Recommendation using Bayesian inference for product configuration

Hélène Fargier , Pierre-François Gimenez , Jérôme Mengin

IRIT, CNRS, University of Toulouse, France
{helene.fargier,pierre-francois.gimenez,jerome.mengin}@irit.fr

ABSTRACT. *This paper deals with the problems of value recommendation in interactive product configuration. It proposes to learn independencies off-line, under the form of a dtree; and to use a variant of the principle of recursive conditioning, where the conditional probabilities are extrapolated on-line from the sample. Our experiments on real world datasets show that this approach is compatible with an on-line exploitation in terms of CPU time, and has a very good accuracy – its success rate is close to the best possible one.*

RÉSUMÉ. *Ce article propose une approche du problème de la recommandation de valeurs dans le cadre de la configuration interactive de produit. L'idée est d'apprendre, hors ligne, des indépendances entre variables et puis d'utiliser, en ligne, une variation de l'algorithme Recursive Conditioning qui utilise directement l'historique de vente. Nos expérimentations montrent que cette approche est compatible avec une exploitation en ligne en termes de temps CPU et possède une très bonne précision: son taux de succès est proche du meilleur possible.*

KEYWORDS: *Bayesian networks, recommender systems, product configuration, Bayesian inference*

MOTS-CLÉS : *Réseaux bayésiens, recommendation, configuration, inférence bayésienne*

1. Introduction

In on-line sale contexts, one of the main limiting factors is the difficulty for the user to find product(s) 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-catalogue, 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, the values of which have to be chosen by the user. The search space is thus highly combinatorial. It is generally explored following a step-by-step 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 values for the current

variable, one which is most likely to suit the user. In the context of the present work no prior knowledge about the user is available, but the system has a list of products previously configured and bought by other users ("sell histories").

Classical content-based and collaborative filtering approaches cannot be used here. The first reason is that the number of possible products is huge – exponential in the number of configuration variables. 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. 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, are not bought so often by one individual. So we have no information about similarity between users (upon which collaborative filtering approaches are based) nor on the preferences of the current user (upon which content-based filtering approaches are based).

Our general idea is to recommend, for a given variable at a given step of the configuration process, a value that has been chosen by most 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. Our approach is based on the idea that there is a ground probability distribution p over the space of all feasible products, indicating how likely it is that each object 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 configurated products eventually bought by past users are the ones they prefer. Therefore, if Next is the next variable to be assigned a value, and if u is the vector of values that have been chosen for the variables already decided, we propose to estimate, for each possible value x for Next, the marginal conditional probability $p(x \mid u)$ that Next has value x in the user's preferred product given u; and recommend to the user the value with the highest probability.

A first, naive method to compute $p(x \mid u)$ would be to count the proportion of x within the sold products that verify u. Even if this idea works for small u's, after a few steps the number of products that verify u would be too low and the computations would not be reliable enough (and even impossible when no product in the history verifies u). We explore in the present paper an alternative method, that combines the detection of independencies at work in Bayesian nets and the idea of the evaluation of the probability by a direct counting over the sample.

2. Background and notations

2.1. interactive configuration

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 $o \in \prod_{X \in \mathcal{X}} \underline{X}$; we denote by $\underline{\mathcal{X}}$ the set of all of them.

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

In interactive configuration problems, the user builds the product she is interested in through a variable by variable interaction. At each step, let u be the tuple of values already assigned to variables previously examined: the user freely selects the next variable to be assigned (we denoted Next this variable) and the system recommends value for Next.

In the context considered by this paper, sales histories are available, on which the system can rely to base its recommendation. Formally, a sales history is a (multi) set $\mathcal{H} \subseteq \underline{\mathcal{X}}$ of complete configurations that correspond to products that have been bought by past users. Users have different preferences, depending on their taste and their environment, which make them prefer different products - hence there is a wide variety of products in the histories. We do not have any information about their taste, nor do we use any information about their environment. Instead, we assume that there is a ground probability distribution p over the set configurations: $p(o)$ is the probability that o is the most preferred product for an anonymous user in an unknown environment.

For a value $x \in \underline{\mathrm{Next}}$, and a partial configuration u, $p(x \mid u)$ is the marginal probability that Next has value x in the most preferred product of the current user, given that u is also in that product, and we can recommend the most probable value:

$$\operatorname*{argmax}_{x \in \underline{X}} p(x \mid u).$$

Note that, since u is fixed, this is equivalent to finding the value x that maximizes $p(xu)$.

If we consider that the sales history is a sample of $\underline{\mathcal{X}}$ according to the unknown distribution p, we can use it to estimate the probabilities. In the sequel, for a partial configuration u, $\#(u)$ will denote the number of configurations in $\mathcal{H}$ that extend u. We have already mentioned that estimating $p(xu)$ with $\#(ux)$ is not feasible after a few steps because the number of products that verify u becomes too low (it can even quickly become null).

2.2. Bayesian networks

It is possible to learn from the data set a Bayesian network that represents, or at least approximates, the probability distribution p on the set of products. This network can then be used, on-line, during the step-by-step configuration session, to estimate the relevant marginal probabilities.

Given a Bayesian network $\mathcal{N}$, for $X \in \mathcal{X}$ we denote by $\mathrm{Pa}(X)$ the set of parents of X in the graph, and by $\Theta(X \mid \mathrm{Pa}(X))$ the probability table associated to X: it contains, for every $x \in \underline{\mathcal{X}}$ and every $u \in \underline{\mathrm{Pa}(x)}$, a parameter $\Theta(x \mid u)$. The Bayesian network uniquely defines a probability distribution p over $\mathcal{X}$: the probability of a complete configuration $o \in \underline{\mathcal{X}}$ is

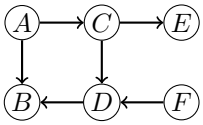
$$p(o) = \prod_{X \in \mathcal{X}} \Theta(o[X] \mid o[\mathrm{Pa}(X)])$$

As a shorthand notation, for a partial configuration $v \in \underline{V}$ such that V contains a variable X and all its parents U, we can define:

$$\Theta(X, v) = \Theta(v[X] \mid v[U]).$$

With this notation, we can write $p(o) = \prod_{X \in \mathcal{X}} \Theta(X, o)$.

EXAMPLE 1. — Consider the Bayesian network opposite, with $\mathcal{X} = \{A, B, C, D, E, F\}$. The probability of the complete configuration $abcdef \in \underline{\mathcal{X}}$ can be computed as:



$$\Theta(a)\Theta(c \mid a)\Theta(e \mid c)\Theta(f)\Theta(d \mid cf)\Theta(b \mid ad)$$

and $\Theta(D, abdcf)$ is defined to be $\Theta(d \mid cf)$. □

2.3. Recursive Conditioning

Inference in Bayesian networks is known to be hard, but several methods have been successfully applied to compute marginal probabilities in reasonable time. Recursive conditioning (Darwiche, 2001) is one of them. It is a divide and conquer algorithm: the principle is to "sum out" variables from a *cutset*, so that the remaining variables are separated into two sets that are not connected any more, and the probabilities of which can be computed independently. For tuples of variables $U, V \subseteq \mathcal{X}$ such that $\mathrm{Pa}(V) \subseteq U \cup V$, $u \in \underline{U}$, define:

$$F_V(u) = \sum_{v \in \underline{V \setminus U}} \prod_{X \in V} \Theta(X, uv)$$

Then, by definition, $p(u) = F_{\mathcal{X}}(u)$, and this product can be broken down using cutsets, that we first define:

DEFINITION 2. — *Let $V \subseteq \mathcal{X}$ be a tuple of variables, and let $\{V_1, V_2\}$ be a partition of V, the corresponding* cutset *is then* $(\mathrm{Pa}(V_1) \cap V_2) \cup (\mathrm{Pa}(V_2) \cap V_1)$.

Recursive conditioning then uses the following property:

PROPERTY 3. — *Let $U, V \subseteq \mathcal{X}$ such that $\mathrm{Pa}(V) \subseteq U \cup V$, $u \in \underline{U}$; let $\{V_1, V_2\}$ be a partition of V, let C be the corresponding* cutset*, then*

$$F_V(u) = \sum_{c \in \underline{C \setminus U}} F_{V_1}(uc) \times F_{V_2}(uc)$$

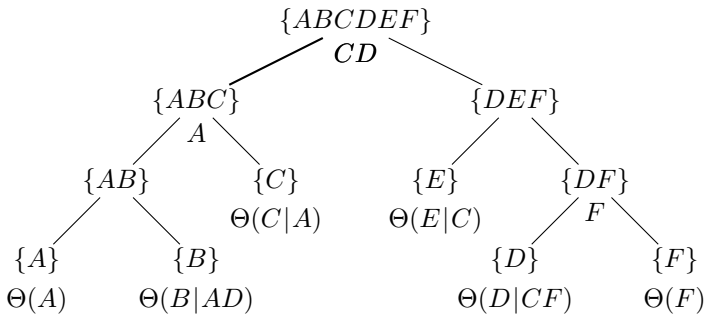


Figure 1. A possible balanced dtree for the Bayesian network of example 1.

In order to limit the number of terms in the sum, it is important that C be as small as possible. The choice of a good partition relies on the notion of d-separation (Geiger *et al.*, 1989), which is, in a faithful BN, equivalent to conditional independence.

The recursive conditioning algorithm recursively decomposes the sum-product computation by partitioning in two the set of variables at each step. Technically, a *dtree* is built: it is a binary tree, with leaves corresponding to the network CPTs. Each dtree provides a complete recursive decomposition of the initial set of variables $\mathcal{X}$, with a cutset for each level of the decomposition. Figure 1 depicts a possible balanced dtree for the Bayesian network of example 1. To each node of a dtree is associated a subset V_T of $\mathcal{X}$. If T is a leaf, then V_T is a singleton. If T is an internal node with children T_1 and T_2, then $\{V_{T_1}, V_{T_2}\}$ is a partition of V_T. If T is the root of the tree then $V_T = \mathcal{X}$. The tree is balanced if its height is 1 plus the integral part of $\log_2(|\mathcal{X}|)$. For each internal node T with children T_1 and T_2, let $C_{T,1} \stackrel{\text{def}}{=} \mathrm{Pa}(V_{T_2}) \cap V_{T_1}$ and $C_{T,2} \stackrel{\text{def}}{=} \mathrm{Pa}(V_{T,1}) \cap V_{T_2}$: the corresponding cutset is $C_T = C_{T,1} \cup C_{T,2}$.

3. Recommendation with direct Bayesian inference

When learning a Bayesian network from the sales history, the learnt network is only a rough approximation of the ground probability distribution p from which the sales history is supposed to have been generated, for several reasons. First, one assumes a certain type of (in)dependence structure: not every probability distribution can be represented by a DAG with some constraint on e.g. the in-degree of the nodes or the treewidth of the graph. Computing a marginal probability using approximate independencies can lead to approximate results, even if the local probability tables have correct values, just because these tables do not represent all necessary information. Second, the local probability tables associated with the chosen structure are estimated from the data, and these estimates are prone to error if not sufficient data is available.

In order to obtain better, and faster, estimates of the probabilities of interest, we now describe a way to directly estimate probabilities from the sales history, after they have been decomposed, using a dtree, into factors involving sufficiently few instanciated variables.

3.1. Direct Recursive Conditioning

When computing marginals with few instantiated variables, the Recursive Conditioning algorithm needs to go down to most leaves of the dtree, which is costly in time. But the data sample can often provide earlier (that is, at non leaf nodes) a better estimate of necessary probabilities, which leads to good performances, both in terms of precision and of time. Consider instances u and e of disjoint subsets U and E of $\mathcal{X}$, and suppose that we want to compute $p(u \mid e)$: it can be estimated as $\#(ue)/\#(e)$ if $\#(e)$ is large enough. The greater $\#(e)$, the more precise the estimation will be. It means that the fewer variables are in E, the better we can predict.

The RC algorithm splits sets of variables in two, so the cardinality of the sets of assigned variables of interest will usually decrease quickly as the algorithm goes down the branches of the dtree – even with the variables in the cutset added to it – and the corresponding subsample will grow accordingly. So, we propose a variation of the Recursive Conditioning algorithm: we use a dtree computed from the dependency structure of the induced Bayesian network *but* estimate the probabilities from the data sample, as soon as, in each branch of the dtree, the corresponding subsample is considered sufficient to give a good estimates.

We named this method Direct Recursive Conditioning (DRC), because in contrary to the Bayesian network inference, it works directly on the original data.

It is based on the following result, for which we introduce the notation $E_T \stackrel{\text{def}}{=} \mathrm{Pa}(V_T) \setminus V_T$ for every node T of the dtree. Algorithm 1 describes our approach by Direct Recursive conditioning.

PROPOSITION 4. — *Let T be a node of the dtree, let $U \subseteq V_T$, let $u \in \underline{U}$ and $e \in \underline{E_T}$. Then:*

$$p(u \mid e) = \sum_{c \in \underline{C_T \setminus U}} p(u_{c,1} \mid e_{c,1}) \times p(u_{c,2} \mid e_{c,2}) \times \lambda_c$$

where $u_{c,i} = (uc)[V_{T_i}]$ is the projection onto V_{T_i} of uc, and where $e_{c,i} = (uce)[E_{T_i}]$ is the projection of uce onto E_{T_i}; and where $\lambda_c = p(c \mid e)/(p(c[C_{T_2}] \mid c[C_{T_1}]e)p(c[C_{T_1}] \mid c[C_{T_2}]e))$ is a kind of normalization factor.

The two probabilities $p(u_{c,i} \mid e_{c,i})$ can be computed independently from each other, and each involves partial configurations over fewer variables if the size of the cutset is smaller than the size of V_{T_1} and V_{T_2}. The factor λ_C cannot be decomposed in the same way, since it somehow precisely measures how independent the variables in the two branches of the tree are. In the sequel, we make the assumption that C_{T_1} and C_{T_2} are independent given E, so that $\lambda_c = 1$.

Algorithm 1: DRC

Input: T: dtree node
$u \in \underline{U}$ for some $U \subseteq V_T$; $e \in \underline{E_T}$
Output: $p(u \mid e)$
main:
1 **if** *T is a leaf node or ($E \neq \varnothing$ and $count(e,T) > threshold$)* **then**
2 **return** `Estimate` (T, u, e)
3 **else**
4 $p \leftarrow 0$; $T_1, T_2 \leftarrow$ the two children of T
5 **for each** $c \in \underline{C_T}$, $c \sim u$ **do**
6 $p \leftarrow p+$ `DRC`$(T_1, (uc)[V_{T,1}], (uc)[C_2]e[E_1])$
 $\times$ `DRC`$(T_2, (uc)[V_{T,2}], (uc)[C_1]e[E_2])$
7 **return** p

Function `Estimate` $(T,\ u,\ e)$ *// estimate* $p(u \mid e)$
1 **if** $count(e,T) = 0$ **then return** *0*
2 **else return** $count(ue,T)/count(e,T)$

EXAMPLE 5 (continuation of example1). — Suppose we want to compute $p(abdf)$, using the dtree of figure 1, and assume that $\#(d)$ and $\#(c)$ are large enough, then:

$$p(abdf) \approx \sum_c p(abc \mid d) \times p(df \mid c) \approx \sum_c \frac{\#(abd)}{\#(d)} \times \Theta(c \mid a) \times \frac{\#(cdf)}{\#(c)}$$

□

Function $Estimate$ computes the proportion $\#(ue)/\#(u)$ as an estimation of $P(e|u)$. In order to perform efficiently these counting requests, the sample is compactly represented under the form of an ADD (Bahar *et al.*, 1997) (counting is takes linear time on this kind of data structure). It would be possible to use only one ADD to compactly represent the sample. However, since the complexity of the counting request in an ADD depends on the number of its variables, we associate one ADD to each node T of the dtree, with the variables needed at that node, i.e. $V_T \cup E_T$. The farther away from the root a node T is, the smaller $|V_T \cup E_T|$, the smaller the ADD, the faster the counting request. So, there are as many ADDs as there are dtree nodes. Adding these small ADDs necessitates more memory, but not too much (actually, the memory used is multiplied by the depth of the dtree). That is why the calls to function $count$, in Algorithm 1 take T as an input.

The main difference with Darwiche's RC algorithm is that the latter computes conditional probabilities for leaf nodes only while in DRC a call to $Estimate(T, u, e)$ may return $p(u[V_T] \mid E_T = e)$ when T is not a leaf node: a threshold is used at line 1 that decides whether the sales history can be used to estimate the probability or if it is better to continue the recursive conditioning. Hence the branches explored in DRC

are generally shorter than the ones explored by RC. But, ultimately, the complexity of DRC is not reduced with respect to the one of RC: indeed, even if we potentially terminate the inference earlier, extra work appears in the form of counting requests.

4. Experiments

The approache proposed in this paper has been tested on a case study of three sales histories provided by Renault, a French automobile manufacturer, available at http://www.irit.fr/~Helene.Fargier/BR4CP/benches.html. These datasets, named "*small*", "*medium*" and "*big*", 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: dataset "*small*" (resp. "*medium*", resp. "*big*") has 48 (resp. 44, resp .87) variables and 27088 (resp. 14786 examples, resp. 17724) examples. Most of the variables are binary, but not all of them.

We compare this approach with the basic one, that learns a bayesien network and build recommendation by computing marginals on the variables of interest. We used the R package *bnlearn* to learn the Bayesian networks.The average number of parents of a node in the obtained BN is about 1.17, 1.02 and 0.98 - for *small*, *medium* and *big*, respectively. We have tested two algorithms of inference, the jointree algorithm provided by the library Jayes and our own implementation of RC. We have also implemented the DRC approach. In the following experiments, both RC and DRC infer with the same dtree, which has been generated using the hypergraph partitioning method described by Darwiche and Hopkins (2001). The threshold used in DRC is 10.

4.1. Experimental protocol

We used two-fold cross-validation: each dataset has been divided in two halves, one half of the samples are used to learn the Bayesian network and estimate probabilities, recommendations are produced for samples of the other half, and compared to actual values in each sample.

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 genuine configuration sessions. That is why we generate a session $session$ for each product P in the history by randomly ordering its variable-value assignments . Then, for each variable-value assignment (X, x) in this sequence, the recommender is asked for a recommendation for X, say r: r may be equal to x; or not, if r more probable than x; then X is set to x. We consider a recommendation as correct if the recommended value is the one assigned to X in the product P (i.e. if $r = x$). Any other value is considered as incorrect.

The recommendation algorithm is evaluated by (i) the time needed for computing the recommendations and (ii) its success rate, obtained by counting the number of correct and incorrect recommendations.

In order to easily interpret the results of the cross-validation, we also computed the highest success rate attainable for the test set. If we were using an algorithm that already knows the testing set, it would use the probability distribution estimated from this testing set. Therefore it would recommend for the variable Next, given the assigned values u, the most probable value of X in the subset of products, in the test set, that respect u. More precisely, for any x in the domain of Next, it would estimate $p(x|u)$ as $\#(ux)/\#(u)$. Notice that $\#(u)$ is never equal to zero, since the test set contains at least one product consistent with u: the one corresponding to the current session. It is an algorithm overfitted to the testing set. We call this algorithm "Oracle". Its success rate is higher than the one of any other strategy. But it is not 100% since there is an variability in the users (otherwise only one product would be sold ...).

4.2. Results

The experiment have been made on a computer with a quad-core processor i5-3570 at 3.4Ghz, using a single core. All algorithms are written in Java, and the Java Virtual Machine used was OpenJDK.

Figure 2 gives the success rate of the BN-based approach and of DRC-approach proposed in this paper. As to BN-based approach, we have tested the RC and the jointree algorithms for the inference of the marginals. Because they are two exact inference algorithms and proceed on the same Bayesian network, they have got the same success rate. The oracle is given as an ideal line.

It appears that DRC has a slightly better success rate than the BN-based approach. This difference is explained by the small loss of information during the Bayesian network learning, while DRC always goes back to the sample and performs an extrapolation that takes the current context (the assignment) into account.

The average success rates of both the BN-based approach and the DRC one are close to the one of the Oracle. The gap gets larger when the number of assigned variables increases: the Oracle's performance becomes less and less attainable. Indeed, the prediction of the Oracle relies on the testing sample, that includes the product of the ongoing configuration. When few variables are instantiated, the Oracle uses a rather big subsample to make its estimation. When a lot of variables are instantiated, the Oracle uses a small subsample, so small that sometimes it contains only the ongoing configuration. In this case, the Oracle can't make a bad recommendation. This can be interpreted as a overfitting, since the Oracle is tested on the sample it learnt.

One can check than on this dataset, which corresponds to a real world application, the CPU time are sustainable for all the algorithms: less than $10ms$ for the *small* and *medium* data set, less than 0.25 s for the *big* one. Moreover DRC inference is generally

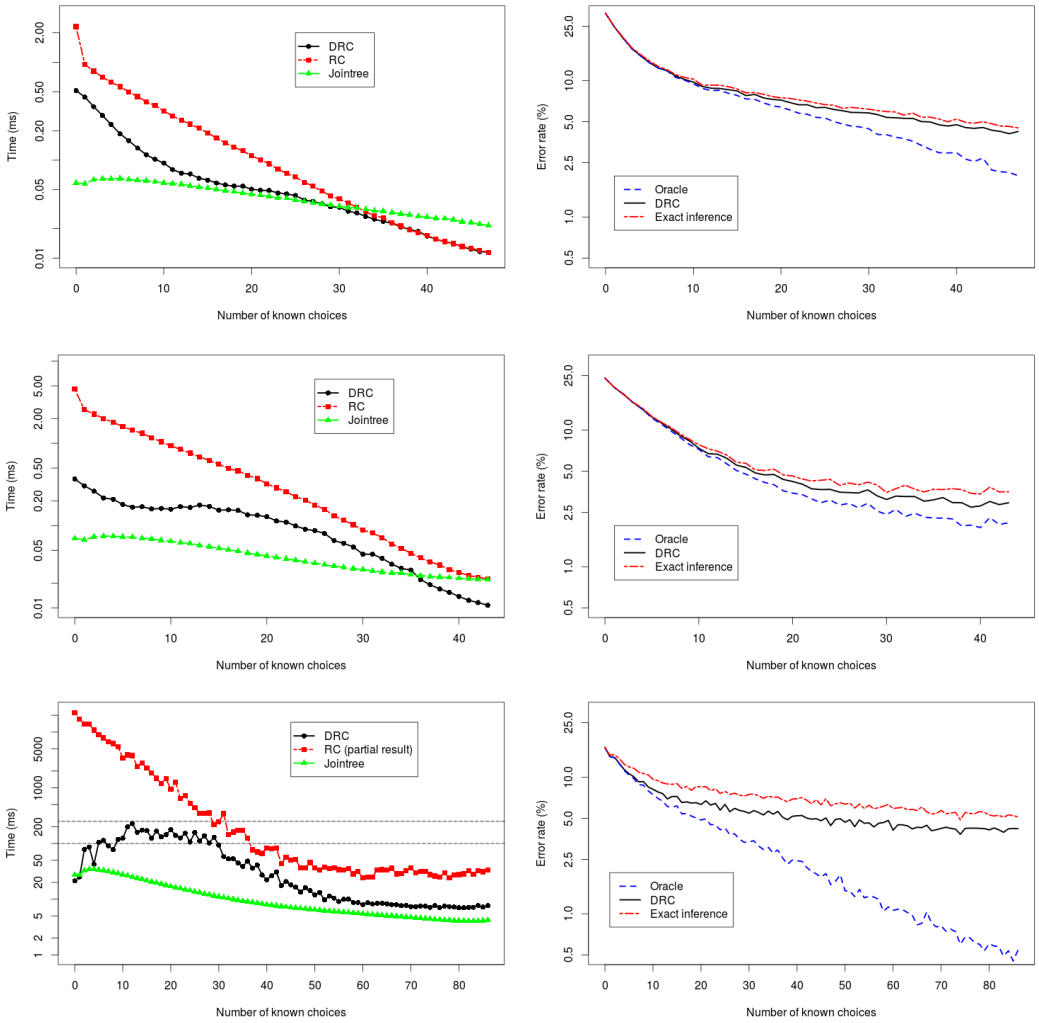


Figure 2. Average time (left) and average error rate (right) on datasets "small" (upper figures), "medium" (middle figures) and "big" (lower figures)

faster than the original RC. Depending on the dataset, the recommending time is either decreasing (see "*small*" result) or either shaped as a bell.

5. Conclusion

This paper has proposed two approaches to the problem of value recommendation in interactive product configuration. Both are based on Bayesian independence but differ in the way the sample is used. The first one learns a Bayesian net from the sample in an off-line phase. The Bayesian net is then used on-line to perform the recommendation task. The other one, called Direct Recursive Conditioning, learns independencies off line (when computing the dtree); but the probabilities are estimated on-line, by a direct reference to the sample: this allows the context (the current assignment) to be taken into account.

Our experiments on real world datasets show that both are compatible with an on-line context, and both have a very good accuracy - their success rate is close to the best possible one. It also appears that, the DRC approach has a slightly better accuracy.

Beyond recommendation for interactive configuration, the DRC approach can be used for any other application based on Bayesian inference on a sample of combinatorial objects. In an orthogonal way, the idea of directly using the sample to estimate a probability on the fly could be extended to other inference algorithms than RC.

References

Bahar R. I., Frohm E. A., Gaona C. M., Hachtel G. D., Macii E., Pardo A. *et al.* (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, Vol. 10, No. 2/3, pp. 171–206.

Darwiche A. (2001). Recursive conditioning. *Artif. Intell.*, Vol. 126, No. 1-2, pp. 5–41.

Darwiche A., Hopkins M. (2001). Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *Symbolic and quantitative approaches to reasoning with uncertainty, 6th european conference, ECSQARU 2001, toulouse, france, september 19-21, 2001, proceedings*, pp. 180–191.

Geiger D., Verma T., Pearl J. (1989). d-separation: From theorems to algorithms. In *UAI '89: Proceedings of the fifth annual conference on uncertainty in artificial intelligence, windsor, ontario, canada, august 18-20, 1989*, pp. 139–148.