayesian Network-based approaches (Bayesian network and Naive Bayes), the methods based on k closest neighbors and the k-LP-tree (with three clusters). The oracle is given as an ideal line.

It appears that the Naive Bayesian network, which makes very strong independence assumptions, has a low success rate (this error rate is bad also on classical Bayesian networks benchmarks). This is not surprising, since the variables are not independent from one another, at least because of the constraints. The independence assumptions at work in the methods based on the k closest neighbors are in a sense less drastic, since the distance used to select the neighborhood implicitly captures some dependencies.

The k-LP-trees also have a low success rate, especially on *Renault-44*, probably because of their limited expressivity. Furthermore, since they are ordinal models, they cannot evaluate the most probable value a posteriori but must first compute the preferred product and then recommend the value of the variable of interest in this preferred product.

Three methods have very good results: Classical Bayes Net, Naive Bayes Voter and Most Popular Choice. Their success rate is very good (only a few points from the Oracle). The gap with the Oracle gets larger when the number

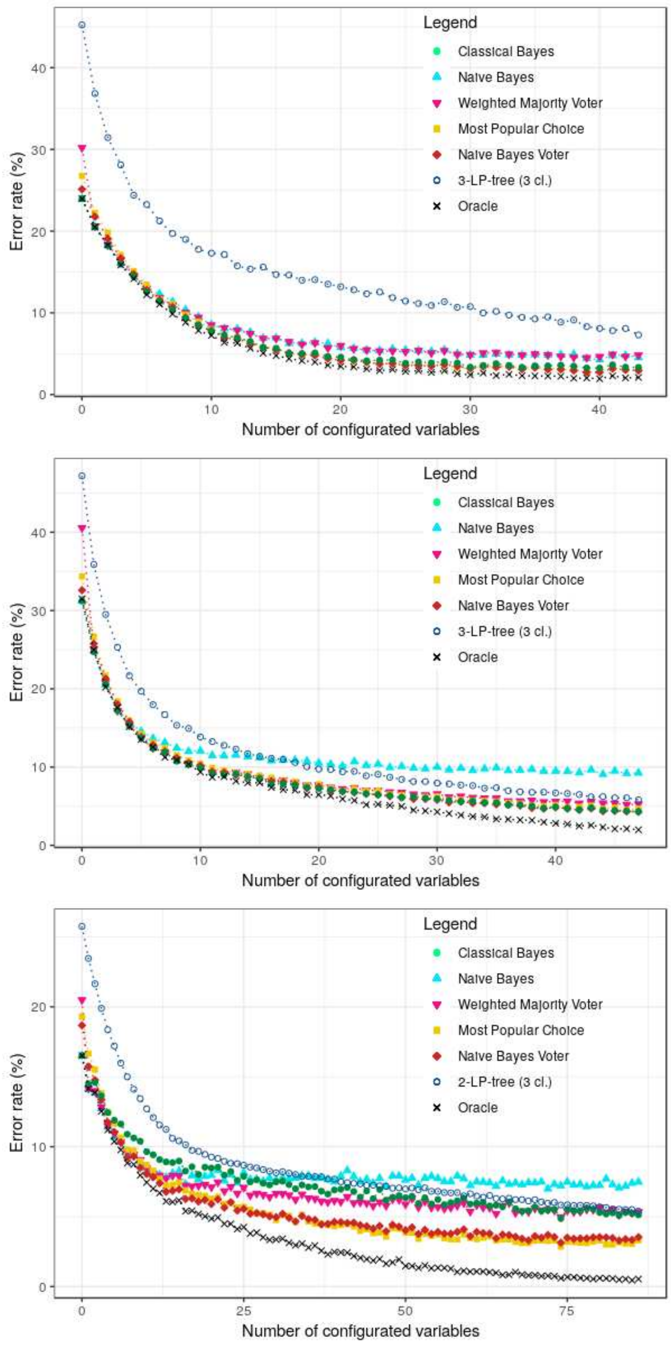


Figure 5: Average error rate on datasets *Renault-44* (top) *Renault-48* (middle) and *Renault-87* (bottom)

of assigned variables increases: the Oracle's performance becomes less and less attainable. This phenomena is especially visible with the dataset *Renault-87*, because it has more variables that *Renault-44* or *Renault-48*.

As presented in the introduction, a simple recommendation method uses a default value for each variable. We can estimate its maximum success rate from the Oracle results. Indeed, since default values don't depend on the values of the configured variables, its success rate is bounded by the success rate of the Oracle with no configured variables (for example, this error rate is 24% on *Renault-44*). We conclude that the success rate of this method is far below the success rate of the other methods but this method could usefully be used in conjunction with k-LP-trees when few variables are configured.

The CPU time (see Figure 6) clearly breaks the set of algorithms in three groups: the ones that learn the dependencies off-line from the entire dataset and have a linear recommendation time (k-LP-tree and Naive Bayesian Network), the one that learn the dependencies off-line from the entire dataset and has a non-polynomial recommendation time (the Bayesian network) and the ones that compute a new neighborhood at each step.

The first group is clearly the fastest, being up to four orders of magnitude quicker than the neighborhood-based methods, for example. The naive Bayes network and the k-LP-trees recommendation time is between 0.001ms and 0.050ms. Furthermore, the k-LP-tree recommendation time seems less sensitive to the number of configured variables than the naive Bayes network. The second group, represented by the Bayesian network, needs less than 0.1 ms for the *Renault-44* and *Renault-48* datasets. It stays under 40 ms for the *Renault-87* dataset. However, since its inference algorithm is not polynomial in n, one can expect its recommendation time to explode with larger n. Finally, the third group is the slowest on *Renault-44* and *Renault-48*, which is explained by the time needed to extract the k best neighbors before computing the recommendation. However, this time is not too sensitive to the size of the problem – it remains low on the *Renault-87* instance.

One can check that on these datasets, which correspond to a real world application, the CPU times of all the method tested are compatible with an on-line use, with less than 40 ms in any case.

6.2.4 Influence of the sample's size

The drawback of the methods based on a neighborhood is that their performances seem to depend on the size of the original sample: the larger the sample size, the better the prediction but the higher the time needed to make it. To confirm this, we performed another experiment, varying the size of the sample, from the full sample to a sample containing only $1/100^{th}$ of the original one.

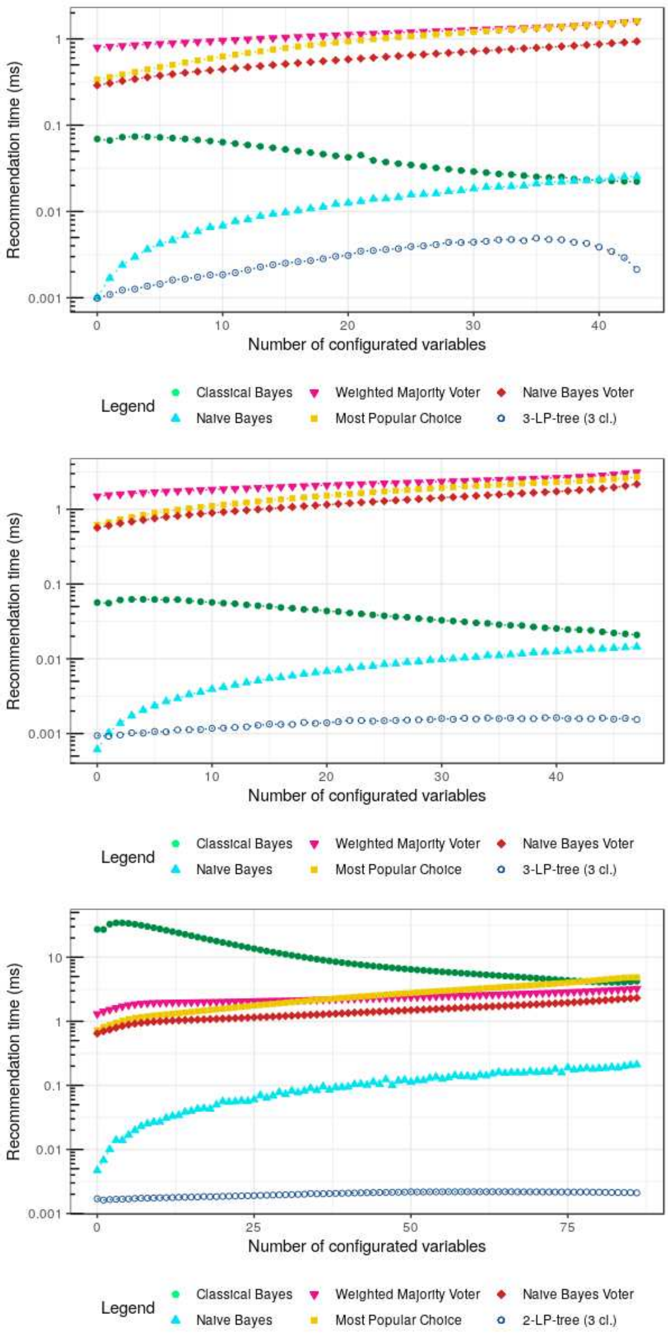


Figure 6: Average recommendation time on datasets *Renault-44* (top) *Renault-48* (middle) and *Renault-87* (bottom)

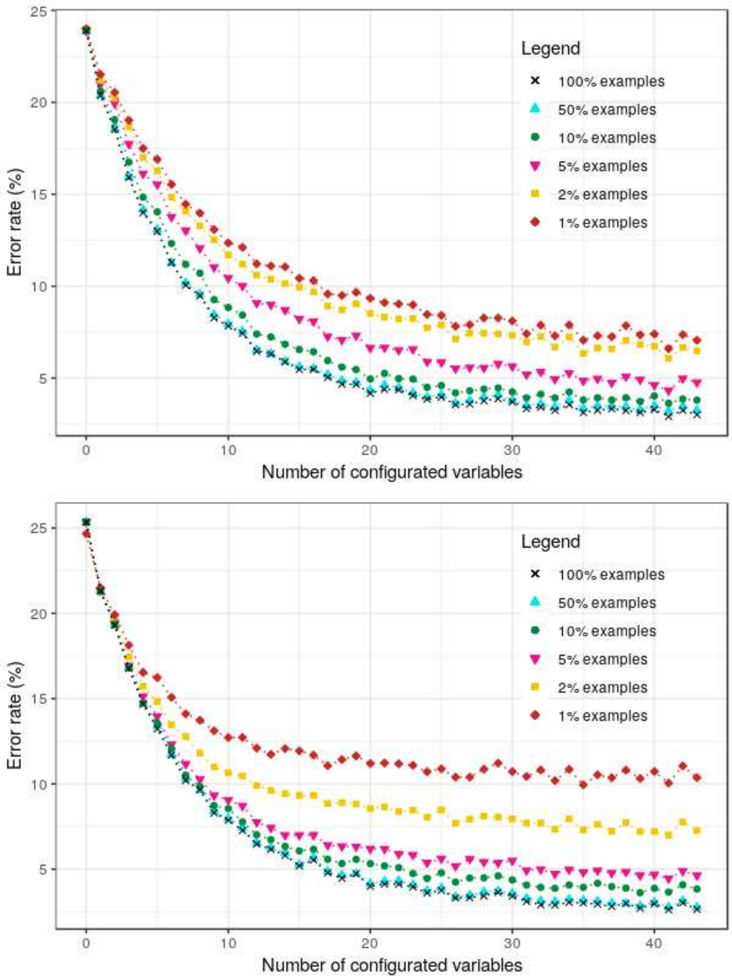


Figure 7: Average error rate of Bayesian network (top) Naive Bayes Voter (bottom) on dataset *Renault-44* w.r.t. the size of the learning sample

Results are shown in Figure 7. Both Bayesian network and Naive Bayes voter suffer an accuracy degradation when the sample size is reduced. However, we can remark that the accuracy degradation of the Bayesian network is moderate (error rate from 17.20% to 18.47%) while it is much worse for Naive Bayes voter (error rate from 16.76% to 19.76%). The Bayesian network success rate depends much less on the sample size that the neighborhood-based methods success rate.

Error rate on *Renault-44*	All examples	Consistent examples only
Naive Bayes Voter	19.90%	18.13%
Weighted Maj. Voter	20.14%	19.24%
Most Pop. Choice	20.39%	19.12%
Bayesian network	19.14%	18.28%
Naive B. network	23.71%	22.52%
3-LP-tree (3 cl.)	21.60%	21.91%

Error rate on *Renault-87*	All examples	Consistent examples only
Naive Bayes Voter	7.97%	9.11%
Weighted Maj. Voter	7.99%	9.03%
Most Pop. Choice	8.40%	9.52%
Bayesian network	12.51%	13.17%
Naive B. network	12.89%	13.52%
2-LP-tree (3 cl.)	11.73%	10.08%

Table 2: The error rate on *Renault-44* (top) and *Renault-87* (bottom) when using the entire training set or only the examples consistent with the constraints

6.2.5 Influence of the inconsistent examples

Even though we know that all future sold products will satisfy the constraints, the learning set may contain examples that do not (because constraints change over time). The question of whether discarding these examples may (or not) enhance the success rate is not trivial. On one hand, these examples, although they don't comply with the constraints, contain information about the user preferences. On the other hand, they may harm the success rate because they are not representative of examples in the test set. Hence the need of an experimental verification of whether it is worth, or not, to learn these inconsistent examples. We ran both experiments on the datasets; results are presented in Table 2.

While the error rate is decreased by discarding the inconsistent examples for the dataset *Renault-44* (results are similar for *Renault-48*), it is increased for *Renault-87* for all recommenders except k-LP-trees. It seems that the change of success rate (that either increases or decreases) primary depends on the dataset and not on the algorithm. Since the error rate difference is significant, we cannot conclude in the general case.

Remark that Bayesian networks tend to have a lower success rate in presence of constraints. This comes certainly from their probabilistic nature and the fact that some properties are lost when the probability distribution learnt has impossible (i.e. null probability) values, which is the case with constraints.

6.2.6 Influence of the constraints

In order to study the influence of the constraints on the recommendation success rate, we created two lightened version of the CSP of *Renault-48*: a CSP with about 30% of the constraint and a CSP with about 60%. We focus the study on the two recommenders with the best success rate: Naive Bayes Voter with $k = 35$ and the Bayesian network.

The success rate w.r.t. the ratio of constraints is summarized in Table 3. We can see a clear degradation of the success rate as constraints are added for both recommenders. The success rate difference is barely noticeable between the two recommenders – they are similarly affected by the constraints.

Ratio of constraints	Naive Bayes Voter	Bayesian Network
0%	6.49%	6.60%
30%	8.40%	8.14%
60%	12.30%	11.97%
100%	19.55%	19.14%

Table 3: The error rate of two recommenders w.r.t. the ratio of constraints on *Renault-48*

7 Conclusion

This paper has presented and experimented two families of approaches for the task of recommendation in interactive product configuration – the line proposed in [Coster et al., 2002], relying on the on-line selection a k-neighborhood, and an original line, in two step: a model of the entire sample is learned off-line, as representation of the users' preferences, and used on-line to recommend a pertinent value; three families of models are experimented here: Bayesian networks, naive Bayesian networks and lexicographic preferences trees.

Our experiments on real world datasets show that these methods are compatible with an on-line context. Bayesian Nets have a success rate close to the best possible one. The naive Bayes approximation leads to a very quick recommendation, but of lower quality. We showed that k-LP-trees have relatively low success rate because they have a limited expressivity and cannot perform a maximum a posteriori value recommendation. While we cannot change this second limitation, we can mitigate the first one by using clustering to enhance k-LP-trees expressivity and achieve a better success rate. They are the fastest recommenders we experimented with. The other approaches proposed in the literature (Naive Bayes Voter, Weighted Majority Voter and Most Popular Voter) have a success rate similar to the one of Classical Bayesian Nets, and a CPU time that is barely sensitive to the size of the instance – but strongly depends on the size of the sample.

We shall thus conclude in favor of the approach based on Bayesian net learning for problems with either a reduced sample or a very large sample, with little constraints and a reasonable number of variable. Otherwise, when the sample size is large enough and can still be explicitly memorized, the methods based on k-neighborhood constitute a simple yet accurate solution. Finally, one should keep in mind that naive Bayes and k-LP-trees shall be alternatives on situations involving very big instances, very limited memory resource or an extremely quick recommendation requirement.

References

[Adomavicius and Tuzhilin, 2005] Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749.

[Amilhastre et al., 2002] Amilhastre, J., Fargier, H., and Marquis, P. (2002). Consistency restoration and explanations in dynamic csps application to configuration. *Artificial Intelligence*, 135(1-2):199–234.

[Astesana et al., 2010] Astesana, J., Cosserat, L., and Fargier, H. (2010). Constraint-based vehicle configuration: A case study. In *Proceedings of ICTAI 2010*, pages 68–75.

[Bessiere et al., 2013] Bessiere, C., Fargier, H., and Lecoutre, C. (2013). Global inverse consistency for interactive constraint satisfaction. In *Proceedings of CP'13*, pages 159–174. Springer.

[Booth et al., 2010] Booth, R., Chevaleyre, Y., Lang, J., Mengin, J., and Sombattheera, C. (2010). Learning conditionally lexicographic preference relations. In *Proceedings of ECAI'10*, pages 269–274.

[Boutilier et al., 2001] Boutilier, C., Bacchus, F., and Brafman, R. I. (2001). Ucp-networks: A directed graphical representation of conditional utilities. In *Proceedings of UAI'01*, pages 56–64.

[Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research (JAIR)*, 21:135–191.

[Brafman and Domshlak, 2002] Brafman, R. I. and Domshlak, C. (2002). Introducing variable importance tradeoffs into cp-nets. In *Proceedings of UAI'02*, pages 69–76.

[Bräuning et al., 2017] Bräuning, M., Hüllermeier, E., Keller, T., and Glaum, M. (2017). Lexicographic preferences for predictive modeling of human decision making: A new machine learning method with an application in accounting. *European Journal of Operational Research*, 258(1):295–306.

[Bräuning and Hüllermeyer, 2012] Bräuning, M. and Hüllermeyer, E. (2012). Learning conditional lexicographic preference trees. In *Preference Learning (PL-12) - ECAI'12 Workshop*.

[Braziunas and Boutilier, 2005] Braziunas, D. and Boutilier, C. (2005). Local utility elicitation in GAI models. In *Proceedings of UAI'05*, pages 42–49.

[Chickering, 1995] Chickering, D. M. (1995). Learning bayesian networks is NP-complete. In *Proceedings of AISTATS'95*, pages 121–130.

[Cooper and Herskovits, 1992] Cooper, G. F. and Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine learning*, 9(4):309–347.

[Coster et al., 2002] Coster, R., Gustavsson, A., Olsson, T., Åsa Rudström, and Rudström, A. (2002). Enhancing web-based configuration with recommendations and cluster-based help. In *In Proceedings of the AH'2002 Workshop on Recommendation and Personalization in eCommerce*, pages 30–40.

[Dagum and Luby, 1993] Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153.

[Falkner et al., 2011] Falkner, A. A., Felfernig, A., and Haag, A. (2011). Recommendation technologies for configurable products. *AI Magazine*, 32(3):99–108.

[Fargier et al., 2016] Fargier, H., Gimenez, P.-F., and Mengin, J. (2016). Recommendation for product configuration: an experimental evaluation. In *Proceedings of the 18th International Configuration Workshop (CWS 2016)*, pages 9–16.

[Fargier et al., 2018] Fargier, H., Gimenez, P.-F., and Mengin, J. (2018). Learning lexicographic preference trees from positive examples. In *Proceedings of AAAI-18*, pages 2959–2966.

[Gonzales and Perny, 2004] Gonzales, C. and Perny, P. (2004). GAI networks for utility elicitation. In *Proceedings of KR'04*, pages 224–234.

[Hadzic et al., 2007] Hadzic, T., Wasowski, A., and Andersen, H. R. (2007). Techniques for efficient interactive configuration of distribution networks. In *Proceedings of IJCAI 2007*, pages 100–105.

[Jannach et al., 2010] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). *Recommender Systems - An Introduction*. Cambridge University Press.

[Kim and Pearl, 1983] Kim, J. H. and Pearl, J. (1983). A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of IJCAI'83*, pages 190–193.

[Lauritzen and Spiegelhalter, 1988] Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B*, 2(50):157–224.

[Lemmer and Kanal, 2014] Lemmer, J. and Kanal, L. (2014). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. *Proceedings of UAI'14*, page 149.

[Pargamin, 2002] Pargamin, B. (2002). Vehicle sales configuration : the cluster tree approach. In *Proceedings of ECAI'02 Configuration Workshop*.

[Pearl, 1989] Pearl, J. (1989). *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann.

[Ricci et al., 2011] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors (2011). *Recommender Systems Handbook*. Springer.

[Schiex et al., 1995] Schiex, T., Fargier, H., Verfaillie, G., et al. (1995). Valued constraint satisfaction problems: Hard and easy problems. *Proceedings of IJCAI'95*, pages 631–639.

[Schmidt, 2015] Schmidt, N. (2015). *Compilation de préférences - application à la configuration de produits*. Thèse de doctorat, Université d'Artois. (Soutenance le 14/09/2015).

[Spirtes et al., 2000] Spirtes, P., Glymour, C. N., and Scheines, R. (2000). *Causation, prediction, and search*. MIT press.

[Tsamardinos et al., 2006] Tsamardinos, I., Brown, L. E., and Aliferis, C. F. (2006). The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78.

[Ungar and Foster, 1998] Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, pages 114–129.