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

Hélène Fargier[1] Pierre Marquis[2]
Alexandre Niveau[3] Nicolas Schmidt[1,2]

[1] IRIT-CNRS, Univ. Paul Sabatier, Toulouse, France

[2] CRIL-CNRS, Univ. Artois, Lens, France

[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

# 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 $\rightarrow$ constraint satisfaction problem (CSP)
  - Configuration parameter = a CSP variable (finite domain)
  - Constraints

$$\begin{cases} \mathit{motor} = \text{pedals} & \rightarrow & \mathit{color} = \text{red} \\ \mathit{motor} = \text{solar} & \rightarrow & \mathit{size} > \text{twoseater} \\ \mathit{size} = \text{twoseater} & \vee & \mathit{radio} = \text{with} \end{cases}$$

  - each solution corresponds to an admissible configuration

# Basic Problem

- Configurable product $\rightarrow$ constraint satisfaction problem (CSP)
  - Configuration parameter = a CSP variable (finite domain)
  - Constraints

$$\begin{cases} \mathit{motor} = \text{pedals} & \rightarrow & \mathit{color} = \text{red} \\ \mathit{motor} = \text{solar} & \rightarrow & \mathit{size} > \text{twoseater} \\ \mathit{size} = \text{twoseater} & \vee & \mathit{radio} = \text{with} \end{cases}$$

  - 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 → constraint satisfaction problem (CSP)
  - Configuration parameter = a CSP variable (finite domain)
  - Constraints

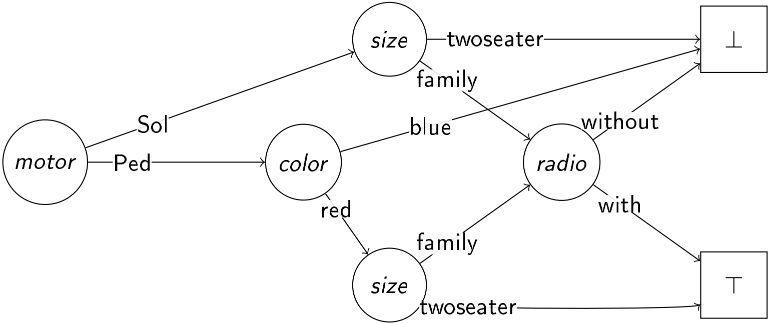
$$\begin{cases} \mathit{motor} = \text{pedals} & \rightarrow & \mathit{color} = \text{red} \\ \mathit{motor} = \text{solar} & \rightarrow & \mathit{size} > \text{twoseater} \\ \mathit{size} = \text{twoseater} & \vee & \mathit{radio} = \text{with} \end{cases}$$

  - 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 ?
- NP-complete problem . . . 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

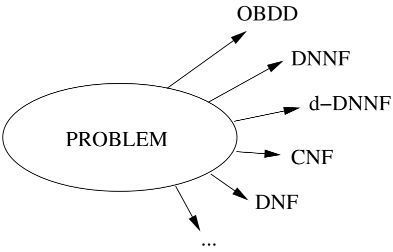
# Configuration and Compilation

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 langages according to two criteria:
  1. efficiency of operations
  2. succinctness

# Compilation map: operations

- All online manipulations amount to elementary queries and transformations

| L | CO (consistency) | VA (validity) | CE (clause entailmt.) | IM (implicant check) | EQ (equivalence) | SE (entailment) | CT (model count) | ME (model enum.) |
|---|---|---|---|---|---|---|---|---|
| NNF | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |
| DNNF | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\surd$ |
| BDD | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |
| FBDD | $\surd$ | $\surd$ | $\surd$ | $\surd$ | ? | $\circ$ | $\surd$ | $\surd$ |
| OBDD | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ | $\surd$ | $\surd$ |
| DNF | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\surd$ |
| CNF | $\circ$ | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |

| L | CD (conditioning) | FO (forgetting) | SFO (single forg.) | $\wedge$C (conjunction) | $\wedge$BC (bounded conj.) | $\vee$C (disjunction) | $\vee$BC (bounded disj.) | $\neg$C (negation) |
|---|---|---|---|---|---|---|---|---|
| NNF | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| DNNF | $\surd$ | $\surd$ | $\surd$ | $\circ$ | $\circ$ | $\surd$ | $\surd$ | $\circ$ |
| BDD | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| FBDD | $\surd$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\surd$ |
| OBDD | $\surd$ | $\bullet$ | $\surd$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\surd$ |
| DNF | $\surd$ | $\surd$ | $\surd$ | $\bullet$ | $\surd$ | $\surd$ | $\surd$ | $\bullet$ |
| CNF | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\bullet$ | $\surd$ | $\bullet$ |

$\surd$ polynomial
$\circ$ not polynomial, unless $P = NP$
$\bullet$ not polynomial

# Compilation map: operations

- All online manipulations amount to elementary queries and transformations

| L | CO (consistency) | VA (validity) | CE (clause entailmt.) | IM (implicant check) | EQ (equivalence) | SE (entailment) | CT (model count) | ME (model enum.) |
|---|---|---|---|---|---|---|---|---|
| NNF | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |
| DNNF | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\surd$ |
| BDD | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |
| FBDD | $\surd$ | $\surd$ | $\surd$ | $\surd$ | ? | $\circ$ | $\surd$ | $\surd$ |
| OBDD | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ | $\surd$ | $\surd$ |
| DNF | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ | $\surd$ |
| CNF | $\circ$ | $\surd$ | $\circ$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |

| L | CD (conditioning) | FO (forgetting) | SFO (single forg.) | $\wedge$C (conjunction) | $\wedge$BC (bounded conj.) | $\vee$C (disjunction) | $\vee$BC (bounded disj.) | $\neg$C (negation) |
|---|---|---|---|---|---|---|---|---|
| NNF | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| DNNF | $\surd$ | $\surd$ | $\surd$ | $\circ$ | $\circ$ | $\surd$ | $\surd$ | $\circ$ |
| BDD | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| FBDD | $\surd$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\surd$ |
| OBDD | $\surd$ | $\bullet$ | $\surd$ | $\bullet$ | $\circ$ | $\bullet$ | $\circ$ | $\surd$ |
| DNF | $\surd$ | $\surd$ | $\surd$ | $\bullet$ | $\surd$ | $\surd$ | $\surd$ | $\bullet$ |
| CNF | $\surd$ | $\circ$ | $\surd$ | $\surd$ | $\surd$ | $\bullet$ | $\surd$ | $\bullet$ |

$\surd$ polynomial
$\circ$ not polynomial, unless $P = NP$
$\bullet$ not polynomial

# Compilation map: operations

- All online manipulations amount to elementary queries and transformations

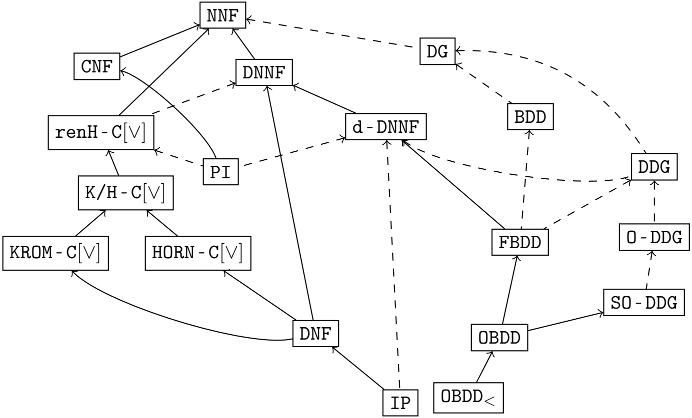
| L | CO (consistency) | VA (validity) | CE (clause entailmt.) | IM (implicant check) | EQ (equivalence) | SE (entailment) | CT (model count) | ME (model enum.) |
|---|---|---|---|---|---|---|---|---|
| NNF | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| DNNF | √ | ○ | √ | ○ | ○ | ○ | ○ | √ |
| BDD | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| FBDD | √ | √ | √ | √ | ? | ○ | √ | √ |
| OBDD | √ | √ | √ | √ | √ | ○ | √ | √ |
| DNF | √ | ○ | √ | ○ | ○ | ○ | ○ | √ |
| CNF | ○ | √ | ○ | √ | ○ | ○ | ○ | ○ |

| L | CD (conditioning) | FO (forgetting) | SFO (single forg.) | $\wedge$C (conjunction) | $\wedge$BC (bounded conj.) | $\vee$C (disjunction) | $\vee$BC (bounded disj.) | $\neg$C (negation) |
|---|---|---|---|---|---|---|---|---|
| NNF | √ | ○ | √ | √ | √ | √ | √ | √ |
| DNNF | √ | √ | √ | ○ | ○ | √ | √ | ○ |
| BDD | √ | ○ | √ | √ | √ | √ | √ | √ |
| FBDD | √ | • | ○ | • | ○ | • | ○ | √ |
| OBDD | √ | • | √ | • | ○ | • | ○ | √ |
| DNF | √ | √ | √ | • | √ | √ | √ | • |
| CNF | √ | ○ | √ | √ | √ | • | √ | • |

√ polynomial
○ not polynomial, unless P = NP
• not polynomial

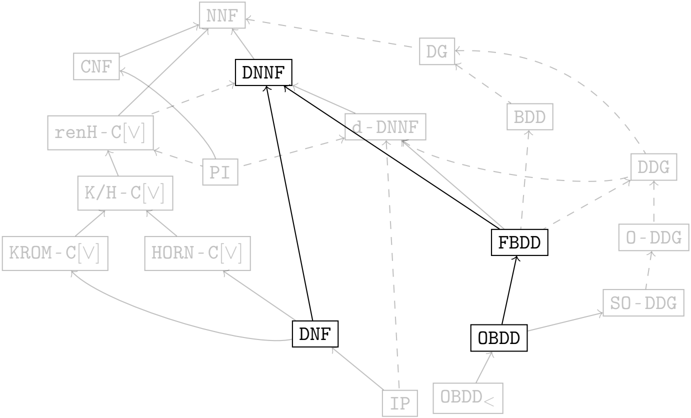
# Compilation map: succinctness

- Succinctness relation ($\leq_s$): orders languages
- $\mathtt{L}_1 \leq_s \mathtt{L}_2$ means "$\mathtt{L}_1$ is at least as succinct as $\mathtt{L}_2$"



# Compilation map: succinctness

- Succinctness relation ($\leq_s$): orders languages
- $\mathtt{L}_1 \leq_s \mathtt{L}_2$ means "$\mathtt{L}_1$ is at least as succinct as $\mathtt{L}_2$"



# 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 ny prescribed order)

# 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 ny prescribed order)
- The programm 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 ny prescribed order)
- The programm 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 ny prescribed order)
- The programm 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 deman, it present 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 ny prescribed order)
- The programm 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 deman, it present the minimal and maximal costs associated to the remaining choices
- The programm 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 ny prescribed order)
- The programm 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 deman, it present the minimal and maximal costs associated to the remaining choices
- The programm 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. . .

# 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

# 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

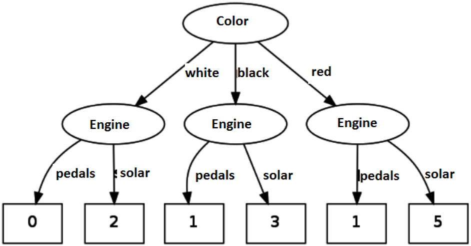
Valued Decision Diagrams

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 $\mathcal{V}$




$(\mathcal{V} = \mathbb{R}^+)$

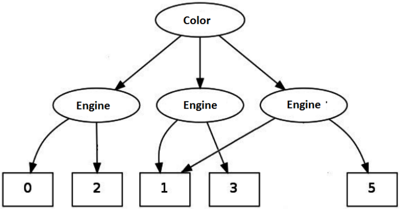
- Optimization is trivial, Conditionning and Marginalization on one variable are easy

# 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
$\rightarrow$ $\texttt{SLDD}_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product
$\rightarrow$ $\texttt{SLDD}_\times$ language $\rightarrow$ for probability distributions

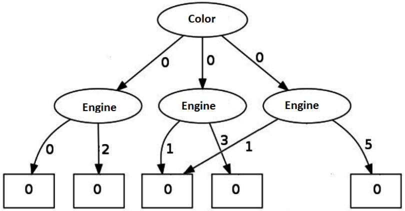


# SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- 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
$\rightarrow$ $\texttt{SLDD}_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product
$\rightarrow$ $\texttt{SLDD}_\times$ language $\rightarrow$ for probability distributions

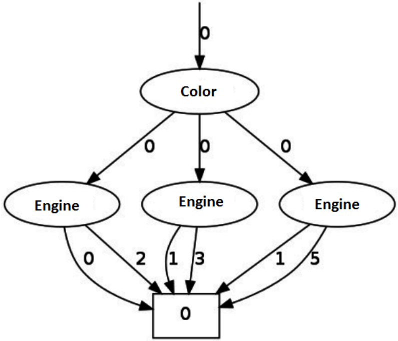


# SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- 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 $\rightarrow$ $\mathtt{SLDD}_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product $\rightarrow$ $\mathtt{SLDD}_\times$ language $\rightarrow$ for probability distributions

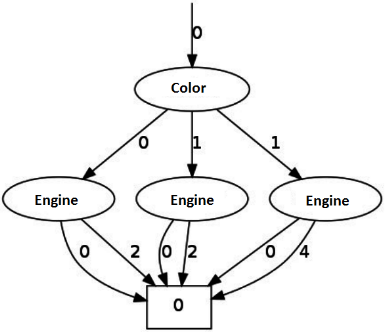


# SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- 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 $\rightarrow$ $\texttt{SLDD}_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product $\rightarrow$ $\texttt{SLDD}_\times$ language $\rightarrow$ for probability distributions

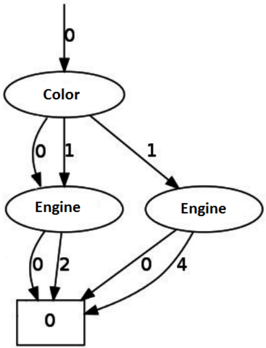


# SLDDs: semiring-labeled decision diagrams [Wilson, 2005]

- 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
$\rightarrow$ $\mathtt{SLDD}_+$ language

Other possibility for $\mathcal{V} = \mathbb{R}^+$: aggregating by product
$\rightarrow$ $\mathtt{SLDD}_\times$ language $\rightarrow$ for probability distributions



# AADDs: Affine Agebraic DD [Sanner and McAllester, 2005]

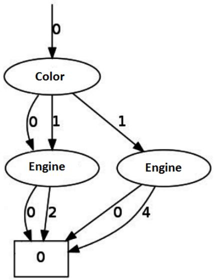
- A variant of SLDD: aggregation by a combination of sum and product

→ 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 Agebraic DD [Sanner and McAllester, 2005]

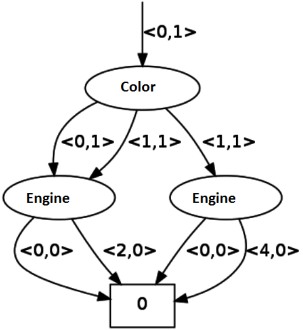
- A variant of SLDD: aggregation by a combination of sum and product

→ 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 Agebraic DD [Sanner and McAllester, 2005]

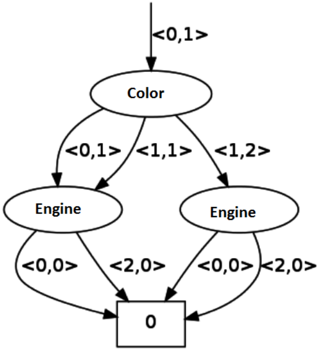
- A variant of SLDD: aggregation by a combination of sum and product

$\rightarrow$ 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": $0 + 1.(1 + 1.(4 + 1.0)) = 5$



# AADDs: Affine Agebraic DD [Sanner and McAllester, 2005]

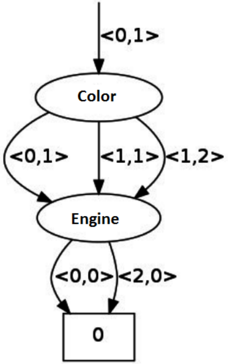
- A variant of SLDD: aggregation by a combination of sum and product

→ 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 Agebraic DD [Sanner and McAllester, 2005]

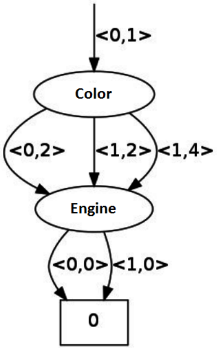
- A variant of SLDD: aggregation by a combination of sum and product

→ two factors on each arc $a$, an additive one and a multiplicative one $\langle q, f \rangle$

- Path starting with $a$ : value $q + f \times V_{\mathrm{rec}}$, with $V_{\mathrm{rec}}$ 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 Agebraic DD [Sanner and McAllester, 2005]

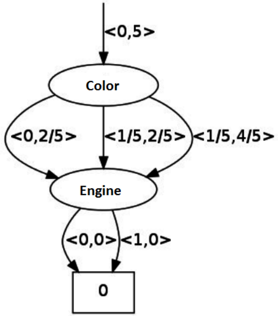
- A variant of SLDD: aggregation by a combination of sum and product

→ 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 Agebraic DD [Sanner and McAllester, 2005]

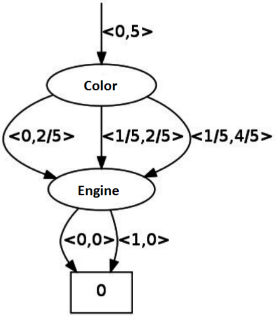
- A variant of SLDD: aggregation by a combination of sum and product

$\rightarrow$ 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":
$0 + 1.(1 + 1.(4 + 1.0)) = 5$




- 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

# The $\mathbb{R}^+$-VDDs languages

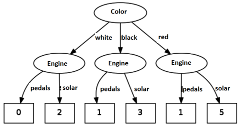
Recall that a $L$-representation $\alpha$ is a data structure that represent a function $f^L_\alpha(\vec{x})$

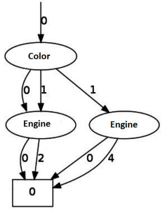
- We ca have a `AADD`, `VCSP` or a `ADD` representation of function $f(x_1, \ldots, x_n) = \Sigma_{i=1,n} 2^{i-1} x_i$ on $\{0,1\}^n$

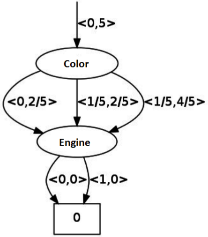
- Two representations $\alpha$ and $\beta$ are equivalent iff $f^L_\alpha = f^{L'}_\beta$

# The $\mathbb{R}^+$-VDDs languages

- We restrict ourselves to languages ADD on $\mathbb{R}^+$, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$ and AADD.
- All satisfy canonicity (upon normalization) : equivalent sub-functions are isomorphic ; caching is efficient.
- A hierarchy of languages : $\texttt{ADD} \sqsubseteq \texttt{SLDD}_+, \texttt{SLDD}_\times \sqsubseteq \texttt{AADD}$



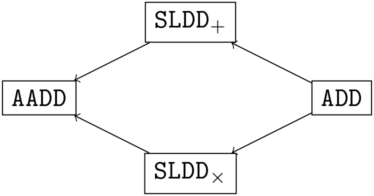



# Map for $\mathbb{R}^+$-VDDs: Succinctness

$L_1$ is *at least as succinct as* $L_2$, denoted $\mathcal{L}_1 \leq_s \mathcal{L}_2$, 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. $size(\beta) \leq p(size(\alpha))$.

# Map for $\mathbb{R}^+$-VDDs: Succinctness

$L_1$ is *at least as succinct as* $L_2$, denoted $\mathcal{L}_1 \leq_s \mathcal{L}_2$, 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. $size(\beta) \leq p(size(\alpha))$.




e.g. because the function $f(x_1, \ldots, x_n) = \Sigma_{i=1,n} 2^{i-1} x_i$ 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 **EQ** similar to the Boolean case: indicating whether $\forall \vec{x}, f_\alpha^{\mathrm{L}}(\vec{x}) = f_\beta^{\mathrm{L}}(\vec{x})$
  $\rightarrow$ are these two catalogs the same?

# 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 $\forall \vec{x}, f_\alpha^{\mathrm{L}}(\vec{x}) = f_\beta^{\mathrm{L}}(\vec{x})$
  → are these two catalogs the same?
- Sentential entailment **SE**: given a preorder $\preceq$ on $\mathcal{V}$, indicating whether $\forall \vec{x}, f_\alpha^{\mathrm{L}}(\vec{x}) \succeq f_\beta^{\mathrm{L}}(\vec{x})$
  → 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 $\forall \vec{x}, f_\alpha^{\mathtt{L}}(\vec{x}) = f_\beta^{\mathtt{L}}(\vec{x})$
  $\rightarrow$ are these two catalogs the same?
- Sentential entailment **SE**: given a preorder $\preceq$ on $\mathcal{V}$, indicating whether $\forall \vec{x}, f_\alpha^{\mathtt{L}}(\vec{x}) \succeq f_\beta^{\mathtt{L}}(\vec{x})$
  $\rightarrow$ Is this e-shop always cheaper than this other one?
- A language $\mathtt{L}$ satisfies $\mathbf{OPT}_{\mathsf{min}}$ if there exists a polynomial algorithm mapping any formula $\alpha$ of $\mathtt{L}$ to the value $\min_{\vec{x}} f_\alpha^{\mathtt{L}}(\vec{x})$.
  $\rightarrow$ 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:

- $CUT^{\preceq\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \preceq \gamma \}$
  $\rightarrow$ cars cheaper than 10 000 euros
- $CUT^{\sim\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \sim \gamma \}$
  $\rightarrow$ cars costing exactly 10 000 euros
- $CUT^{\min}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \prec f(\vec{x}^*)) \}$
  $\rightarrow$ the cheapest cars

# Queries on cuts

Cut $\approx$ set of "models"

- $\mathbf{CT}_{\mathsf{min}}$: counting minimal elements for $\preceq$ (i.e., returning the cardinal of $CUT^{\mathsf{min}}(f_\alpha^{\mathrm{L}})$)
  $\rightarrow$ how many cheapest configurations?
- Partial consistency $\mathbf{CO}_{\sim\gamma}$: indicating whether $\exists\vec{x}, f_\alpha^{\mathrm{L}}(\vec{x}) \sim \gamma$ (i.e., whether $CUT^{\sim\gamma}(f_\alpha^{\mathrm{L}}) \neq \varnothing$)
  $\rightarrow$ is there a car costing exactly 10 000 euros?
- $\mathbf{MX}_{\preceq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$: exhibiting an $\vec{x}$, enumerating all $\vec{x}$ such that $f_\alpha^{\mathrm{L}}(\vec{x}) \preceq \gamma$
  $\rightarrow$ which cars are cheaper than 10 000 euros?

. . . and the other combinations

# Map for queries

| Query | ADD | $SLDD_+$ | $SLDD_\times$ | AADD | $VCSP_+$ |
|---|---|---|---|---|---|
| **EQ** | $\surd$ | $\surd$ | $\surd$ | $\surd$ | ? |
| **SE** | $\surd$ | $\surd$ | $\surd$ | ? | $\circ$ |
| $\mathbf{OPT}_{\min}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ |
| $\mathbf{MX}_{\min}$ / $\mathbf{ME}_{\min}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ |
| $\mathbf{CT}_{\min}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ |
| $\mathbf{CO}_{\sim\gamma}$ / $\mathbf{MX}_{\sim\gamma}$ / $\mathbf{ME}_{\sim\gamma}$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |
| $\mathbf{CO}_{\preceq\gamma}$ / $\mathbf{MX}_{\preceq\gamma}$ / $\mathbf{ME}_{\preceq\gamma}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ | $\circ$ |
| $\mathbf{CT}_{\sim\gamma}$ / $\mathbf{CT}_{\preceq\gamma}$ | $\surd$ | $\circ$ | $\circ$ | $\circ$ | $\circ$ |

- ADD satisfies all queries
- $SLDD_+$, $SLDD_\times$, and AADD behave the same on queries
- Queries on optimal cuts are easy
- Counting is hard on $\gamma$-cuts
- All queries on exact $\gamma$-cuts are hard (red. from SUBSET SUM)

# Cut transformations

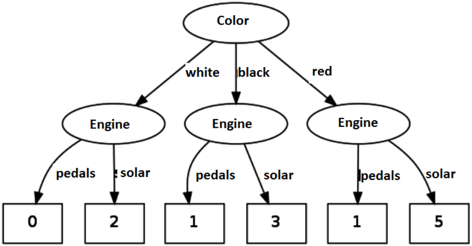
A language L satisfies a transformation if there exists a polynomial algorithm performing it while staying in L

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

- **CUT**$_{\min}$: compute a $L$ representation of the set of cheapest cars
- **CUT**$_{\preceq\gamma}$: compute a $L$ representing the set of cars are cheaper than 10 000 euros
- **CUT**$_{\sim\gamma}$: compute a $L$ representing the set of cars costing exactly 10 000 euros

# Cut transformations

On ADD, $\mathbf{CUT}_{\min}$,$\mathbf{CUT}_{\preceq\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, etc. are trivial:




this is why ADD satisfies all queries related to cuts.

# Cut-based transformations

| Transformation | $\mathtt{ADD}$ | $\mathtt{SLDD}_+$ | $\mathtt{SLDD}_\times$ | $\mathtt{AADD}$ |
|---|---|---|---|---|
| $\mathbf{CUT}_{\min}$ | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| $\mathbf{CUT}_{\sim\gamma}$ | $\surd$ | $\bullet$ | $\bullet$ | $\bullet$ |
| $\mathbf{CUT}_{\preceq\gamma}$ | $\surd$ | $\bullet$ | $\bullet$ | $\bullet$ |

- Cutting to the optimum is easy, even on $\mathtt{SLDD}$ and $\mathtt{AADD}$: 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 **CD** defined as in the Boolean case

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

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

# Map for transformations: combinations

| Transformation | ADD | $\text{SLDD}_+$ | $\text{SLDD}_\times$ | AADD |
|---|---|---|---|---|
| min**C** / +**C** / ×**C** | • | • | • | • |
| min**BC** | √ | • | • | • |
| +**BC** | √ | √ | • | • |
| ×**BC** | √ | • | √ | • |

- ADD satisfies all bounded combinations
  → "apply" algorithm, similar to OBDDs
- $\text{SLDD}_+$ satisfies the combination by $+$
  $\text{SLDD}_\times$ satisfies the combination by $\times$
  → the "apply" algorithm also works because the operators are associative and commutative

# Map for transformations: combinations

| Transformation | ADD | $\mathtt{SLDD}_+$ | $\mathtt{SLDD}_\times$ | AADD |
|---|---|---|---|---|
| min**C** / $+$**C** / $\times$**C** | $\bullet$ | $\bullet$ | $\bullet$ | $\bullet$ |
| min**BC** | $\surd$ | $\bullet$ | $\bullet$ | $\bullet$ |
| $+$**BC** | $\surd$ | $\surd$ | $\bullet$ | $\bullet$ |
| $\times$**BC** | $\surd$ | $\bullet$ | $\surd$ | $\bullet$ |

- $\mathtt{SLDD}_+$ does not satisfy the combination by $\times$: 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 $\mathtt{SLDD}_+$ representation, but $f \times g$ has only exponential $\mathtt{SLDD}_+$ representations
- $\mathtt{SLDD}_\times$ does not satisfy the combination by $+$: similar proof
- AADD does not satisfy any bounded combination.

# Transformations: variable elimination

- $\odot$**Elim**, elimination of variables $Y$ w.r.t. $\odot$ : building a formula in $\mathtt{L}$ representing $\bigodot_{\vec{y}} f_{\alpha}^{\mathtt{L}}|_{\vec{y}}$
  $\rightarrow$ e.g., forgetting $=$ max-elimination

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

# Map for transformations: marginalization

| Transformation | ADD | $\texttt{SLDD}_+$ | $\texttt{SLDD}_\times$ | AADD |
|---|---|---|---|---|
| min**Marg** | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| $+$**Marg** | $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| $\times$**Marg** | $\surd$ | ? | $\surd$ | ? |

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

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

$\rightarrow$ does not work for $\times$**Marg** on $\texttt{SLDD}_+$ and AADD

# Map for transformations: Variable Elimination

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

| Transformation | ADD | $\mathtt{SLDD}_+$ | $\mathtt{SLDD}_\times$ | AADD |
|---|---|---|---|---|
| min**Elim**/ +**Elim** / $\times$**Elim** | • | • | • | • |
| **S**min**Elim** / **S**+**Elim** / **S**$\times$**Elim** | • | • | • | • |
| **SB**max**Elim** / **SB**min**Elim** | $\surd$ | • | • | • |
| **SB**+**Elim** | $\surd$ | $\surd$ | • | • |
| **SB**$\times$**Elim** | $\surd$ | • | $\surd$ | • |

**S**$\odot$**Elim**: eliminating a single variable

**SB**$\odot$**Elim**: eliminating a single bounded-domain variable

# Summary

- Conditionning and Optimization satisfied on AADD,$SLDD_+$, $SLDD_\times$, ADD
- min**BC** satisfied on ADD only
- AADD "more succinct" than $SLDD_+$, $SLDD_\times$, themselves "more succinct" than ADD
- $+$**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 pratical succintness of valued decision diagrams

- Design of a bottom-up ordered $\mathtt{SLDD}_+$ $\mathtt{SLDD}_\times$ compiler.
  - Input: VCSP instance (XML format) or Bayesian Nets (XML format).
  - Output: an equivalent $\mathtt{SLDD}_+$ / $\mathtt{SLDD}_\times$,
- Test of a large set of variable ordering heuristics.
- Design of toolbox of transformation procedures (that are basically normalization procedures)
  - $\mathtt{SLDD}_+$ (resp. $\mathtt{SLDD}_\times$) to $\mathtt{ADD}$
  - $\mathtt{ADD}$ to $\mathtt{SLDD}_+$, $\mathtt{SLDD}_\times$
  - $\mathtt{SLDD}_+$ (resp. $\mathtt{SLDD}_\times$) to $\mathtt{AADD}$

# 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

| | MCF | | *Band-Width* | | MCS | | *Force* | |
|---|---|---|---|---|---|---|---|---|
| Instance | nodes | cpu | nodes | cpu | nodes | cpu | nodes | cpu |
| VCSP $\mapsto$SLDD$_+$ | | | | | | | | |
| Small | 3 100 | 1,2s | 4 349 | 1,0s | 2 344 | 1,0s | 3 415 | 1,2s |
| Medium | 5 660 | 1,5s | 11 700 | 1,6s | 6 242 | 1,4s | 13 603 | 1,5s |
| Big | m-o | - | 326 884 | 112s | 196 098 | 71s | m-o | - |
| Bayes $\mapsto$SLDD$_\times$ | | | | | | | | |
| Asia | 35 | 0,06s | 29 | 0,06s | 23 | 0,06s | 25 | 0,06s |
| Car-starts | 60 | 0,1s | 40 | 0,09s | 40 | 0,09s | 41 | 0,09s |
| Alarm | m-o | - | 5 843 | 0,8s | 1 301 | 0.5 | 7 054 | 1,0s |
| Hail | m-o | - | m-o | - | 15 333 | 1,3s | 139 172 | 114s |
| finder25 | | | | | | | | |

MCS = Maximum Cardinality Search heuristic
[Tarjan and Yannakakis, 1984] in reverse order

# Pratical Succinctness

| | $SLDD_+$ | | ADD | $SLDD_\times$ | AADD |
|---|---|---|---|---|---|
| Instance | nodes | temps | nodes | nodes | nodes |
| Small | 1 744 | 0,9s | 28 971 | 19 930 | 1 744 |
| Medium | 3 238 | 1,3s | 463 383 | 354 122 | 3 156 |
| Big | 73 702 | 34s | m-o | m-o | 73 702 |

| Rés. bay. | $SLDD_\times$ | | ADD | $SLDD_+$ | AADD |
|---|---|---|---|---|---|
| Instance | nodes | temps | nodes | nodes | nodes |
| Asia | 23 | $0,07s$ | 415 | 216 | 23 |
| Car-starts | 40 | $0,1s$ | 42 741 | 19 632 | 40 |
| Alarm | 1 301 | $0,5s$ | m-o | m-o | 1 301 |
| Hailfinder25 | 15 333 | 1,8s | m-o | m-o | 15 331 |

- $\mathtt{AADD}, \mathtt{SLDD}_+, \mathtt{SLDD}_\times < \mathtt{ADD}$;
- $\mathtt{AADD} < \mathtt{SLDD}_+, \mathtt{SLDD}_\times$ but not so much :
  - $\mathtt{AADD}$ and $\mathtt{SLDD}_+$ comparable on additive pricing functions,
  - $\mathtt{AADD}$ and $\mathtt{SLDD}_\times$ comparable on bayesian nets (multiplicative)

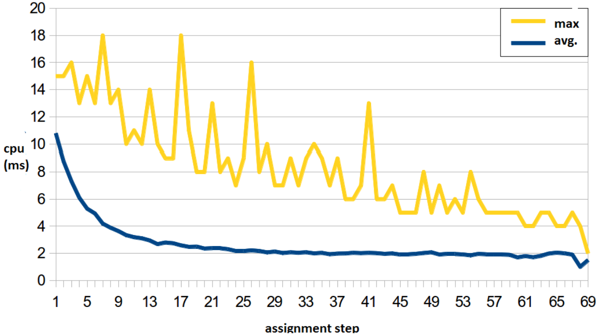
# On line use : CD + marginalization on each variable

| VCSP | $SLDD_+$ | AADD | ratio |
|---|---|---|---|
| Small | 222$\mu$s | 281$\mu$s | 1,27 |
| Medium | 487$\mu$s | 578$\mu$s | 1,19 |
| Big | 22,1ms | 39,9ms | 1,81 |
| **Bayes** | **$SLDD_\times$** | **AADD** | **ratio** |
| Asia | 29,0$\mu$s | 32,3$\mu$s | 1,11 |
| Car-starts | 61,5$\mu$s | 75,6$\mu$s | 1,23 |
| Alarm | 259$\mu$s | 292$\mu$s | 1,13 |
| Hailfinder25 | 7,68ms | 9,16ms | 1,19 |

`SLDD` is more efficient: less number manipulations (`AADD` makes many unsuccessful attempts of saving space), less rounding errors

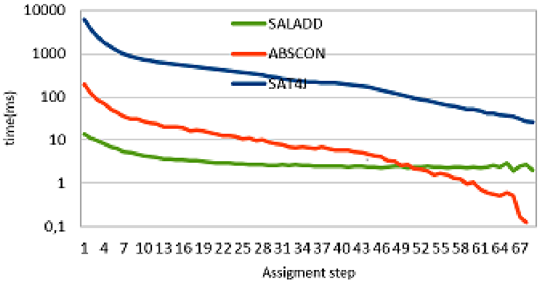
# On line use : CD + marginalization on each variable

Figure: Average and maximal time (ms) for conditionning + marginalization on the **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)

# 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 ($\mathtt{SLDD}_+$ 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 ($\mathtt{SLDD}_+$ 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 ($\texttt{SLDD}_+$ 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 ($\mathtt{SLDD}_+$ 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 : $\mathtt{SLDD}_\times$ , 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 ($\texttt{SLDD}_+$ 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 : $\texttt{SLDD}_\times$ , Bayesian nets, SDDs

# Bibliography


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.

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., and Somenzi, F. (1993).
Algebraic decision diagrams and their applications.
In *Proceedings of ICCAD'93*, pages 188–191.

Darwiche, A. and Marquis, P. (2002).
A knowledge compilation map.
*Journal of Artificial Intelligence Research (JAIR)*, 17:229–264.

Fargier, H., Marquis, P., and Schmidt, N. (2013).
Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues.
In *Proceedings of IJCAI'13*.

Fargier, H., Niveau, A., Marquis, P., and Schmidt, N. (2014).
A knowledge compilation map for ordered real-valued decision diagrams.
In *Proceedings of AAAI'2014*.
Accepté pour publication.

Hadzic, T. (2004).
A bdd-based approach to interactive configuration.
In *Proceedings of CP'04*, page 797.

Kisa, D., den Broeck, G. V., Choi, A., and Darwiche, A. (2014).
Probabilistic sentential decision diagrams.

# Bibliography

[Fargier et al., 2014, Fargier et al., 2013] [Kisa et al., 2014]