Interactive configuration with constraints consistency and recommendation

Hélène Fargier
Pierre-François Gimenez
Jérôme Mengin

IRIT-CNRS, University of Toulouse
IRISA, CentraleSupélec

20th May 2022

Complex products

Complex, highly customizable products (combinatorial domains)
$\rightarrow$ **cars**, computers, travels, kitchens...
$\rightarrow$ number of possibilities exponential in the number of configuration variables
$\rightarrow$ all products aren't feasible (like a convertible car with a sunroof)

Presence of hard constraints

The constraints are hard : some products are infeasible

They come from :

- technical limitations (no sunroof on a convertible car)
- commercial considerations (no leather wheel on a lower-end car)
- stock variability (out-of-stock item)
- etc.

Renault Master : 10^{21} cars, 10^{16} feasible cars

Interactive configuration process

Product construction: the interactive configuration process

- the user chooses a configuration variable
- the configurator proposes possible values
- the user chooses a value for this variable

This process continues until the product is fully defined

Every proposed value must lead to a possible vehicle, but it's an NP-hard problem ! Two techniques :

- constraints propagation [Wal72]
- compilation [AFM02]

Recommendation

At each step of the interactive configuration, there is a partial, ongoing configuration

Recommendation = recommend, given a **partial configuration** u, a **value** for a **variable** Next

A good recommendation is:

- relevant
  $\rightarrow$ the user is willing to accept
- quick
  $\rightarrow$ on-line application

Context

- We have a sales history from Renault, no other information
  $\rightarrow$ no information about the user
- The user chooses the variables one by one
  $\rightarrow$ the order of the variables is unknown
- There are constraints on allowed configurations
  $\rightarrow$ we use the *SaLaDD* compiler [Sch15]
- The sales history products may or may not satisfy the constraints

State of the art

Recommendation in interactive configuration not very studied

Two categories of existing tools:

- Bayesian networks
- *k*-nearest neighbours [CGO+02]

We also contributed on a third class of models: lexicographic preference tree (LP-tree)

Goal: experiment and compare these methods

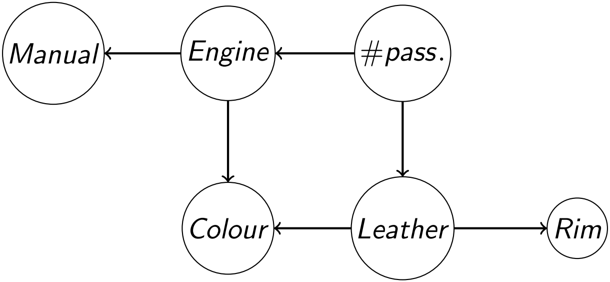
Outline

1. Context and issue
2. Algorithms
   1. based on Bayesian networks
   2. based on k-nearest neighbours
   3. based on LP-tree
3. Experiments
4. Conclusion

Bayesian network

Bayesian networks represent a probability distribution on the configurations by means of a direct acyclic graph (DAG) and probability tables

- Each node is a variable
- An edge between A and B means that the probability of A depends on the value of B (and vice-versa)



How to recommend with a Bayesian network ?

Probability $p(o)$ that a car o will be bought

Our recommendation is based on:

$$\underset{x \in \underline{\mathrm{Next}}}{\operatorname{argmax}} \, p(\mathrm{Next} = x \mid \mathrm{Assigned} = u)$$

Next is the configuration variable chosen by the user, u the partial configuration

We assume the sales history are a representative sample of future user choices

Two phases:

- Learn a Bayesian network from the sales history off-line
  $\rightarrow$ constraints aren't taken into account during the learning
- Recommend a value of the conf. variable on-line
  $\rightarrow$ the learning isn't critical, the inference is

Neighbourhood-based algorithms

3 algorithms based on *k*-nearest neighbours

Instead of using the whole sample, they use previous sales similar to the current one

The 3 algorithms process these neighbours in a different way

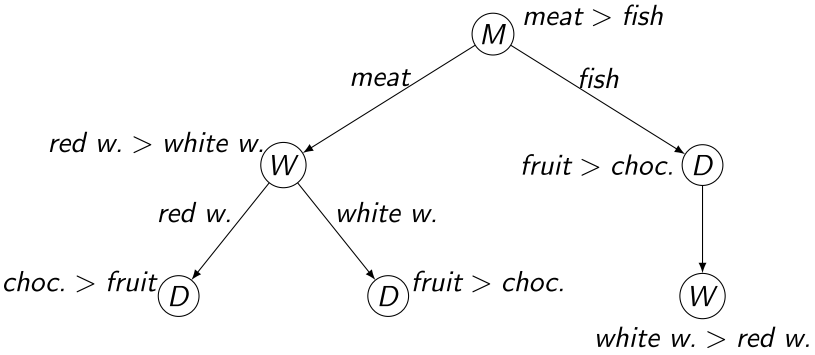
Three algorithms

Among the k-nearest neighbours of the current partial configuration

Weighted Majority Voter: each neighbours votes with a weight proportional to its similarity with the current configuration

Naive Bayes voter: uses the neighbours to learn a naive Bayesian network. No learning is possible off-line

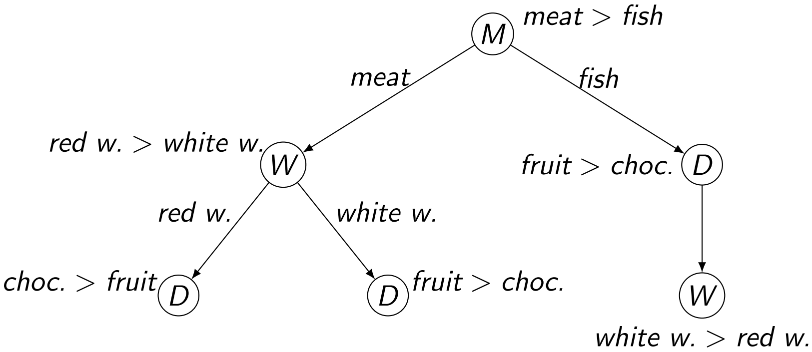
Most popular choice: computes the most probable completion of the current configuration and recommend the value of Next in it



LP-tree definition

- Tree of attributes ordered by importance (root: most important)
- Edges can be labelled by a value or not
- Preferences rules associated to each node

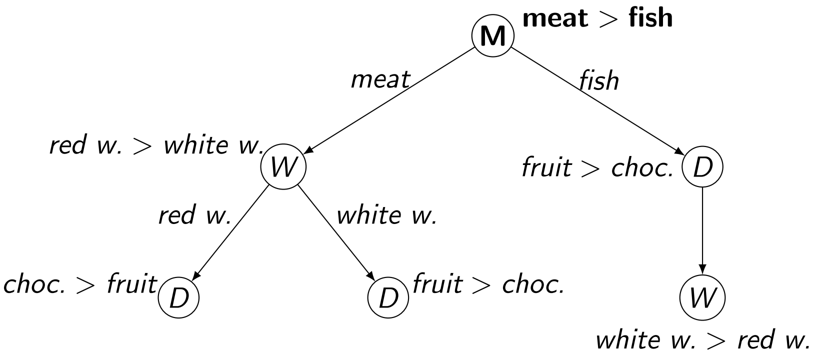
Lexicographic Preference Trees (LP-trees)



We would like to compare:

meat – red w. – fruit and *fish – white w. – fruit*

Lexicographic Preference Trees (LP-trees)

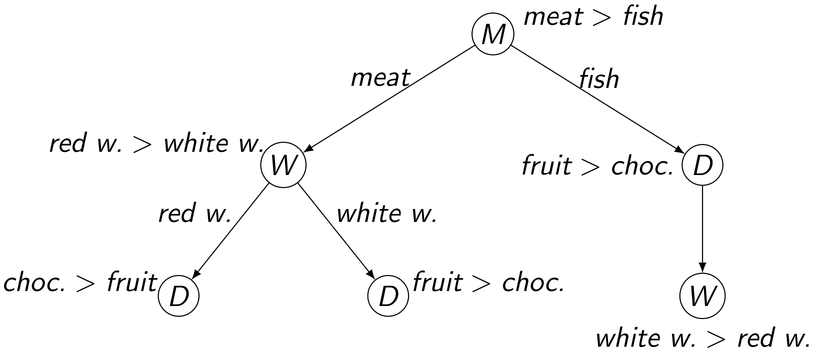


We would like to compare:

meat – red w. – fruit $\succ$ *fish – white w. – fruit*

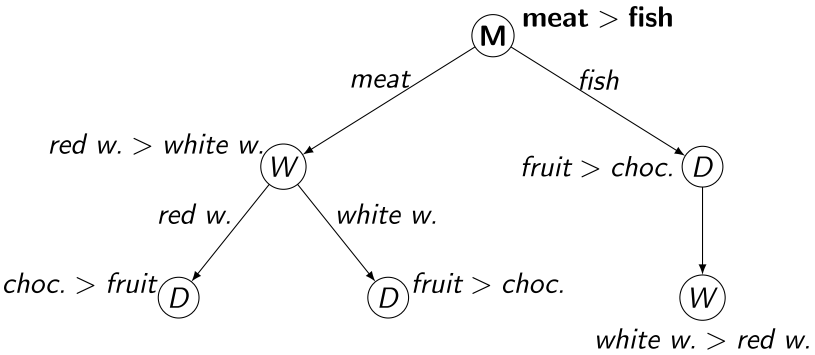
Any menu with meat is preferred to any menu with fish

Lexicographic Preference Trees (LP-trees)



We would like to compare:

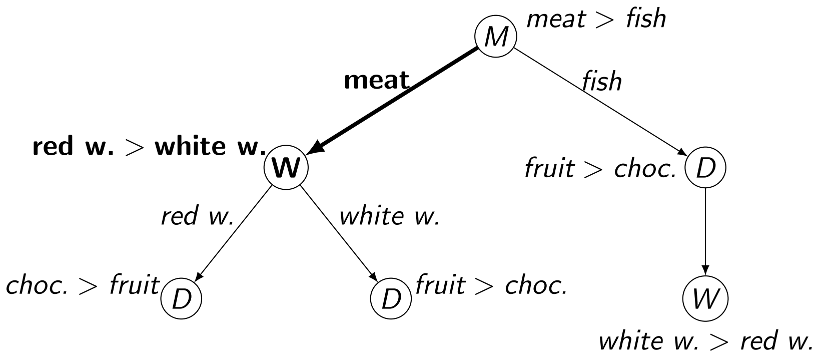
meat – white w. – fruit and *meat – white w. – choc.*



We would like to compare:

meat – white w. – fruit and *meat – white w. – choc.*

Root node cannot decide the comparison

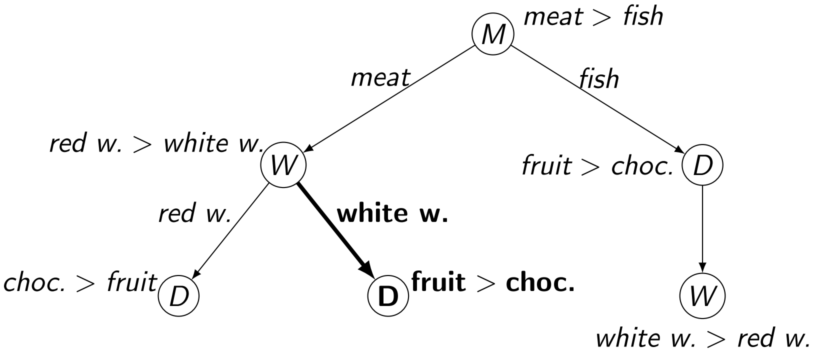


We would like to compare:

meat – white w. – fruit and *meat – white w. – choc.*

Among meat menus, any menu with red wine is preferred to any menu with white wine

Lexicographic Preference Trees (LP-trees)



We would like to compare:

meat – white w. – fruit $\succ$ *meat – white w. – choc.*

How to learn LP-trees?

Observation

We don't always choose our most preferred outcome
(e.g. because of lack of availability, a desire of variety, mass confusion, etc.)

Ground idea

- The more preferred an outcome is, the more often it is chosen
- **Probability $p(\mathbf{o})$ of selecting o increases w.r.t. the preference relation $\succ$: $p(\mathbf{o}) \geq p(\mathbf{o}')$ iff $\mathbf{o} \succeq \mathbf{o}'$**

Contribution

One of my PhD contributions was algorithms to learn LP-trees from sales history based on this ground idea

$\rightarrow$ I won't go into the details

Experimental protocol

10 folds cross-validation : history sales split into a training set and a test set

- Training set: Bayesian networks learning / neighbours searching
- Test set: for each item we simulate a configuration session
  For each recommendation for Next, we compare the recommended value with the value really chosen
  → Only one possible value: no evaluation
  → Recommended = chosen: success, else: failure

We measure the success rate and the recommendation time w.r.t. the number of assigned variables

Oracle: lower bound on error rate

We know that car sales are 40% diesel and 60% petrol. What is the best strategy of engine recommendation ?

Probability theorem

Let X be a random variable drawn from a probability distribution p. The estimator $\hat{x}$ that maximises $p(X = \hat{x})$ is $\hat{x} = \text{argmax}_{x \in \underline{X}}\, p(x)$ (i.e. MAP estimator)

Oracle

- The strategy that maximises the ratio of accepted recommendation is the MAP recommendation
- The oracle is a simple MAP recommender that *knows the test set*
- It achieves the highest possible accuracy: upper bound on accuracy

Datasets from Renault

Experiments made on i5 processor at 3.4GHz, using one core
All algorithms are written in Java

- dataset "*Renault-44*"
  - 44 variables
  - 14786 examples, 8252 examples consistent with the constraints
  - 70.80% recommendations are trivial
- dataset "*Renault-48*"
  - 48 variables
  - 27088 examples, 710 examples consistent with the constraints
  - 71.73% recommendations are trivial
- dataset "*Renault-87*"
  - 87 variables
  - 17715 examples, 8335 examples consistent with the constraints
  - 46.89% recommendations are trivial

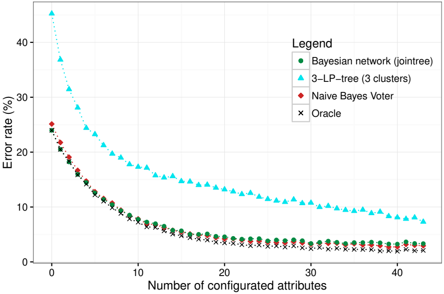
Should we use clusters? (Renault-48)

Error rate	1 cluster	2 clusters	3 clusters
Naive Bayes Voter	8.55 %	9.49 %	10.06 %
Bayesian network	8.49 %	9.49 %	10.04 %
3-LP-tree	17.24 %	13.52 %	11.98 %

Results

Clusters are useful with LP-trees because they enhance the expressivity

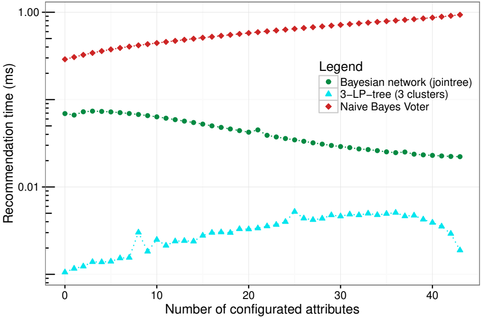
Error rate w.r.t. the number of assigned variables



Results on Renault-44

- Bayesian network and Naive Bayes Voter very similar ($\approx 7\%$), very close to the oracle
- LP-tree less accurate ($\approx 15\%$) because:
  - less expressive
  - not MAP

Time w.r.t. the number of assigned variables



Results on Renault-44

- Very quick recommendation (< 1ms)
- Bayesian network inference is NP-hard
- LP-tree inference is linear in n

Should we learn using “invalid” examples?

Datasets contain examples that don’t satisfy the constraints

Results on Renault-44:

Error rate	All examples	Consistent examples
Naive Bayes Voter	19.90 %	18.13 %
Bayesian network	19.14 %	18.28 %
3-LP-tree	21.60 %	21.91 %

Results

- Higher precision for Renault-44 and Renault-48
- Lower precision for Renault-87
- Cannot conclude for the general case

Error rate w.r.t. the sample size

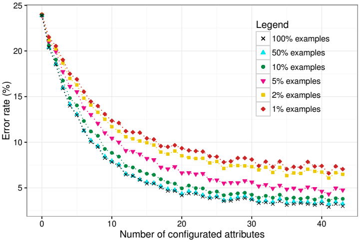


Figure: Bayesian network

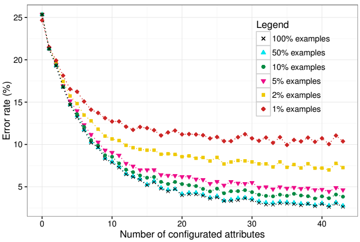


Figure: Naive Bayes Voter

Results on Renault-44

- The more examples, the more precise
- Bayesian networks are more precise with less examples

Error rate w.r.t. the amount of constraints

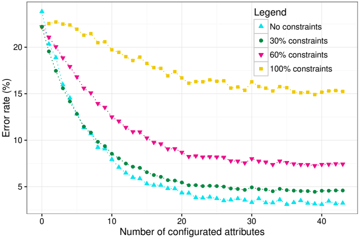


Figure: Bayesian network

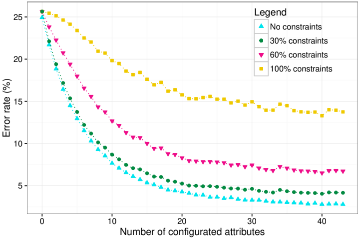


Figure: Naive Bayes Voter

Results on Renault-44

- The precision is much lower with constraints
- Both methods are equally affected

Conclusion

- Constraint compilation allows for on-line configuration
- Evaluation of state of the art and new methods
- Study on genuine datasets
- Very promising: fast methods with high accuracy
  - *k*-nearest neighbours and Bayesian networks are accurate and fast enough
  - LP-tree is adapted when execution time is more critical than accuracy

Bibliographie I

Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis.
Consistency restoration and explanations in dynamic csps application to configuration.
Artificial Intelligence, 135(1-2):199–234, 2002.

Rickard Coster, Andreas Gustavsson, Tomas Olsson, Åsa Rudström, and Asa Rudström.
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, 2002.

Nicolas Schmidt.
SALADD, le compilateur SLDD.
https://github.com/SchmidtNicolas/SALADD, 2015.

David L Waltz.
Generating semantic descriptions from drawings of scenes with shadows.
1972.