A KC Map of Valued Decision Diagrams
- application to product configuration -

Hélène Fargier\textsuperscript{1}  Pierre Marquis\textsuperscript{2}
Alexandre Niveau\textsuperscript{3} Nicolas Schmidt\textsuperscript{1,2}

\textsuperscript{1} IRIT-CNRS, Univ. Paul Sabatier, Toulouse, France
\textsuperscript{2} CRIL-CNRS, Univ. Artois, Lens, France
\textsuperscript{3} GREYC-CNRS, Univ. Caen, France

June 4, 2015
Outline

Configuration and Compilation

Valued Decision Diagrams

A Compilation Map for Real Valued Decision Diagrams

Experiments
Outline

Configuration and Compilation

Valued Decision Diagrams

A Compilation Map for Real Valued Decision Diagrams

Experiments
Introductory example

- Problem of interactive product configuration: a car

- Configure:
  - the motor — solar or pedals
  - the color — blue or red
  - the size — family car or two-seater
  - the radio option — with or without

- Constraints:
  - pedal cars must be red
  - solar panels do not on two-seaters
  - family cars all have a radio
Introductory example

• Problem of interactive product configuration: a car

• Configure:
  ▶ the motor — solar or pedals
  ▶ the color — blue or red
  ▶ the size — family car or two-seater
  ▶ the radio option — with or without

• Constraints:
  ▶ pedal cars must be red
  ▶ solar panels do not fit on two-seaters
  ▶ family cars all have a radio
Basic Problem

- Configurable product $\rightarrow$ \textit{constraint satisfaction problem (CSP)}
  - Configuration parameter $=$ a CSP variable (finite domain)
  - Constraints
    \[
    \begin{align*}
    \text{motor} &= \text{pedals} \rightarrow \text{color} = \text{red} \\
    \text{motor} &= \text{solar} \rightarrow \text{size} > \text{twoseater} \\
    \text{size} &= \text{twoseater} \lor \text{radio} = \text{with}
    \end{align*}
    \]
  - each solution corresponds to an admissible configuration
Basic Problem

- Configurable product $\rightarrow$ constraint satisfaction problem (CSP)
  - Configuration parameter = a CSP variable (finite domain)
  - Constraints
    \[
    \begin{cases}
    motor = \text{pedals} &\rightarrow color = \text{red} \\
    motor = \text{solar} &\rightarrow size > \text{twoseater} \\
    size = \text{twoseater} &\lor radio = \text{with}
    \end{cases}
    \]
  - each solution corresponds to an admissible configuration

- Configuration process:
  - The program presents, for each variable, values that lead to at least one solution
  - The user assigns a value to some variable
  - Which are the values of the free variables that are not consistent?
Basic Problem

- Configurable product $\rightarrow$ constraint satisfaction problem (CSP)
  - Configuration parameter $=$ a CSP variable (finite domain)
  - Constraints
    
    \[
    \begin{align*}
    &\text{motor} = \text{pedals} \rightarrow \text{color} = \text{red} \\
    &\text{motor} = \text{solar} \rightarrow \text{size} > \text{twoseater} \\
    &\text{size} = \text{twoseater} \lor \text{radio} = \text{with}
    \end{align*}
    \]
  - each solution corresponds to an admissible configuration

- Configuration process:
  - The program presents, for each variable, values that lead to at least one solution
  - The user assigns a value to some variable
  - Which are the values of the free variables that are not consistent?

- NP-complete problem $\ldots$ but the user cannot wait too long after each choice
A solution: knowledge compilation

- The CSP is a fixed part of the problem
  → we can compile it into a suitable data structure, such as an OBDD or a MDD:

- Assigning values to variables (conditioning) and checking consistency are polynomial operations on MDDs/OBDDs
  → the user’s wait is reduced
Configuration and Compilation

Configuration is an "Historical" application of compilation techniques

- Synthesis Trees [Weigel and Faltings, 1999]
- Prime Implicates (?) [Sinz, 2002]
- OBDDs, Ordered MDD [Amilhastre et al., 2002, Hadzic, 2004]
- Cluster Trees [Pargamin, 2002]
- ... 

By the way, several properties a not compulsory: "linerarity" of the structure, determinism, ordering of the variables.
Choosing a compilation language

- Which language is the best for my application?
  → use the compilation map [Darwiche and Marquis, 2002]
- Compares languages according to two criteria:
  1. efficiency of operations
  2. succinctness
Compilation map: operations

- All online manipulations amount to elementary queries and transformations

<table>
<thead>
<tr>
<th>L</th>
<th>CO (consistency)</th>
<th>VA (validity)</th>
<th>CE (clause entailment)</th>
<th>IM (implicant check)</th>
<th>EQ (equivalence)</th>
<th>SE (entailment)</th>
<th>CT (model count)</th>
<th>ME (model enum.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNF</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>DNNF</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>BDD</td>
<td>○ ○</td>
<td>○ ○</td>
<td>? ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>DNF</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>CD (conditioning)</th>
<th>FO (forgetting)</th>
<th>SFO (single forg.)</th>
<th>∧C (conjunction)</th>
<th>∧BC (bounded conj.)</th>
<th>∨C (disjunction)</th>
<th>∨BC (bounded disj.)</th>
<th>¬ (negation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNF</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>DNNF</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>BDD</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>? ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>DNF</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
</tbody>
</table>

✓ polynomial
○ not polynomial, unless \( \text{P} = \text{NP} \)
● not polynomial
Compilation map: operations

- All online manipulations amount to elementary queries and transformations

<table>
<thead>
<tr>
<th>L</th>
<th>CO (consistency)</th>
<th>VA (validity)</th>
<th>CE (clause entailment)</th>
<th>IM (implicant check)</th>
<th>EQ (equivalence)</th>
<th>SE (entailment)</th>
<th>CT (model count)</th>
<th>ME (model enum.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNF</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>DNNF</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>BDD</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>? ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>DNF</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>✓ ✓</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>CD (conditioning)</th>
<th>FO (forgetting)</th>
<th>SFO (single forg.)</th>
<th>(\land) C (conjunction)</th>
<th>(\land) BC (bounded conj.)</th>
<th>(\lor) C (disjunction)</th>
<th>(\lor) BC (bounded disj.)</th>
<th>(\lnot) C (negation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNF</td>
<td>✓ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>DNNF</td>
<td>✓ ✓</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>BDD</td>
<td>✓ ✓</td>
<td>○ ○</td>
<td>○ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
<td>✓ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>✓ ✓</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
</tr>
<tr>
<td>OBDD</td>
<td>✓ ✓</td>
<td>● ✓</td>
<td>● ✓</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
<td>● ○</td>
</tr>
<tr>
<td>DNF</td>
<td>✓ ✓</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>✓ ✓</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
</tbody>
</table>

✓ polynomial
○ not polynomial, unless P = NP
● not polynomial
Compilation map: operations

- All online manipulations amount to elementary queries and transformations

<table>
<thead>
<tr>
<th>L</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
<th>CNF</th>
<th>DNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNF</td>
<td>° ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>DNNF</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>BDD</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>? ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>? ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>OBDD</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>DNF</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>○ ○</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>CD</th>
<th>FO</th>
<th>SFO</th>
<th>C</th>
<th>BC</th>
<th>C</th>
<th>BC</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNF</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>DNF</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>BDD</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>FBDD</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>OBDD</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>DNF</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
<tr>
<td>CNF</td>
<td>√ ○</td>
<td>○ ○</td>
<td>○ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
<td>√ ○</td>
</tr>
</tbody>
</table>

√  polynomial
○  not polynomial, unless P = NP
●  not polynomial
Compilation map: succinctness

- Succinctness relation ($\leq_s$): orders languages
- $L_1 \leq_s L_2$ means “$L_1$ is at least as succinct as $L_2$”
Compilation map: succinctness

- Succinctness relation \((\leq_s)\): orders languages
- \(L_1 \leq_s L_2\) means “\(L_1\) is at least as succinct as \(L_2\)”
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others.
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others.
- The user assigns a value to some variable, or removes a previous assignment (without any prescribed order).

Study non-Boolean compilation languages.
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others.
- The user assigns a value to some variable, or removes a previous assignment (without any prescribed order).
- The program should provide explanations for invalid choices, propose restorations, alternative values, etc.
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others
- The user assigns a value to some variable, or *removes a previous assignment* (without any prescribed order)
- The program should provide *explanations* for invalid choices, propose *restorations*, *alternative values*, etc.
- The program maintains the *cost of cheapest car* consistent with the current choices
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others.
- The user assigns a value to some variable, or removes a previous assignment (without any prescribed order).
- The program should provide explanations for invalid choices, propose restorations, alternative values, etc.
- The program maintains the cost of cheapest car consistent with the current choices.
- Upon demand, it presents the minimal and maximal costs associated to the remaining choices.
The full configuration process

A more complex process:

- The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others.
- The user assigns a value to some variable, or removes a previous assignment (without any prescribed order).
- The program should provide explanations for invalid choices, propose restorations, alternative values, etc.
- The program maintains the cost of cheapest car consistent with the current choices.
- Upon demand, it presents the minimal and maximal costs associated to the remaining choices.
- The program shall recommend interesting values for the next variable, given the current choices and selling histories.
The full configuration process

A more complex process:

• The program presents, for each variable, values that satisfy the constraints (given the current choices), and discards the others
• The user assigns a value to some variable, or *removes a previous assignment* (without any prescribed order)
• The program should provide *explanations* for invalid choices, propose *restorations*, *alternative values*, etc
• The program maintains the *cost of cheapest car* consistent with the current choices
• Upon demand, it presents the *minimal and maximal costs* associated to the remaining choices
• The program shall *recommend interesting values* for the next variable, given the current choices and selling histories

Study non-Boolean compilation languages
Problematics

Many AI applications use functions with non-Boolean values

- cost or utility functions (e.g. in configuration problems)
- probability distributions (e.g. selling histories)
- weighted knowledge bases...
Many AI applications use functions with non-Boolean values

- cost or utility functions (e.g. in configuration problems)
- probability distributions (e.g. selling histories)
- weighted knowledge bases...

Compilation into a suitable language

- Valued CSPs, GAI-nets, Bayesian networks, weighted bases: the problem is expressed compactly, but optimization is hard
- Valued Decision Diagrams: ADD, SLDDs, AADDs (generalization of OBDDs)
- More freedom in the structure: arithmetic circuits, probabilistic sentential decision diagrams
Problematics

Many AI applications use functions with non-Boolean values

- cost or utility functions (e.g. in configuration problems)
- probability distributions (e.g. selling histories)
- weighted knowledge bases...

Compilation into a suitable language

- Valued CSPs, GAI-nets, Bayesian networks, weighted bases: the problem is expressed compactly, but optimization is hard
- Valued Decision Diagrams: ADD, SLDDs, AADDs (generalization of OBDDs)
- More freedom in the structure: arithmetic circuits, probabilistic sentential decision diagrams

This talk: Valued Decision Diagrams: KC map + experiments
Outline

Configuration and Compilation

Valued Decision Diagrams

A Compilation Map for Real Valued Decision Diagrams

Experiments
ADDs: algebraic decision diagrams [Bahar et al., 1993]

- Like OBDDs, but each leaf is a value from a set $\mathcal{V}$

\[ \mathcal{V} = \mathbb{R}^+ \]

- Optimization is trivial, Conditioning and Marginalization on one variable are easy
SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- Problem of ADDs: one leaf per value
- Idea: move values up on the arcs, so that they can be shared
- Value of a path = aggregation of encountered values

Example in configuration w.r.t. pricing function: $\mathcal{V} = \mathbb{R}^+$, aggregation by sum
→ SLDD$_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$:
aggregating by product
→ SLDD$_\times$ language → for probability distributions
SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- Problem of ADDs: one leaf per value
- Idea: move values up on the arcs, so that they can be shared
- Value of a path = aggregation of encountered values

Example in configuration w.r.t. pricing function: $V = \mathbb{R}^+$, aggregation by sum
$\rightarrow$ SLDD$_+$ language

Other possibility for $V = \mathbb{R}^+$:
aggregating by product
$\rightarrow$ SLDD$_\times$ language $\rightarrow$ for probability distributions
SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- Problem of ADDs: one leaf per value
- Idea: move values up on the arcs, so that they can be shared
- Value of a path = aggregation of encountered values

Example in configuration w.r.t. pricing function: $\mathcal{V} = \mathbb{R}^+$, aggregation by sum
$\rightarrow$ SLDD$_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product
$\rightarrow$ SLDD$_\times$ language $\rightarrow$ for probability distributions
SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- Problem of ADDs: one leaf per value
- Idea: move values up on the arcs, so that they can be shared
- Value of a path = aggregation of encountered values

Example in configuration w.r.t. pricing function: $\mathcal{V} = \mathbb{R}^+$, aggregation by sum
$\rightarrow$ SLDD$_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$:
aggregating by product
$\rightarrow$ SLDD$_\times$ language $\rightarrow$ for probability distributions
SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- Problem of ADDs: one leaf per value
- Idea: move values up on the arcs, so that they can be shared
- Value of a path = aggregation of encountered values

Example in configuration w.r.t. pricing function: $\mathcal{V} = \mathbb{R}^+$, aggregation by sum
$\rightarrow$ SLDD$_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$:
aggregating by product
$\rightarrow$ SLDD$_\times$ language $\rightarrow$ for probability distributions
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
  - two factors on each arc $a$, an additive one and a multiplicative one $\langle q, f \rangle$
- Path starting with $a$: value $q + f \times V_{\text{rec}}$, with $V_{\text{rec}}$ the value of the rest of the path

SLDD: "Red, Solar": $4 + 1 = 5$
AADD: "Red, Solar": $0 + 1 \times (1 + 1 \times (4 + 1 \times 0)) = 5$
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
- Two factors on each arc $a$, an additive one and a multiplicative one $\langle q, f \rangle$
- Path starting with $a$: value $q + f \times V_{\text{rec}}$, with $V_{\text{rec}}$ the value of the rest of the path

SLDD: "Red, Solar": $4 + 1 = 5$
AADD: "Red, Solar": $0 + 1 \cdot (1 + 1 \cdot (4 + 1.0)) = 5
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
- Two factors on each arc $a$, an additive one and a multiplicative one $\langle q, f \rangle$
- Path starting with $a$: value $q + f \times V_{\text{rec}}$, with $V_{\text{rec}}$ the value of the rest of the path

SLDD: "Red, Solar": $4 + 1 = 5$
AADD: "Red, Solar": $0 + 1.(1 + 1.(4 + 1.0)) = 5$
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
  \( \rightarrow \) two factors on each arc \( a \), an additive one and a multiplicative one \( \langle q, f \rangle \)
- Path starting with \( a \): value \( q + f \times V_{\text{rec}} \), with \( V_{\text{rec}} \) the value of the rest of the path

SLDD: "Red, Solar": \( 4 + 1 = 5 \)
AADD: "Red, Solar": 
\[ 0 + 1 \cdot (1 + 1 \cdot (4 + 1.0)) = 5 \]
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
  \[ \rightarrow \text{two factors on each arc } a, \text{ an additive one and a multiplicative one } \langle q, f \rangle \]
- Path starting with \( a \): value \( q + f \times V_{\text{rec}} \), with \( V_{\text{rec}} \) the value of the rest of the path

SLDD: "Red, Solar": 4 + 1 = 5
AADD: "Red, Solar": 0 + 1.(1 + 1.(4 + 1.0)) = 5
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
- Two factors on each arc $a$, an additive one and a multiplicative one \( \langle q, f \rangle \)
- Path starting with $a$: value $q + f \times V_{\text{rec}}$, with $V_{\text{rec}}$ the value of the rest of the path

SLDD: "Red, Solar": $4 + 1 = 5$

AADD: "Red, Solar":

$0 + 1.(1 + 1.(4 + 1.0)) = 5$
AADDs: Affine Algebraic DD [Sanner and McAllester, 2005]

- A variant of SLDD: aggregation by a combination of sum and product
  - two factors on each arc $a$, an additive one and a multiplicative one $\langle q, f \rangle$
- Path starting with $a$: value $q + f \times V_{\text{rec}}$, with $V_{\text{rec}}$ the value of the rest of the path

SLDD: "Red, Solar": $4 + 1 = 5$
AADD: "Red, Solar":
$0 + 1.(1 + 1.(4 + 1.0)) = 5$

- Normalization conditions $\rightarrow$ all paths to the leaf have value $\in [0, 1]$;
  extrema can be read on the root’s offset
Outline

Configuration and Compilation

Valued Decision Diagrams

A Compilation Map for Real Valued Decision Diagrams

Experiments
Recall that a $L$-representation $\alpha$ is a data structure that represent a function $f^L_L(\vec{x})$

- We can have a AADD, VCSP or a ADD representation of function $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} 2^{i-1} x_i$ on $\{0,1\}^n$

- Two representations $\alpha$ and $\beta$ are equivalent iff $f^L_\alpha = f^L_\beta$
The $\mathbb{R}^+$-VDDs languages

- We restrict ourselves to languages ADD on $\mathbb{R}^+$, SLDD$_+$, SLDD$_\times$ and AADD.

- All satisfy canonicity (upon normalization): equivalent sub-functions are isomorphic; caching is efficient.

- A hierarchy of languages: ADD $\subseteq$ SLDD$_+$, SLDD$_\times$ $\subseteq$ AADD
$L_1$ is at least as succinct as $L_2$, denoted $L_1 \leq_s L_2$, iff there exists a polynomial $p$ such that for every $L_2$ representation $\alpha$, there exists a $L_1$ representation $\beta$ which is equivalent to $\alpha$ and s.t. $\text{size}(\beta) \leq p(\text{size}(\alpha))$. 

\[ \text{Map for } \mathbb{R}^+\text{-VDDs: Succinctness} \]
Map for $\mathbb{R}^+$-VDDs: Succinctness

$L_1$ is at least as succinct as $L_2$, denoted $L_1 \leq_s L_2$, iff there exists a polynomial $p$ such that for every $L_2$ representation $\alpha$, there exists a $L_1$ representation $\beta$ which is equivalent to $\alpha$ and s.t. $\text{size}(\beta) \leq p(\text{size}(\alpha))$.

e.g. because the function $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} 2^{i-1} x_i$ on $\{0, 1\}^n$ maps to an exponential set of values and cannot be represented by a product.
Queries

A VDD $\alpha$ represent function $f_\alpha(\vec{x})$ taking its values in an ordered valuation scale $\mathcal{V}$ (here, $\mathcal{V} = \mathbb{R}^+$)

- Equivalence query $\text{EQ}$ similar to the Boolean case: indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) = f^L_\beta(\vec{x})$
  $\rightarrow$ are these two catalogs the same?

- Sentential entailment $\text{SE}$: given a preorder $\preceq$ on $\mathcal{V}$, indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) \preceq f^L_\beta(\vec{x})$

- A language $L$ satisfies $\text{OPT}_{\min}$ if there exists a polynomial algorithm mapping any formula $\alpha$ of $L$ to the value $\min \#x f^L_\alpha(\vec{x})$.
  $\rightarrow$ what is the price of the cheapest cars?
A VDD $\alpha$ represent function $f_\alpha(\vec{x})$ taking its values in an ordered valuation scale $\mathcal{V}$ (here, $\mathcal{V} = \mathbb{R}^+$)

- Equivalence query $\textbf{EQ}$ similar to the Boolean case: indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) = f^L_\beta(\vec{x})$  
  $\rightarrow$ are these two catalogs the same?

- Sentential entailment $\textbf{SE}$: given a preorder $\preceq$ on $\mathcal{V}$, indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) \preceq f^L_\beta(\vec{x})$  
  $\rightarrow$ Is this e-shop always cheaper than this other one?
Queries

A VDD $\alpha$ represent function $f_\alpha(\vec{x})$ taking its values in an ordered valuation scale $\mathcal{V}$ (here, $\mathcal{V} = \mathbb{R}^+$)

- Equivalence query **EQ** similar to the Boolean case: indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) = f^L_\beta(\vec{x})$
  $\rightarrow$ are these two catalogs the same?

- Sentential entailment **SE**: given a preorder $\preceq$ on $\mathcal{V}$, indicating whether $\forall \vec{x}, f^L_\alpha(\vec{x}) \preceq f^L_\beta(\vec{x})$
  $\rightarrow$ Is this e-shop always cheaper than this other one?

- A language $L$ satisfies **OPT$_{\min}$** if there exists a polynomial algorithm mapping any formula $\alpha$ of $L$ to the value $\min_{\vec{x}} f^L_\alpha(\vec{x})$.
  $\rightarrow$ what is the price of the cheapest cars?
Queries on cuts

Many of the other queries are based on cuts.

Let \( f \) be a \( \mathcal{V} \)-valued function, \( \preceq \) a preorder on \( \mathcal{V} \), and \( \gamma \in \mathcal{V} \); we define the following sets:

- \( \text{CUT}^{\preceq \gamma}(f) = \{ \vec{x} | f(\vec{x}) \preceq \gamma \} \rightarrow \text{cars cheaper than 10,000 euros} \)

- \( \text{CUT}^{\sim \gamma}(f) = \{ \vec{x} | f(\vec{x}) \sim \gamma \} \rightarrow \text{cars costing exactly 10,000 euros} \)

- \( \text{CUT}^{\min}(f) = \{ \vec{x}^* | \forall \vec{x}, \neg(f(\vec{x}) \prec f(\vec{x}^*)) \} \rightarrow \text{the cheapest cars} \)
Queries on cuts

Cut \approx \text{set of “models”}

- **CT_{\text{min}}**: counting minimal elements for \preceq (i.e., returning the cardinal of $CUT_{\text{min}}(f^L_\alpha)$)
  \rightarrow \text{how many cheapest configurations?}

- Partial consistency $CO_{\sim \gamma}$: indicating whether $\exists \vec{x}, f^L_\alpha(\vec{x}) \sim \gamma$ (i.e., whether $CUT_{\sim \gamma}(f^L_\alpha) \neq \emptyset$)
  \rightarrow \text{is there a car costing exactly 10 000 euros?}

- **MX_{\preceq \gamma}, ME_{\preceq \gamma}**: exhibiting an $\vec{x}$, enumerating all $\vec{x}$ such that $f^L_\alpha(\vec{x}) \preceq \gamma$
  \rightarrow \text{which cars are cheaper than 10 000 euros?}

\ldots and the other combinations
Map for queries

<table>
<thead>
<tr>
<th>Query</th>
<th>ADD</th>
<th>SLDD⁺</th>
<th>SLDD⊗</th>
<th>AADD</th>
<th>VCSP⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
</tr>
<tr>
<td>SE</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>?</td>
<td>○</td>
</tr>
<tr>
<td>OPTₘin</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>○</td>
</tr>
<tr>
<td>MXₘin / MEₘin</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>○</td>
</tr>
<tr>
<td>CTₘin</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>○</td>
</tr>
<tr>
<td>CO∼γ / MX∼γ / ME∼γ</td>
<td>√</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>CO≤γ / MX≤γ / ME≤γ</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>○</td>
</tr>
<tr>
<td>CT∼γ / CT≤γ</td>
<td>√</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

- ADD satisfies all queries
- SLDD⁺, SLDD⊗, and AADD behave the same on queries
- Queries on optimal cuts are easy
- Counting is hard on γ-cuts
- All queries on exact γ-cuts are hard (red. from SUBSET SUM)
Cut transformations

A language $L$ satisfies a transformation if there exists a polynomial algorithm performing it while staying in $L$

Given a $L$ representation $\alpha$ of $f$, we want a $L$ representation of a cut of $f$:

- $\text{CUT}_{\text{min}}$: compute a $L$ representation of the set of cheapest cars

- $\text{CUT}_{\leq \gamma}$: compute a $L$ representing the set of cars are cheaper than 10 000 euros

- $\text{CUT}_{\sim \gamma}$: compute a $L$ representing the set of cars costing exactly 10 000 euros
On ADD, $\text{CUT}_{\min}, \text{CUT}_{\leq \gamma}, \text{CUT}_{\sim \gamma}$, etc. are trivial:

this is why ADD satisfies all queries related to cuts.
Cut-based transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ADD</th>
<th>SLDD⁺</th>
<th>SLDD⁻</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUT_{min}</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>CUT_{∼γ}</td>
<td>√</td>
<td>⋄</td>
<td>⋄</td>
<td>⋄</td>
</tr>
<tr>
<td>CUT_{≤γ}</td>
<td>√</td>
<td>⋄</td>
<td>⋄</td>
<td>⋄</td>
</tr>
</tbody>
</table>

- **Cutting to the optimum is easy**, even on SLDD and AADD: after normalizing, the minimal paths are those in which all arcs have factor 0
- **Cutting w.r.t. a threshold is not polynomial** (it may require a complete unfolding of the structure)
Conditioning and Combinations

Conditioning $\mathbf{CD}$ defined as in the Boolean case

The other transformations are parameterized by an associative and commutative binary operator $\odot$ on $\mathcal{V}$

- $\odot\mathbf{C}$: combining $n$ formulas by $\odot$ (i.e., building a formula in $\Lambda$ representing the function $\bigodot_{i=1}^{n} f_{\alpha_i}^\Lambda$)
  - $+\mathbf{C} \times\mathbf{C}$: useful for bottom un compilation
- $\odot\mathbf{BC}$: combining a bounded number of $\Lambda$ representations
  - $\times\mathbf{BC}$
    - making a discount
  - $\min\mathbf{BC}$
    - choosing in two catalogs
Map for transformations: combinations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ADD</th>
<th>SLDD⁺</th>
<th>SLDD⁻</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>minC / +C / ×C</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>minBC</td>
<td>√</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>+BC</td>
<td>√</td>
<td>√</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>×BC</td>
<td>√</td>
<td>⬤</td>
<td>√</td>
<td>⬤</td>
</tr>
</tbody>
</table>

- ADD satisfies all bounded combinations
  → “apply” algorithm, similar to OBDDs
- SLDD⁺ satisfies the combination by +
- SLDD⁻ satisfies the combination by ×
  → the “apply” algorithm also works because the operators are associative and commutative
Map for transformations: combinations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ADD</th>
<th>SLDD_+</th>
<th>SLDD_×</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>minC / +C / ×C</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>minBC</td>
<td>✓</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>+BC</td>
<td>✓</td>
<td>✓</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td>×BC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>⬤</td>
</tr>
</tbody>
</table>

- SLDD\_+ does not satisfy the combination by ×: consider the function $f(\vec{x}) = \sum_{i=0}^{n-1} x_i \cdot 2^i$ and $g(\vec{x}) = 2^{n+1} - 1 - f(\vec{x})$; linear SLDD\_+ representation, but $f \times g$ has only exponential SLDD\_+ representations

- SLDD\_× does not satisfy the combination by +: similar proof

- AADD does not satisfy any bounded combination.
Transformations: variable elimination

- ⊙**Elim**, elimination of variables $Y$ w.r.t. ⊙: building a formula in $L$
  representing $\bigodot_{\not{Y}} f^L_\alpha | \not{Y}$
  $\rightarrow$ e.g., forgetting $=$ max-elimination

- ⊙**Marg**, marginalization on a single variable w.r.t. ⊙: eliminating all variables but one
  $\rightarrow$ $+$-marginalization on a variable in Bayesian networks
### Map for transformations: marginalization

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ADD</th>
<th>SLDD$_+$</th>
<th>SLDD$_\times$</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min\text{Marg}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\text{+Marg}$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>$\times\text{Marg}$</td>
<td>✓</td>
<td>?</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>

Marginalization is easy when the elimination of the last variable can be done in linear time.

Works for $\text{+Marg}$ on SLDD$\_\times$ and AADD basically because multiplication distributes over addition.

→ does not work for $\times\text{Marg}$ on SLDD$\_+$ and AADD.
Map for transformations: Variable Elimination

No language satisfies any elimination, even of a single variable, as long as its domain is unbounded.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>ADD</th>
<th>SLDD$_+$</th>
<th>SLDD$_\times$</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>minElim/ +Elim / ×Elim</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SminElim / S+Elim / S×Elim</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SBmaxElim / SBminElim</td>
<td>✓</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SB+Elim</td>
<td>✓</td>
<td>✓</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>SB×Elim</td>
<td>✓</td>
<td>•</td>
<td>✓</td>
<td>•</td>
</tr>
</tbody>
</table>

$S\odot{\text{Elim}}$: eliminating a single variable

$SB\odot{\text{Elim}}$: eliminating a single bounded-domain variable
Summary

- Conditionning and Optimization satisfied on AADD, SLDD⁺, SLDDₓ, ADD
- \textbf{minBC} satisfied on ADD only
- AADD "more succinct" than SLDD⁺, SLDDₓ, themselves "more succinct" than ADD
- \textbf{+BC} ok on SLDD⁺ and ADD only
Outline

Configuration and Compilation

Valued Decision Diagrams

A Compilation Map for Real Valued Decision Diagrams

Experiments
On the practical succinctness of valued decision diagrams

- Design of a bottom-up ordered SLDD\(_+\) SLDD\(_\times\) compiler.
  - Input: VCSP instance (XML format) or Bayesian Nets (XML format).
  - Output: an equivalent SLDD\(_+\) / SLDD\(_\times\).

- Test of a large set of variable ordering heuristics.

- Design of toolbox of transformation procedures (that are basically normalization procedures)
  - SLDD\(_+\) (resp. SLDD\(_\times\)) to ADD
  - ADD to SLDD\(_+\), SLDD\(_\times\)
  - SLDD\(_+\) (resp. SLDD\(_\times\)) to AADD
Benchmark tested

Two families of benchmarks.

- VCSP instances encoding car configurations problems with pricing functions
  - Small: #variables=139; max. domain size=16; #constraints=176 (including 29 soft constraints)
  - Medium: #variables=148; max. domain size=20; #constraints=268 (including 94 soft constraints)
  - Big: #variables=268; max. domain size=324; #constraints=2157 (including 1825 soft constraints)

- Bayesian networks: Cancer, Asia, Car-starts, Alarm, Hailfinder25
### Heuristics

**MCS = Maximum Cardinality Search heuristic**

[Tarjan and Yannakakis, 1984] in reverse order

<table>
<thead>
<tr>
<th>Instance</th>
<th>MCF</th>
<th>Band-Width</th>
<th>MCS</th>
<th>Force</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nodes</td>
<td>cpu</td>
<td>nodes</td>
<td>cpu</td>
</tr>
<tr>
<td>VCSP (\mapsto) SLDD⁺</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 100</td>
<td>1.2s</td>
<td>4 349</td>
<td>1.0s</td>
</tr>
<tr>
<td></td>
<td>5 660</td>
<td>1.5s</td>
<td>11 700</td>
<td>1.6s</td>
</tr>
<tr>
<td></td>
<td>m-o</td>
<td>-</td>
<td>326 884</td>
<td>112s</td>
</tr>
<tr>
<td>Medium</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 700</td>
<td>1.5s</td>
<td>11 700</td>
<td>1.6s</td>
</tr>
<tr>
<td></td>
<td>242 603</td>
<td>1.4s</td>
<td>242 603</td>
<td>1.4s</td>
</tr>
<tr>
<td>Big</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>326 884</td>
<td>112s</td>
<td>326 884</td>
<td>112s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bayes (\mapsto) SLDDₓ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asia</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>0.06s</td>
<td>29</td>
<td>0.06s</td>
</tr>
<tr>
<td></td>
<td>60</td>
<td>0.1s</td>
<td>40</td>
<td>0.09s</td>
</tr>
<tr>
<td></td>
<td>m-o</td>
<td>-</td>
<td>5 843</td>
<td>0.8s</td>
</tr>
<tr>
<td>Car-starts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>0.06s</td>
<td>29</td>
<td>0.06s</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>0.09s</td>
<td>40</td>
<td>0.09s</td>
</tr>
<tr>
<td></td>
<td>m-o</td>
<td>-</td>
<td>5 843</td>
<td>0.8s</td>
</tr>
<tr>
<td>Alarm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 333</td>
<td>1.3s</td>
<td>15 333</td>
<td>1.3s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 333</td>
<td>1.3s</td>
<td>15 333</td>
<td>1.3s</td>
</tr>
<tr>
<td>finder25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MCS = Maximum Cardinality Search heuristic**

[Tarjan and Yannakakis, 1984] in reverse order
Practical Succinctness

<table>
<thead>
<tr>
<th>Instance</th>
<th>SLDD&lt;sub&gt;+&lt;/sub&gt;</th>
<th>ADD</th>
<th>SLDD&lt;sub&gt;×&lt;/sub&gt;</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>nodes</td>
<td>temps</td>
<td>nodes</td>
<td>nodes</td>
</tr>
<tr>
<td>Medium</td>
<td>1744</td>
<td>0,9s</td>
<td>28 971</td>
<td>19 930</td>
</tr>
<tr>
<td>Big</td>
<td>73 702</td>
<td>34s</td>
<td>463 383</td>
<td>m-o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rés. bay.</th>
<th>SLDD&lt;sub&gt;×&lt;/sub&gt;</th>
<th>ADD</th>
<th>SLDD&lt;sub&gt;+&lt;/sub&gt;</th>
<th>AADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia</td>
<td>nodes</td>
<td>temps</td>
<td>nodes</td>
<td>nodes</td>
</tr>
<tr>
<td>Car-starts</td>
<td>23</td>
<td>0,07s</td>
<td>415</td>
<td>216</td>
</tr>
<tr>
<td>Alarm</td>
<td>1301</td>
<td>0,5s</td>
<td>42 741</td>
<td>m-o</td>
</tr>
<tr>
<td>Hailfinder25</td>
<td>15 333</td>
<td>1,8s</td>
<td>m-o</td>
<td>m-o</td>
</tr>
</tbody>
</table>

- AADD, SLDD<sub>+</sub>, SLDD<sub>×</sub> < ADD;
- AADD < SLDD<sub>+</sub>, SLDD<sub>×</sub> but not so much:
  - AADD and SLDD<sub>+</sub> comparable on additive pricing functions,
  - AADD and SLDD<sub>×</sub> comparable on bayesian nets (multiplicative)
On line use : $CD + \text{ marginalization on each variable}$

<table>
<thead>
<tr>
<th></th>
<th>$\text{SLDD}_+$</th>
<th>$\text{AADD}$</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>$222 \mu s$</td>
<td>$281 \mu s$</td>
<td>1,27</td>
</tr>
<tr>
<td>Medium</td>
<td>$487 \mu s$</td>
<td>$578 \mu s$</td>
<td>1,19</td>
</tr>
<tr>
<td>Big</td>
<td>$22,1 ms$</td>
<td>$39,9 ms$</td>
<td>1,81</td>
</tr>
<tr>
<td>Bayes</td>
<td>$\text{SLDD}_\times$</td>
<td>$\text{AADD}$</td>
<td>ratio</td>
</tr>
<tr>
<td>Asia</td>
<td>$29,0 \mu s$</td>
<td>$32,3 \mu s$</td>
<td>1,11</td>
</tr>
<tr>
<td>Car-starts</td>
<td>$61,5 \mu s$</td>
<td>$75,6 \mu s$</td>
<td>1,23</td>
</tr>
<tr>
<td>Alarm</td>
<td>$259 \mu s$</td>
<td>$292 \mu s$</td>
<td>1,13</td>
</tr>
<tr>
<td>Hailfinder25</td>
<td>$7,68 ms$</td>
<td>$9,16 ms$</td>
<td>1,19</td>
</tr>
</tbody>
</table>

$\text{SLDD}$ is more efficient: less number manipulations ($\text{AADD}$ makes many unsuccessful attempts of saving space), less rounding errors
On line use : \text{CD} + marginalization on each variable

\textbf{Figure}: Average and maximal time (ms) for conditionning + marginalization on the \textbf{big} car configuration instance.
On line use : full configuration process (without prices)

**Figure:** Average time (ms) for conditionning + marginalization on the **big** car configuration instance.
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: \( R^+ \)-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)

To Do / Further Research:

- Complete the KC map: Arithmetic circuits, V-A OMDD, Sentential Networks (ideally an Algebraic map)
- Application of AADD to problems that need their full power
- Learning preferences: SLDD \times, Bayesian nets, SDDs
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: \(R^+\)-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
- Experimental results may contrast with theoretical ones on some instances (SLDD+ vs. AADD)
  - Do not necessarily "recompile" on line: fusion of isomorphic nodes, determinism are not compulsory
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
- Experimental results may contrast with theoretical ones on some instances (SLDD+ vs. AADD)
  - Do not necessarily "recompile" on line: fusion of isomorphic nodes, determinism are not compulsory

To Do / Further Research

- Complete the KC map: Arithmetic circuits, V-AOMDD, Sentential Networks (ideally an Algebraic map)
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
- Experimental results may contrast with theoretical ones on some instances (SLDD vs. AADD)
  - Do not necessarily "recompile" on line: fusion of isomorphic nodes, determinism are not compulsory

To Do / Further Research

- Complete the KC map: Arithmetic circuits, V-AOMDD, Sentential Networks (ideally an Algebraic map)
- Application of AADD to problems that need their full power
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
- Experimental results may contrast with theoretical ones on some instances (SLDD$_+$ vs. AADD)
  - Do not necessarily "recompile" on line: fusion of isomorphic nodes, determinism are not compulsory

To Do / Further Research

- Complete the KC map: Arithmetic circuits, V-AOMDD, Sentential Networks (ideally an Algebraic map)
- Application of AADD to problems that need their full power
- Learning preferences: SLDD$_\times$, Bayesian nets, SDDs
Conclusion and perspectives

Done:

- Premisses of a KC map of non-Boolean functions (here: $R^+$-valued functions)
- SLDD: implementation of a compiler + a toolbox (SALADD)
- Very efficient on our configuration problems
- Experimental results may contrast with theoretical ones on some instances (SLDD$_+$ vs. AADD)
  - Do not necessarily "recompile" on line: fusion of isomorphic nodes, determinism are not compulsory

To Do / Further Research

- Complete the KC map: Arithmetic circuits, V-AOMDD, Sentential Networks (ideally an Algebraic map)
- Application of AADD to problems that need their full power
- Learning preferences: SLDD$_\times$, Bayesian nets, SDDs
Bibliography

Consistency restoration and explanations in dynamic CSPs: Application to configuration.

Algebraic decision diagrams and their applications.

A knowledge compilation map.
*Journal of Artificial Intelligence Research (JAIR)*, 17:229–264.

Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues.
In *Proceedings of IJCAI’13*.

A knowledge compilation map for ordered real-valued decision diagrams.
In *Proceedings of AAAI’2014*.
Accepté pour publication.

A bdd-based approach to interactive configuration.
In *Proceedings of CP’04*, page 797.

Probabilistic sentential decision diagrams.
[Fargier et al., 2014, Fargier et al., 2013] [Kisa et al., 2014]